KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

CURRICULUM DELIVERY PROGRESS
AUDIT FORM

Faculty Name: Mr. Neil Gogte **Academic year:** 2016-17
Branch: IT **Class:** II

Mid wise split up-of syllabus

Mid-I Units Covered	Periods		Date Of		Remarks
	Required	Taken	Start	Completion	
Unit-I	17	11	14/06/16	25/06/16	
Unit-II	19	20	28/06/16	16/07/16	

Mid-II Units Covered	Periods		Date Of		Remarks
	Required	Taken	Start	Completion	
Unit-III	18	16	19/07/16	20/08/16	
Unit-IV	19	14	23/08/16	29/09/16	
Unit -V	13	10	04/10/16	01/11/16	

Signature of the Faculty **HOD** **Principal**

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Information Technology

Audit Form

Data Structures for Course File

Faculty Name:

SNO	Topic	15/5 Audit 1	Audit 2	Audit 3
1	V / M / PEO / POs /PSOs	✓		
2	Course Structure	✓		
3	Course syllabus	✓		
4	Course Outcomes (CO)	✓		
5	Mapping	✓		
6	Academic Calendar	✓		
7	Time table(class &I individual)	✓		
8	Lesson plan	✓		
9	Topics beyond syllabus (TBS)	✓		
10	Web references	✓		
11	Lecture notes	✓	mention unit numbers	
12	Power point presentations / Videos	✓		
13	University Question papers	✓		
14	Internal Question papers with Key	✓	Keys	2015-16 TL/ COs to be mentioned topics
15	Assignment Question papers	✓		
16	Tutorial evidence	✓		
17	Result Analysis to identify	✓		2015-16 pending

18	Weak and advanced learners			
19	Result Analysis at the end of the course	✓		
20	Course Assessment	✓	2015-16 PO, PSO mapping	
21	Guest talks, field visits etc.			
22	Attendance register	✓		
23	Course file (Digital form)			

Seen 15/5

IQAC Committee In charge

V/M/PEO ①
PO/PSO

V/M/PEO/POs /PSOs

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Vision of the Institution:

To be the fountain head of latest technologies,
producing highly skilled, globally competent engineers.

Mission of the Institution:

- To provide a learning environment that helps students to enhance problem solving skills, be successful in their professional lives and to prepare students to be lifelong learners through multi model platforms and educating them about their professional, and ethical responsibilities.
- To establish Industry Institute Interaction to make students ready for the industry.
- To provide exposure to students to the latest tools and technologies in the area of hardware and software.
- To promote research based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises
- To induce in the students a spirit of nationalism which will enable the student to develop and understand
  India's problems and to encourage them to come up with effective solutions for the same
- To support the faculty in their endeavors to accelerate their learning curve in order to continue to deliver excellent service to students

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Information Technology

Vision & Mission of Department

Vision of the Department:

Producing quality graduates trained in the latest software technologies and related tools and striving to make India a world leader in software products and services.

Mission of the Department:

- Mission of the Department: To create a faculty pool which has a deep understanding and passion for algorithmic thought process.
- To impart skills beyond university prescribed to transform students into a well-rounded IT professional.
- To inculcate an ability in students to pursue Information technology education throughout their lifetime by use of multimodal learning platform including e-learning, blended learning, remote testing and skilling.
- Exposure to different domains, paradigms and exposure to the financial and commercial underpinning of the modern business environment through the entrepreneur development cell.
- To encourage collaboration with various organizations of repute for research, consultancy and industrial interactions.
- To create socially conscious and emotionally mature individuals with awareness on India's challenges, opportunities, their role and responsibility as engineers towards achieving the goal of job and wealth creation.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Information Technology

PROGRAM OUTCOMES (POs)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assesssocietal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Information Technology

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE & Govt of T.S and Affiliated to JNTUH)
3-5-1026, Narayanaguda, Hyderabad-29. Ph: 040-23261407

Department Of Information Technology

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Information Technology solutions for social upliftments.

PSO2: Shall have expertise on the evolving technologies like Mobile Apps, CRM, ERP, Big Data, etc.

Course Structure

Grams: "TECHNOLOGY"
E Mail: dapjntuh@gmail.com

R13

Phone: Off: +91-40-23156115
Fax: +91-40-23158665

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

(Established by Andhra Pradesh Act No.30 of 2008)

Kukatpally, Hyderabad - 500 085, Andhra Pradesh (India)

B. TECH. INFORMATION TECHNOLOGY /COMPUTER SCIENCE AND TECHNOLOGY

I YEAR

Code	Subject	L	T/P/D	C
	English	2	-	4
	Mathematics – I	3	1	6
	Mathematical Methods	3	-	6
	Engineering Physics	3	-	6
	Engineering Chemistry	3	-	6
	Computer Programming	3	-	6
	Engineering Drawing	2	3	6
	Computer Programming Lab	-	3	4
	Engineering Physics / Engineering Chemistry Lab	-	3	4
	English Language Communication Skills Lab	-	3	4
	IT Workshop / Engineering Workshop	-	3	4
	Total	**19**	**16**	**56**

II YEAR I SEMESTER

Code	Subject	L	T/P/D	C
	Probability and Statistics	4	-	4
	Mathematical Foundations of Computer Science	4	-	4
	Data Structures	4	-	4
	Digital Logic Design and Computer Organization	4	-	4
	Electronic Devices and Circuits	4	-	4
	Basic Electrical Engineering	4	-	4
	Electrical and Electronics Lab	-	3	2
	Data Structures Lab	-	3	2
	Total	**24**	**6**	**28**

II YEAR II SEMESTER

Code	Subject	L	T/P/D	C
	Principles of Programming Languages	4	-	4
	Database Management Systems	4	-	4
	Java Programming	4	-	4
	Environmental Studies	4	-	4
	Data Communication	4	-	4
	Design and Analysis of Algorithms	4	-	4
	Java Programming Lab	-	3	2
	Database Management Systems Lab	-	3	2
	Total	**24**	**6**	**28**

III YEAR I SEMESTER

Code	Subject	L	T/P/D	C
	Automata and Compiler Design	4	-	4
	Linux Programming	4	-	4
	Software Engineering	4	-	4
	Operating Systems	4	-	4
	Computer Networks	4	-	4
	Managerial Economics and Financial Analysis	4	-	4
	Operating Systems Lab	-	3	2
	Computer Networks Lab (Through Linux)	-	3	2
	Total	**24**	**6**	**28**

III YEAR II SEMESTER

Code	Subject	L	T/P/D	C
	Web Technologies	4	-	4
	OPEN ELECTIVE Human Values and Professional Ethics Intellectual Property Rights Disaster Management	4	-	4
	Object Oriented Analysis and Design	4	-	4
	Data Warehousing and Data Mining	4	-	4
	Software Testing Methodologies	4	-	4
	Cloud Computing	4	-	4
	Data Mining and Web Technologies Lab	-	3	2
	Advanced English Communication Skills Lab	-	3	2
	Total	**24**	**6**	**28**

IV YEAR I SEMESTER

Code	Subject	L	T/P/D	C
	Information Security	4	-	4
	Design Patterns	4	-	4
	Mobile Application Development	4	-	4
	Information Retrieval Systems	4	-	4
	ELECTIVE – I Wireless Networks and Mobile Computing Image Processing and Pattern Recognition Soft Computing Semantic Web and Social Networks Operations Research	4	-	4
	ELECTIVE – II Software Project Management Computer Graphics Human Computer Interaction Scripting Languages Computer Forensics	4	-	4
	Case Tools and Software Testing Lab	-	3	2
	Mobile Applications Development Lab	-	3	2
	Total	**24**	**6**	**28**

IV YEAR II SEMESTER

Code	Subject	L	T/P/D	C
	Management Science	4	-	4
	ELECTIVE III Web Services E – Commerce Middleware Technologies Ad hoc and Sensor Networks	4	-	4
	ELECTIVE IV Multimedia & Rich Internet Applications Artificial Intelligence Storage Area Networks Machine Learning	4	-	4
	Industry Oriented Mini Project	-	-	2
	Seminar	-	6	2
	Project Work	-	15	10
	Comprehensive Viva	-	-	2
	Total	**12**	**21**	**28**

Note: All End Examinations (Theory and Practical) are of three hours duration.
T Tutorial L – Theory P – Practical/Drawing C Credits

Course (3)
Syllabus

Course Syllabus

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

II Year B.Tech. IT/CST-I Sem	**L**	**T/P/D**	**C**
	4	**-/-/-**	**4**

(A30502) DATA STRUCTURES

Objectives:

- To understand the basic concepts such as Abstract Data Types, Linear and Non Linear Data structures.
- To understand the notations used to analyze the Performance of algorithms.
- To understand the behavior of data structures such as stacks, queues, trees, hash tables, search trees, Graphs and their representations.
- To choose the appropriate data structure for a specified application.
- To understand and analyze various searching and sorting algorithms.
- To write programs in C to solve problems using data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables, search trees.

UNIT- I

Basic concepts- Algorithm Specification-Introduction, Recursive algorithms, Data Abstraction Performance analysis- time complexity and space complexity, Asymptotic Notation-Big O, Omega and Theta notations, Introduction to Linear and Non Linear data structures.

Singly Linked Lists-Operations-Insertion, Deletion, Concatenating singly linked lists, Circularly linked lists-Operations for Circularly linked lists, Doubly Linked Lists- Operations- Insertion, Deletion.

Representation of single, two dimensional arrays, sparse matrices-array and linked representations.

UNIT- II

Stack ADT, definition, operations, array and linked implementations in C, applications-infix to postfix conversion, Postfix expression evaluation, recursion implementation, Queue ADT, definition and operations ,array and linked Implementations in C, Circular queues-Insertion and deletion operations, Deque (Double ended queue)ADT, array and linked implementations in C.

UNIT- III

Trees – Terminology, Representation of Trees, Binary tree ADT, Properties of Binary Trees, Binary Tree Representations-array and linked representations, Binary Tree traversals, Threaded binary trees, Max Priority Queue ADT-implementation-Max Heap-Definition, Insertion into a Max Heap,

Deletion from a Max Heap.

Graphs – Introduction, Definition, Terminology, Graph ADT, Graph Representations- Adjacency matrix, Adjacency lists, Graph traversals- DFS and BFS.

UNIT- IV

Searching- Linear Search, Binary Search, Static Hashing-Introduction, hash tables, hash functions, Overflow Handling.

Sorting-Insertion Sort, Selection Sort, Radix Sort, Quick sort, Heap Sort, Comparison of Sorting methods.

UNIT- V

Search Trees-Binary Search Trees, Definition, Operations- Searching, Insertion and Deletion, AVL Trees-Definition and Examples, Insertion into an AVL Tree ,B-Trees, Definition, B-Tree of order m, operations-Insertion and Searching, Introduction to Red-Black and Splay Trees(Elementary treatment-only Definitions and Examples), Comparison of Search Trees.

Pattern matching algorithm- The Knuth-Morris-Pratt algorithm, Tries (examples only).

TEXT BOOKS:

1. Fundamentals of Data structures in C, 2nd Edition, E.Horowitz, S.Sahni and Susan Anderson-Freed, Universities Press.
2. Data structures A Programming Approach with C, D.S.Kushwaha and A.K.Misra, PHI.

REFERENCE BOOKS:

1. Data structures: A Pseudocode Approach with C, 2nd edition, R.F.Gilberg And B.A.Forouzan, Cengage Learning.
2. Data structures and Algorithm Analysis in C, 2nd edition, M.A.Weiss, Pearson.
3. Data Structures using C, A.M.Tanenbaum,Y. Langsam, M.J.Augenstein, Pearson.
4. Data structures and Program Design in C, 2nd edition, R.Kruse, C.L.Tondo and B.Leung,Pearson.
5. Data Structures and Algorithms made easy in JAVA, 2nd Edition, Narsimha Karumanchi, CareerMonk Publications.
6. Data Structures using C, R.Thareja, Oxford University Press.
7. Data Structures, S.Lipscutz,Schaum's Outlines, TMH.
8. Data structures using C, A.K.Sharma, 2nd edition, Pearson..
9. Data Structures using C &C++, R.Shukla, Wiley India.
10. Classic Data Structures, D.Samanta, 2nd edition, PHI.

Course ④
outcome

Course Outcomes(CO)

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Information Technology

Course Outcomes

Name of the Faculty: Mr. Neil Gogte **Academic Year: 2016-17**

Name of the Subject: Data Structures **Year/ Semester: II/I**

Upon successful completion of the course requirements, a student should be able to:

IT213.CO1: Describe the basic data structures such as arrays, linked lists, stacks and queues. (Level 1)

IT213.CO2: Explain the abstract properties of various data structures such as stacks, queues, lists, trees and graphs. (Level 2)

IT213.CO3: Apply Algorithms for solving problems like sorting, searching, insertion and deletion of data. (Level 3)

IT213.CO4: Use data structure concepts for realistic problems. (Level 3)

IT213.CO5: Analyze performance of algorithms. (Level 4)

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Department of Information Technology

Course Outcomes

Faculty Name:Mr. Neil Gogte **Academic Year: 2015-2016**

Subject: Data Structures **Year/Sem : II/I**

Upon successful completion of the course requirements, a student should be able to:

IT214.CO1: Describe the basic data structures such as arrays &,linked list (Level 1).

IT214.CO2: Explain the abstract properties of various data structures such as stacks,queues,trees and graphs (Level 2).

IT214.CO3: Apply algorithms for solving problems like sorting & searching (Level 3)

IT214.CO4: Use data structures concept for realistic problems (Level 3)

IT214.CO5: Analyze performance of algorithm. (Level 4)

Mapping

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Information Technology

Mapping of CO with PO,PSO

Name of Faculty: Mr Neil Gogte Academic Year: 2016-17

Branch : IT Year: II Semester: I

Subject: Data Structures

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	M		M										M	L
CO2	M	M	M	M									M	H
CO3	M		M										M	M
CO4		L	H										H	M
CO5	M	H	L	M	M								M	M

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Department of Information Technology

Mapping of CO with PO,PSO

Faculty Name:Mr. Neil Gogte **Academic Year: 2015-2016**

Subject: Data Structures **Year/Sem : II/I**

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	Ps01	Ps02
CO1	M		M										M	L
CO2	M	M	M	M									M	H
CO3	M		M										M	M
CO4		L	H										H	M
CO5	M	H	L	M	M								M	M

Academic Calendar

Grams: "TECHNOLOGY"
E Mail: dap@jntuh.ac.in
dapjntuh@gmail.com

Phone: Off: +91–40–23156115
Fax: +91–40–23158665

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

(Established by Andhra Pradesh Act No. 30 of 2008)

Kukatpally, Hyderabad – 500 085, Telangana (India)

Dr. B.N. BHANDARI
Ph.D (IIT KGP).
Professor of Elect. & Commn. Engg., &
Director,
Academic & Planning

Lr.No:A1/ Academic Calendar/B. Tech & B. Pharm./2016 Dated: 10.06.2016

To

The Principals of Constituent Colleges.
The Principals of Affiliated Engineering/Pharmacy colleges of JNTUH.

Sir,

Sub:- JNTUH, Hyderabad – Academic & Planning –Approval of Academic Calendar for **II, III and IV years of B. Tech and B. Pharmacy I & II Semester** for the academic year 2016-17 – Communicated.

***

The Academic Calendar for **II, III and IV years of B. Tech and B. Pharmacy I & II Semester** (Regular) for the academic year 2016-17 is approved. The details are as follows:

I Semester:

Description	Period	Duration
Commencement of Class Work	13.06.2016	
First Spell of Instructions	13.06.2016 to 06.08.2016	(8 w)
First Mid Examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session)02.00 pm to 4.00 pm (Afternoon Session)	08.08.2016 to 13.08.2016	(1 w)
Second Spell instructions	16.08.2016 to 04.10.2016	(7 w)
Dussehra Holidays	05.10.2016 to 12.10.2016	(1 w)
Supplementary Examinations	13.10. 2016 to 26.10.2016	(2w)
Second Spell continuation	27.10.2016 to 03.11.2016	(1 w)

Second Mid Examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session)02.00 pm to 4.00 pm (Afternoon Session)	04.11.2016 to 10.11.2016	(1w)
Preparations and Practical Examinations	11.11.2016 to 17.11.2016	(1w)
End semester Examinations	18.11.2016 to 01.12.2016	(2w)

II Semester

Description	Period	Duration
Commencement of class work	**02.12.2016**	
First Spell of Instructions	02.12.2016.to 27.01.2017	(8 w)
First Mid Examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session)02.00 pm to 4.00 pm (Afternoon Session)	28.01.2017 to 04.02.2017	(1w)
Supplementary Examinations	05.02.2017 to 18.02.2017	(2w)
Second Spell of Instructions	19.02.2017 to 14.04.2017	(8 w)
Second Mid Examinations Timings:10.00 am to 12.00 Noon (Forenoon Session) 02.00 pm to 4.00 pm (Afternoon Session)	15.04.2017 to 21.04.2017	(1w)
Preparation and Practical Examinations	22.04.2017 to 28.04.2017	(1 w)
End semester examinations	29.04.2017 to 12.05.2017	(2 w)
Summer Vacation	13.05.2017 to 11.06.2017	(4w)
Commencement of class work for the next academic year 2016-17	**13.06.2017**	

* **Dussehra holidays from 05.10.2016 to 12.10.2016 may change subject to the directions from the Government of Telangana**

Yours faithfully

DIRECTOR

Copy to:

The Director of Evaluation
The Controller of Examinations.
P.A to VC, Rector and Registrar

E Mail: dap@jntuh.ac.in

Phone: Off: +91–40–23055123
Fax: +91–40–23156115

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

(Established by Andhra Pradesh Act No. 30 of 2008)

Kukatpally, Hyderabad – 500 085, Telangana (India)

Dr. A. Damodaram
B.Tech. (CSE)., M.Tech., (CS) Ph.D (CS).
Professor of Comp. Sc. & Engg., &
Director,
Academic & Planning

Lr.No:A1/Revised Academic Calendar/B. Tech & B. Pharm./2015 Dated:16.10.2015

To
The Principals of Constituent Colleges.
The Principals of Affiliated Engineering/Pharmacy colleges of JNTUH.

Sir,

Sub:- JNTUH, Hyderabad – Academic & Planning –Revised Academic Calendar for **II, III and IV years of B. Tech. and B. Pharmacy** for the academic year 2015-16 – Communicated.

Ref:- (1) Lr. No. A1/Academic Calendar/ B. Tech & B. Pharm./2015, dated 27.06.2015
(2) Resolution of the XXI Standing Committee of the Academic Senate held on 24.09.2015.

***

Based on the recommendations of the Standing Committee of Academic Senate held on 24.09.2015, the academic calendar for **II, III and IV years of B. Tech and B. Pharmacy** of I & II semesters (Regular) are modified as follows, for conducting supplementary examinations of odd semester in the month of February/March and even semester in the month of August /September of every year.

I Semester:

Description	Existing Period	Revised	Duration
Commencement of Class Work	29.06.2015		
First Spell of Instructions	29.06.2015 to22.08.2015		(8 weeks)
First mid examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session) 02.00 pm to 4.00 pm (Afternoon Session)	24.08.2015 to29.08.2015		(1week)
Second Spell of Instructions	31.08.2015 to 17.10.2015		(7 weeks)
*** Dussehra holidays**	19.10.2015 to 24.10.2015		(1week)
Second mid examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session) 02.00 pm to 4.00 pm (Afternoon Session)	26.10.2015 to 31.10.2015		(1week)
Preparations and Practical examinations	02.11.2015 to 07.11.2015		(1 week)
End semester examinations	09.11.2015 to21.11.2015		(2 weeks)
Supplementary examinations	23.11.2015 to 05.12.2015		(2weeks)

II Semester

Description	Existing Period	Revised	Duration
Commencement of class work	**07.12.2015**		
First Spell of Instructions	07.12.2015.to 30.01.2016		(8 weeks)
First mid examinations Timings: 10.00 am to 12.00 Noon (Forenoon Session) 02.00 pm to 4.00 pm (Afternoon Session)	01.02.2016 to 06.02.2016		(1week)
Supplementary Examinations	02.05..2016 to 14.05.2016	**08.02.2016 to 20.02.2016**	**(2weeks-)**
Second Spell of Instructions	08.02.2016 to 02.04.2016	**22.02.2016 to 16.04.2016**	**(8weeks)**
Second mid examinations **Timings: 10.00 am to 12.00** **Noon (Forenoon Session)** **02.00 pm to 4.00 pm (Afternoon Session)**	04.04.2016 to 09.04.2016	**18.04.2016 to 23.04.2016**	**(1week)**
Preparation and Practical Examinations	11.04.2016 to 16.04.2016	**25.04.2016 to 30.04.2016**	**(1 week)**
End semester examinations	18.04.2016 to 30.04.2016	**02.05.2016 to 14.05.2016**	**(2weeks)**
Summer Vacation	16.05.2016 to 11.06.2016	16.05.2016 to 11.06.2016	(4weeks)
Commencement of class work for the next academic year 2016-17	**13.06.2016**		

* **Dussehra holidays from 19.10.2015 to 24 .10.2015 may change subject to the directions from the Government of Telangana**

Yours faithfully

DIRECTOR

Copy to: The Director of Evaluation
The Controller of Examinations.
P.A to VC, Rector and Registrar

⑦
Time Tables

Time Table (Class & Individual)

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanaguda, Hyderabad-29

DEPARTMENT OF INFORMATION TECHNOLOGY

B.Tech II Year I Sem. IT Time Table(2016-17) **W.E.F : 13-06-2016**

TIME/DAY	9.30-10.20 (1)	10.20-11.10 (2)	11.10-11.25	11.25-12.15 (3)	12.15-1.05 (4)	1.05-1.45	1.45-2.35 (5)	2.35-3.25 (6)	3.25-4.15 (7)
MON	P&S	EE LAB				LUNCH	MFCS	EDC/BEE*	DS
TUE	DS	DS	SHORT BREAK	EDC	DLD		P&S	BEE/EDC*	MFCS
WED	DLD	BEE		EDC	P&S		DLD	MFCS/P&S*	EDC
THU	DS	DS			BEE		DLD/DS*	EDC	MFCS
FRI	DLD*	P&S		MFCS	BEE		EDC	BEE	MENTORING& COUNSELLING
SAT	P&S	DS LAB					DS/DLD*	MFCS	P&S/MFCS*

*-TUTORIAL

S.NO	SUBJECT NAME	NAME OF THE FACULTY
1	PROBABILITY AND STATISTICS	MR. U.BALA KRISHNA(H&S)
2	MATHEMATICAL FOUNDATION S OF COMPUTER SCIENCE	DR. TVA. SASTRY(H&S)
3	DATA STRUCTURES THROUGH C	MR.NEIL GOGTE(IT)
4	DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION	MR. P. BALAKRISHNA(ECE)
5	ELECTRONIC DEVICES AND CIRCUITS	MS.PRABHAVATHI(ECE)
6	BASIC ELECTRICAL ENGINEERING	MRS. AMBIKA(ECE)
7	ELECTRICAL AND ELECTRONICS LAB (LAB -B2 & LAB- C)	MRS. AMBIKA/ MS.PRABHAVATHI (ECE)
8	DATA STRUCTURES LAB(FS-3)	MR.NEIL GOGTE(IT)/MS.BHAGYA SREE/MS.SUNITHA PATIDAR(IT)
9	MENTORING & COUNSELLING	MS.SUNITHA PATIDAR/ MS.NAGA MAHA LAKSHMI /MR. RAKESH

CLASS-INCHARGE **HOD** **PRINCIPAL**

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanaguda, Hyderabad-29

DEPARTMENT OF INFORMATION TECHNOLOGY

W.E.F : 13-06-2016

INDIVIDUAL TIME TABLE

Faculty Name : Neil Gogte

TIME/DAY	9.30-10.20 (1)	10.20-11.10 (2)	11.10-11.25	11.25-12.15 (3)	12.15-1.05 (4)	1.05-1.45	1.45-2.35 (5)	2.35-3.25 (6)	3.25-4.15 (7)
MON						LUNCH			DS
TUE	DS	DS	SHORT BREAK						
WED									
THU	DS	DS							
FRI									
SAT		DS LAB					DS/DLD*		

CLASS-INCHARGE HOD PRINCIPAL

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanaguda, Hyderabad-29

DEPARTMENT OF INFORMATION TECHNOLOGY

B.Tech II- Year IT I Sem. Time Table (2015-16)

W.E.F : 29-06-2015

TIME/DAY	9.30-10.20 (1)	10.20-11.10 (2)	11.10-11.25	11.25-12.15 (3)	12.15-1.05 (4)	1.05-1.45	1.45-2.35 (5)	2.35-3.25 (6)	3.25-4.15 (7)
MON	P&S	EE LAB				LUNCH	MFCS	*EDC/BEE	DS
TUE	DS	DS	SHORT BREAK	EDC	DLD		P&S	*BEE/EDC	MFCS
WED	DLD	BEE		EDC	P&S		DLD	*MFCS/P&S	EDC
THU	DS	DS			BEE		DLD/DS	EDC	MFCS
FRI	DLD	P&S		MFCS	BEE		EDC	BEE	MENTORING& COUNSELLING
SAT	P&S	DS LAB					*DS/DLD	MFCS	*P&S/MFCS

*-TUTORIAL

SL.NO	SUBJECT	FACULTY
1	PROBABILITY AND STATISTICS	MR. U.BALA KRISHNA(H&S)
2	MATHEMATICAL FOUNDATION S OF COMPUTER SCIENCE	DR. TVA. SASTRY(H&S)
3	DATA STRUCTURES THROUGH C	MR.NEIL GOGTE(IT)
4	DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION	MR. P. BALAKRISHNA(ECE)
5	ELECTRONIC DEVICES AND CIRCUITS	MS.PRABHAVATHI(ECE)
6	BASIC ELECTRICAL ENGINEERING	MRS. AMBIKA(ECE)
7	ELECTRICAL AND ELECTRONICS LAB (LAB -B2 & LAB- C)	MRS. AMBIKA/ MS.PRABHAVATHI (ECE)
8	DATA STRUCTURES LAB(FS-3)	MR.NEIL GOGTE(IT)/MS.BHAGYA SREE/MS.SUNITHA PATIDAR(IT)
9	MENTORING & COUNSELLING	MS.BHAGYA SREE/ MS.SUNITHA PATIDAR/ MS.LEELAVATHI

CLASS-INCHARGE **HOD** **PRINCIPAL**

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanaguda, Hyderabad-29

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.Tech II Year IT I Sem. Time Table

(for the year 2015-16)

INDIVIDUAL TIMETABLE : Mr. Neil Gogte

TIME/DAY	9.30-10.20	10.20-11.10	11.10-11.25	11.25-12.15	12.15-01.05	01.05-01.45	01.45-02.35	02.35-03.25	03.25-04.15
	1	2		3	4		5	6	7
MON			SHORT BREAK			LUNCH BREAK			DS
TUE	DS	DS							
WED									
THU	DS	DS							
FRI									
SAT	DS LAB						*DS(TUTORIAL)		

COPY TO: The Director
The HOD/ Co-ordinator Faculty
The Concerned Faculty
The Lab Incharges

PRINCIPAL

Lesson Plan ⑧

Lesson Plan

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

SUBJECT CODE	NAME OF THE SUBJECT	CLASS /SEM	FACULTY NAME/DESIGNATION	NO. OF STUENTS	TOTAL PERIODS	
					LECTURE	TUTORIAL
113BP	DATA STRUCTURES	II/I	Mr.Neil Gogte	60	78	8

Week No.	Unit No.	Lecture No.	Topic	Date	Texbook /References	Teaching Method
1	1	1	Algorithm Specification-Introduction	13-Jun-16	T1	Black Board
		2	Recursive algorithms,	14-Jun-16	T1	Black Board
		3	Towers of Hanoi problem	14-Jun-16	T1	Black Board
		4	time complexity and space complexity	16-Jun-16	T1	Black Board
		5	Asymptotic Notation-Big O, Omega & Theta	16-Jun-16	T1	Black Board
		6	TUTORIAL : Asymptotic notations	18-Jun-16	T1	Black Board
2		7	Introduction to Linear and Non Linear data structur	20-Jun-16	T1	Black Board
		8	Singly Linked Lists	21-Jun-16	T1/T2	Black Board
		9	Insertion, Deletion operations	21-Jun-16	T1	Black Board
		10	Concatenating singly linked lists	23-Jun-16	T1	Black Board
3		12	Circularly linked lists	27-Jun-16	T1	Black Board
		13	Operations for Circularly linked lists	28-Jun-16	T1	Black Board
		14	Doubly Linked Lists,insertion	28-Jun-16	T1	Black Board
		15	deletion operation,searching.	30-Jun-16	T1	Black Board
		16	sparse matrices-array and linked representations.	30-Jun-16	T1	Black Board
		17	TUTORIAL: Linked lists	2-Jul-16	T1	Black Board
4	2	18	Stack ADT, definition, operations	4-Jul-16	T1/T2	PPT
		19	array implementations in C	5-Jul-16	T1	PPT
		20	linked implementation	5-Jul-16	T1/T2	PPT
		21	applications-infix to postfix conversion,	7-Jul-16	T1	Black Board
		22	Postfix expression evaluation	7-Jul-16	T1	Black Board
5		23	recursion implementation	11-Jul-16	T1	Black Board
		24	Queue ADT, definition and operations	12-Jul-16	T1	Black Board
		25	array Implementations in C,	12-Jul-16	T1/T2	Black Board
		26	linked Implementations in C,	14-Jul-16	T1/T2	Black Board
		27	TUTORIAL Stacks and Queue	16-Jul-16	T1	Black Board
6		28	Circular queues	18-Jul-16	T1	Black Board
		29	Insertion Operation	19-Jul-16	T1	Black Board
		30	deletion operations	19-Jul-16	T1	Black Board
		31	Deque ADT	21-Jul-16	T1	Black Board
		32	array implementation	21-Jul-16	T1	Black Board
7		34	linked implementations in C.	25-Jul-16	T1/T2	Black Board
		35	linked implementations in C.	26-Jul-16	T1/T2	Black Board
		36	Review of stack and queue	26-Jul-16	T1	Black Board
	3	37	Trees – Terminology,	28-Jul-16	T1	PPT
		38	Representation of Trees	28-Jul-16	T1	PPT
		39	TUTORIAL Circular queue	30-Jul-16	T1	Black Board
8		40	Binary tree ADT	1-Aug-16	T1	PPT
		41	Properties of Binary Trees	2-Aug-16	T1	PPT
		42	Binary Tree Representations	2-Aug-16	T1	PPT
		43	Array and linked representations	4-Aug-16	T1	PPT
		44	Binary Tree traversals,	4-Aug-16	T1/T2	Black Board

9	3	45	Threaded binary trees	16-Aug-16	T1	Black Board
		46	Max Priority Queue ADT	16-Aug-16	T1	Black Board
		47	Max Heap-Definition	18-Aug-16	T1	Black Board
		48	Insertion into a Max Heap	18-Aug-16	T1	Black Board
		49	TUTORIAL Binary trees	20-Aug-16	T1	Black Board
10		50	Deletion from a Max Heap.	22-Aug-16	T1	Black Board
		51	Adjacency lists	23-Aug-16	T1	Black Board
		52	Graph intro, terminoklogy	23-Aug-16	T1	PPT
11		54	Traversals DFS	29-Aug-16	T1/T2	PPT
	4	55	BFS.	30-Aug-16	T1/T2	Black Board
		56	introduction to searching techniques	30-Aug-16	T1	Black Board
		57	Linerar search	1-Sep-16	T1	Black Board
		58	Binary Search	1-Sep-16	T1	Black Board
		59	TUTORIAL Graphp traversals	3-Sep-16	T1	Black Board
12		60	static hashing	6-Sep-16	T1	Black Board
		61	hash tables and hash functions	8-Sep-16	T1	Black Board
		62	overflow handling	8-Sep-16	T1	Black Board
13		63	SORTING: bubble sort	13-Sep-16	T1	PPT
		64	selection sort	13-Sep-16	T1	PPT
		65	Merge sort	15-Sep-16	T1	PPT
		66	quick sort	15-Sep-16	T1	PPT
		67	radix sort	15-Sep-16	T1	PPT
		68	TUTORIAL Hashing	17-Sep-16	T1	PPT
14		69	radix sort program	19-Sep-16	T1	PPT
		70	heap sort	20-Sep-16	T1	PPT
		71	heap sort	20-Sep-16	T1	PPT
		72	shell sort	22-Sep-16	T1	Black Board
		73	comparision of sorting algorithms	22-Sep-16	T1	Black Board
15	5	75	Search Trees-Binary Search Trees	26-Sep-16	T1	Black Board
		76	Definition, Operations- Searching	27-Sep-16	T1	Black Board
		77	Insertion and Deletion	29-Sep-16	T1	Black Board
		78	AVL Trees- creation, insertion	29-Sep-16	T1	Black Board
		79	TUTORIAL Search Trees	1-Oct-16	T1	Black Board
16		80	AVL Tree deletion	3-Oct-16	T1/T2	Black Board
		81	B-Trees, Definition, B-Tree of order m	4-Oct-16	T1	Black Board
		82	operations-Insertion and Searching,	4-Oct-16	T1	Black Board
17		83	Red black trees, splay trees.	27-Oct-16	T1	Black Board
		84	B+ trees definition.	27-Oct-16	T1	Black Board
		85	Brute force pattern matching algorithm	29-Oct-16	T1	Black Board
		86	KMP algorithm	31-Oct-16	T1	PPT

TEXT BOOKS:

T1. Fundamentals of Data structures in C, 2nd Edition, E.Horowitz, S.Sahni and Susan Anderson-Freed, Universities Press.

T2. Data structures A Programming Approach with C, D.S.Kushwaha and A.K.Misra, PHI.

REFERENCE BOOKS:

R1. Data structures: A Pseudocode Approach with C, 2nd edition, R.F.Gilberg And B.A.Forouzan, Cengage Learning.

R2. Data structures and Algorithm Analysis in C, 2nd edition, M.A.Weiss, Pearson.

Factult signature

date:

HOD signature

date:

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

Subject Code	Name of the Subject	Class/Sem	Name of the Faculty / Designation	Number of Students	Total Proposed Periods per semester/year	
	Data Structures	IT II/I	Mr.Neil Gogte	59	Lectures	Tutorial
					65	11

Week Number	Lecture Number	Topic	Date of completion	Textbook/ Refernces	Teaching aid/methdology
W1	1	Introduction to Algorithm	29/6/2015	T1	Black Board
	2	Characteristics of Algorithm	30/6/2015	T1	Black Board
	3	Recursive algorithms, Tower of Hanoi	30/6/2015	T1	Black Board
	4	Performance analysis- time complexity and space complexity,	2/7/2015	T1	Black Board
	5	Introduction to Asymptotic Notation Big O Notation,	2/7/2015	T1	Black Board
	6	TUTORIAL 1: Recursion and Arrays.	4/7/2015	T1	Black Board
W2	7	Omega and Theta notations,	6/7/2015	T1	Black Board
	8	Introduction to Linear and Non Linear data structures and Types of Linked List	9/7/2015	T1/T2	Black Board
	9	Singly Linked Lists- Operation Insertion	9/7/2015	T1	Black Board
	10	TUTORIAL 2 : Reverse a linked list, Find Length of a Linked List (Recursive)	11/7/2015	T1	Black Board
W3	11	Singly Linked Lists-Operation Deletion	13/7/2015	T1	Black Board
	12	Traversing a linked List	14/7/2015	T1	Black Board
	13	Inserting and Deleting from the middle of the Link List.	14/7/2015	T1	Black Board

Signature of the HOD
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

	14	Concatenating singly linked list.	16/7/2015	T1	Black Board
	15	TUTORIAL 3: Merge two sorted linked lists	16/7/2015	T1	Black Board
W4	16	Introduction ,Operations for Circularly linked lists,	20/7/2015	T1	Black Board
	17	Introduction to Doubly Linked List	21/7/2015	T1	Black Board
	18	Operations- Insertion & Deletion.	21/7/2015	T1	Black Board
	19	Sparse matrices-array & linked representation.	23/7/2015	T1	Black Board
	20	TUTORIAL 4: program to implement sparse matrix using linked list.	25/7/2015	T1	Black Board
W5	21	Introduction to Stack ADT. operations,	27/7/2015	T1/T2	PPT
	22	Stack: array & linked implementations in C,	28/7/2015	T1	PPT
	24	Applications-infix to postfix conversion.	28/7/2015	T1/T2	Black Board
	25	Postfix expression evaluation.	30/7/2015	T1	Black Board
	26	TUTORIAL 5: Towers of Hanoi problem	1/8/2015	T1	Black Board
W6	27	Introduction to Queue ADT, operations,	4/8/2015	T1/T2	Black Board
	28	Queue: array & linked implementations in C,	6/8/2015	T1/T2	Black Board
	29	TUTORIAL 6: Implement a stack using two queues, Implement a queue using two stacks.	8/8/2015	T1	Black Board
	30	Circular queues-Insertion and deletion operations.	10/8/2015	T1	Black Board

Signature of the HOD **Signature of the Faculty**

Date Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

W7	31	Introduction to Deque (Double ended queue) ADT,	11/8/2015	T1	Black Board
	32	Deque :array implementations in C.	11/8/2015	T1	Black Board
	33	Deque :Linked implementations in C.	13/8/2015	T1	Black Board
W8	34	More examples on infix,prefix and postfix expressions	13/8/2015	T1	Black Board
	35	Introduction to Trees and there Terminology,	17/8/2015	T1/T2	PPT
	36	Representation of Trees, Introduction to Binary Trees, Properties of Binary Trees,	18/8/2015	T1	PPT
	37	Representations-array and linked representations.	18/8/2015	T1	PPT
W9	38	Binary Tree traversals: preorder, inorder and postorder	20/8/2015	T1	PPT
	39	Threaded binary trees,	20/8/2015	T1	Black Board
	40	Review Unit 3	21/8/2015	T1	Black Board
	41	TUTORIAL 7: program to find the Total Nodes and Total Leaf Nodes of Binary Tree	22/8/2015	T1	Black Board
W10	42	Introduction to Max Priority Queue ADT	1/9/2015	T1	Black Board
	43	Implementation of Max Heap	1/9/2015	T1	Black Board
	44	Insertion into a Max Heap,	3/9/2015	T1	Black Board
	45	Deletion from a Max Heap.	7/9/2015	T1	Black Board
	46	Introduction to Graphs, Terminology,	8/9/2015	T1	PPT

Signature of the HOD
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

W11	47	Graph Representations- Adjacency matrix,	8/9/2015	T1	PPT
	48	Graph Representations- Adjacency List	10/9/2015	T1	PPT
	49	TUTORIAL -8 Graph traversals- BFS	12/9/2015	T1/T2	PPT
W12	50	Introduction to Searching- Linear Search,	14/9/2015	T1	Black Board /ppt
	51	Binary Search: Non Rec and Rec	15/9/2015	T1	Black Board
	52	Static Hashing-Introduction,	15/9/2015	T1	Black Board
	53	hash tables and hash functions,	19/9/2015	T1	Black Board
W13	54	hash functions cont. Overflow Handling.	21/9/2015	T1	**Black Board**
	55	Introduction to Sorting- Selection Sort, Insertion Sort	22/9/2015	T1	Black Board
	56	TUTORIAL 9: Merge Sort, Shell sort	26/9/2015	T1	Black Board
	57	Quick sort,	28/9/2015	T1	PPT
	58	Heap Sort,	29/19/2015	T1	PPT
W14	59	Radix Sort, Comparison of Sorting methods.	29/9/2015	T1	PPT
	60	Review Unit 4	30/10/2015	T1	Black Board
W15	61	Introduction to Search Trees. Binary Search Tree.	1/10/2015	T1	Black Board
	62	BST Operations- Searching, Insertion, Deletion.	1/10/2015	T1	Black Board
	63	TUTORIAL 10: program to find the height of a Binary search Tree	3/10/2015	T1	Black Board

Signature of the HOD **Signature of the Faculty**

Date Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

	64	Introduction to AVL Trees. Balance factor	5/10/2015	T1	Black Board
	65	Insertion and deletion into an AVL Tree	6/10/2015	T1/T2	Black Board
	66	Introduction to B-Trees operations-Insertion, deletion and Searching	6/10/2015	T1	Black Board
W16	67	Introduction to Red-Black.	8/10/2015	T1/T2	Black Board
	68	Introduction Splay Trees	8/10/2015	T1	Black Board
	69	Comparison of Search Trees.	9/10/2015	T1	Black Board
	70	Pattern matching algorithm The Knuth-Morris-Pratt algorithm	9/10/2015	T1	PPT
	71	TUTORIAL 11 : Brute Force algorithm	10/10/2015	T1	Black Board
W17	72	Introduction to Tries	12/10/2015	T1	Black Board
	73	Example program on BST insertion	13/10/2015	T1	Black Board
	74	examples on AVL insertion	13/10/2015	T1	Black Board
	75	Example on B-Tree insertion	15/10/2015	T1	Black Board
	76	B+ trees introduction	15/10/2015	T1	Black Board

Signature of the HOD
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

Topics (9) Beyond Syllabus

Topics Beyond Syllabus (TBS)

TOPICS BEYOND SYLLABUS

CONTENTS

1. Tower of Hanoi Problem.
2. Merge Sort & Shell Sort
3. Deleting from AVL Tree
4. B^+ Trees
5. Brute Force Pattern Matching Algorithm.

```
}
```

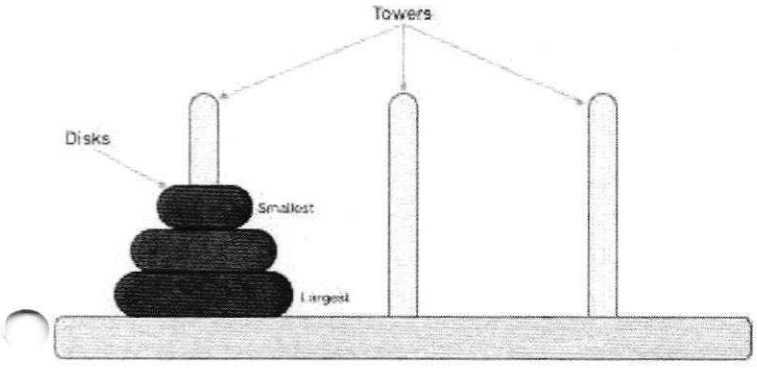
TOPICS BEYOND SYLLABUS (TBS)

S.NO	TOPIC
1	Towers of Hanoi problem
2	Merge Sort, Shell sort
3	AVL Deletion
4	B+ trees
5	Brute Force algorithm

Data Structure & Algorithms - Tower of Hanoi

https://www.tutorialspoint.com/data_structures_algorithms/tower_of_hanoi.htm

Tower of Hanoi, is a mathematical puzzle which consists of three towers *pegs* and more than one rings is as depicted −



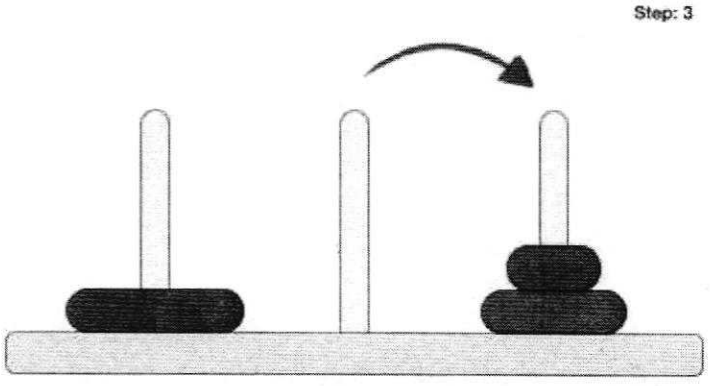
These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Following is an animated representation of solving a Tower of Hanoi puzzle with three disks.



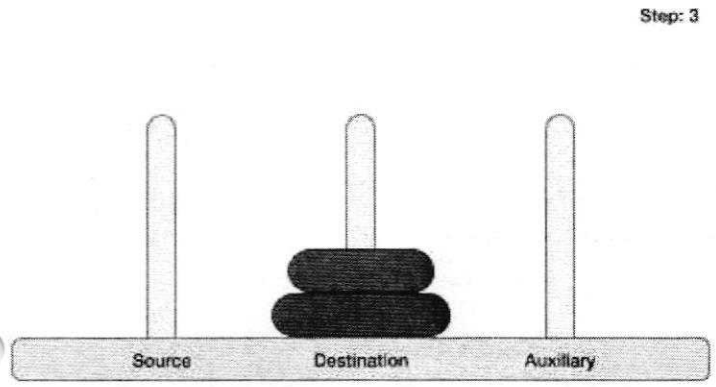
Tower of Hanoi puzzle with n disks can be solved in minimum 2^n-1 steps. This presentation shows that a puzzle with 3 disks has taken $2^3 - 1 = 7$ steps.

Algorithm

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, **source**, **destination** and **aux** *onlytohelpmovingthedisks*. If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks −

- First, we move the smaller *top* disk to aux peg.
- Then, we move the larger *bottom* disk to destination peg.
- And finally, we move the smaller disk from aux to destination peg.



So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (n^{th} disk) is in one part and all other $n - 1$ disks are in the second part.

Our ultimate aim is to move disk **n** from source to destination and then put all other $n1$ disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.

The steps to follow are −

Step 1 - Move n-1 disks from **source** to **aux**
Step 2 - Move n^{th} disk from **source** to **dest**
Step 3 - Move n-1 disks from **aux** to **dest**

A recursive algorithm for Tower of Hanoi can be driven as follows −

```
START
Procedure Hanoi(disk, source, dest, aux)

   IF disk == 0, THEN
      move disk from source to dest
   ELSE
      Hanoi(disk - 1, source, aux, dest)     // Step 1
      move disk from source to dest          // Step 2
      Hanoi(disk - 1, aux, dest, source)     // Step 3
   END IF

END Procedure
STOP
```

To check the implementation in C programming, click here.

Data Structures - Merge Sort Algorithm

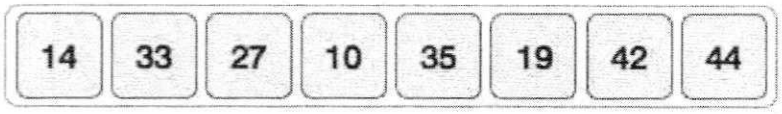
https://www.tutorialspoint.com/data_structures_algorithms/merge_sort_algorithm.htm

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being $Onlogn$, it is one of the most respected algorithms.

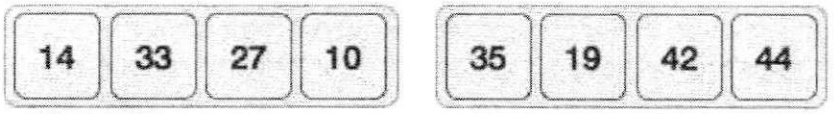
Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How Merge Sort Works?

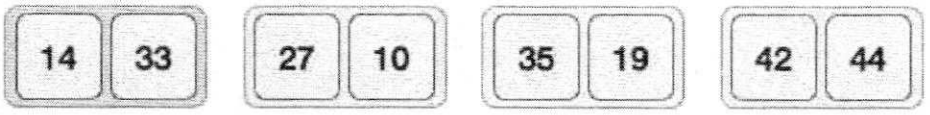
To understand merge sort, we take an unsorted array as the following –



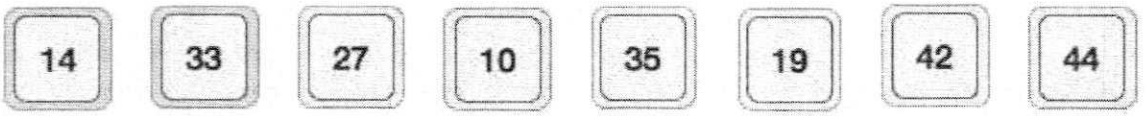
We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.



This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

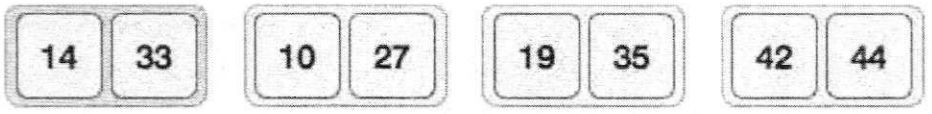


We further divide these arrays and we achieve atomic value which can no more be divided.

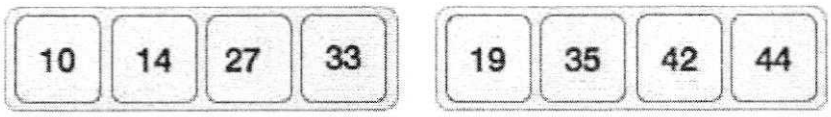


Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

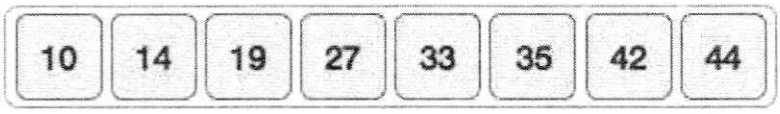
We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.



After the final merging, the list should look like this −



Now we should learn some programming aspects of merge sorting.

Algorithm

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

```
Step 1 - if it is only one element in the list it is already sorted, return.
Step 2 - divide the list recursively into two halves until it can no more be divided.
Step 3 - merge the smaller lists into new list in sorted order.
```

Pseudocode

We shall now see the pseudocodes for merge sort functions. As our algorithms point out two main functions − divide & merge.

Merge sort works with recursion and we shall see our implementation in the same way.

```
procedure mergesort( var a as array )
   if ( n == 1 ) return a

   var l1 as array = a[0] ... a[n/2]
   var l2 as array = a[n/2+1] ... a[n]

   l1 = mergesort( l1 )
   l2 = mergesort( l2 )

   return merge( l1, l2 )
end procedure

procedure merge( var a as array, var b as array )

   var c as array

   while ( a and b have elements )
      if ( a[0] > b[0] )
         add b[0] to the end of c
         remove b[0] from b
      else
         add a[0] to the end of c
         remove a[0] from a
      end if
   end while

   while ( a has elements )
      add a[0] to the end of c
      remove a[0] from a
   end while

   while ( b has elements )
      add b[0] to the end of c
      remove b[0] from b
```

```
   end while

   return c

end procedure
```

To know about merge sort implementation in C programming language, please click here.

Data Structure and Algorithms - Shell Sort

https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as **interval**. This interval is calculated based on Knuth's formula as −

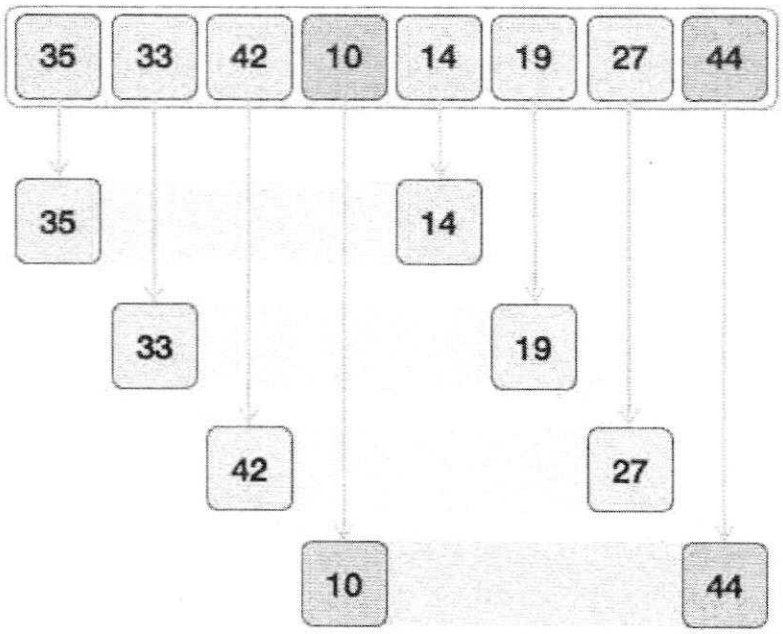
Knuth's Formula

```
h = h * 3 + 1

where -
   h is interval with initial value 1
```

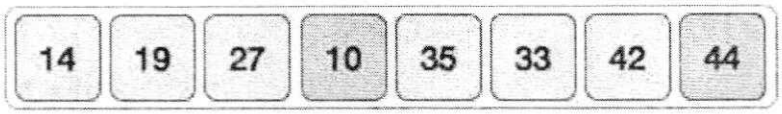
This algorithm is quite efficient for medium-sized data sets as its average and worst case complexity are of O n, where **n** is the number of items.

How Shell Sort Works?

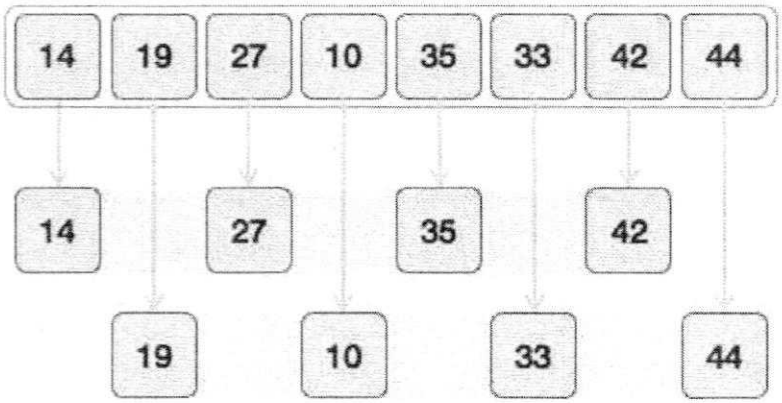
Let us consider the following example to have an idea of how shell sort works. We take the same array we have used in our previous examples. For our example and ease of understanding, we take the interval of 4. Make a virtual sub-list of all values located at the interval of 4 positions. Here these values are {35, 14}, {33, 19}, {42, 27} and {10, 44}



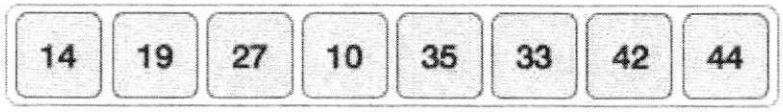
We compare values in each sub-list and swap them $if necessary$ in the original array. After this step, the new array should look like this −



Then, we take interval of 2 and this gap generates two sub-lists - {14, 27, 35, 42}, {19, 10, 33, 44}

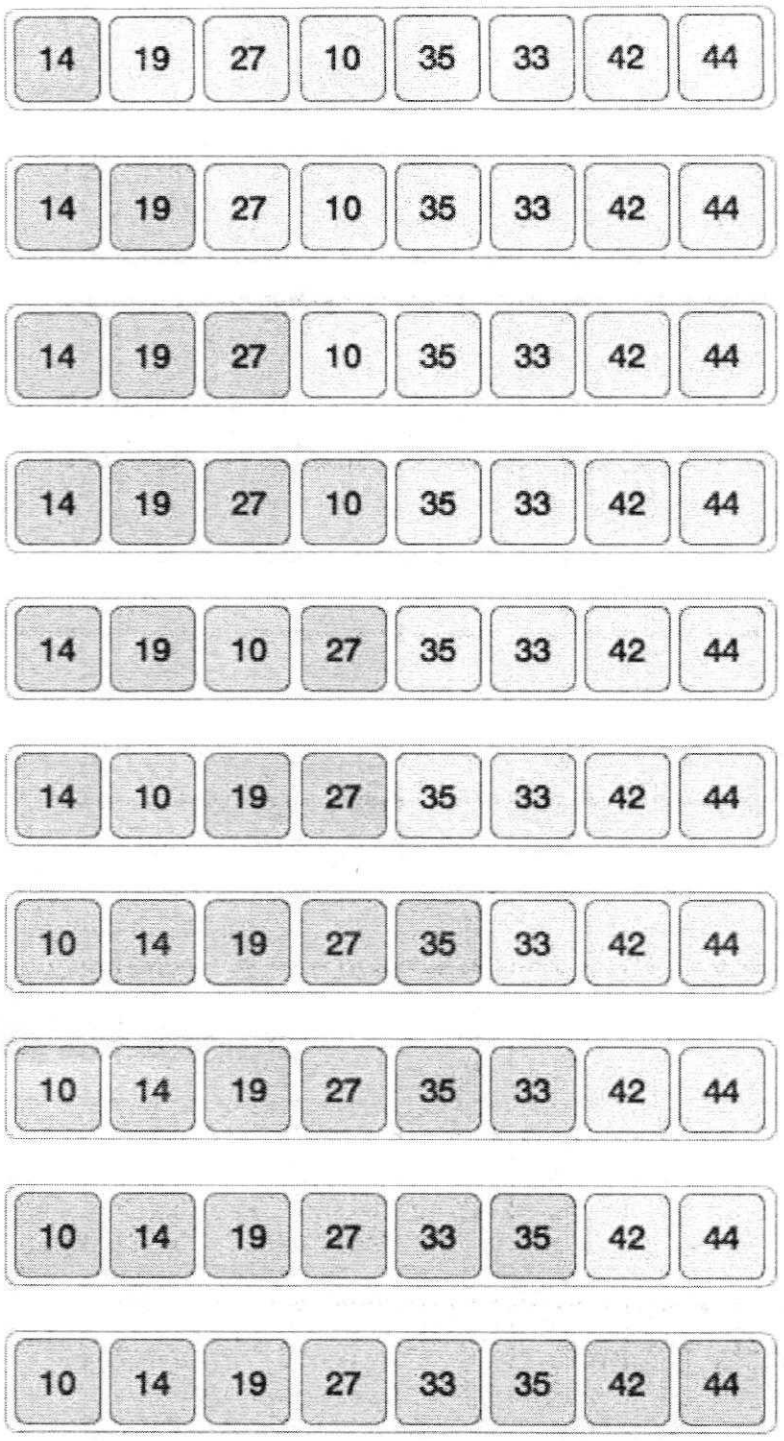


We compare and swap the values, if required, in the original array. After this step, the array should look like this −



Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.

Following is the step-by-step depiction −



We see that it required only four swaps to sort the rest of the array.

Algorithm

Following is the algorithm for shell sort.

```
Step 1 - Initialize the value of h
Step 2 - Divide the list into smaller sub-list of equal interval h
Step 3 - Sort these sub-lists using insertion sort
Step 3 - Repeat until complete list is sorted
```

Pseudocode

Following is the pseudocode for shell sort.

```
procedure shellSort()
   A : array of items
```

```
   /* calculate interval*/
   while interval < A.length /3 do:
      interval = interval * 3 + 1
   end while

   while interval > 0 do:

      for outer = interval; outer < A.length; outer ++ do:

      /* select value to be inserted */
      valueToInsert = A[outer]
      inner = outer;

         /*shift element towards right*/
         while inner > interval -1 && A[inner - interval] >= valueToInsert do:
            A[inner] = A[inner - interval]
            inner = inner - interval
         end while

      /* insert the number at hole position */
      A[inner] = valueToInsert

      end for

   /* calculate interval*/
   interval = (interval -1) /3;

   end while

end procedure
```

To know about shell sort implementation in C programming language, please click here.

Deletion from an AVL Tree

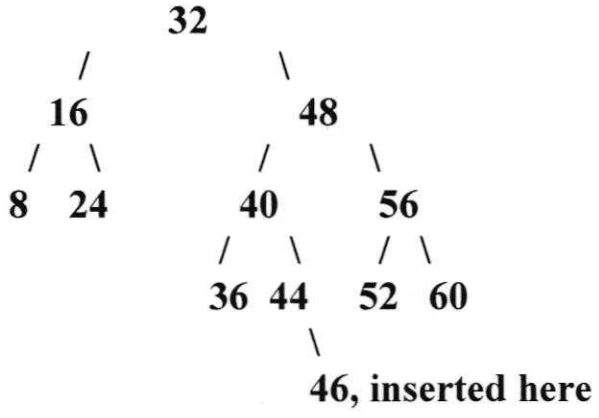
First we will do a normal binary search tree delete. Note that structurally speaking, all deletes from a binary search tree delete nodes with zero or one child. For deleted leaf nodes, clearly the heights of the children of the node do not change. Also, the heights of the children of a deleted node with one child do not change either. Thus, if a delete causes a violation of the AVL Tree height property, this would HAVE to occur on some node on the path from the parent of the deleted node to the root node.

Thus, once again, as above, to restructure the tree after a delete we will call the restructure method on the parent of the deleted node. One thing to node: whereas in an insert there is at most one node that needs to be unbalanced, there may be multiple nodes in the delete that need to be rebalanced. Technically speaking, at any point in the restructuring algorithm ONLY one node will ever be unbalanced. But, what may happen is when that node is fixed, it may propogate an error to an ancestor node. But, this is NOT a problem because our restructuring algorithm goes all the way to the root node. F

Let's trace through a couple examples each for inserts and deletes, using the code handout.

AVL Tree Examples

1) *Consider inserting 46 into the following AVL Tree:*



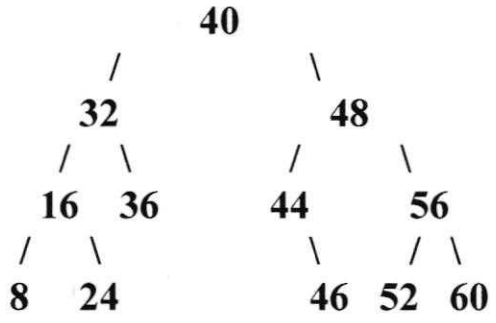
Initially, using the standard binary search tree insert, 46 would go to the right of 44. Now, let's trace through the rebalancing process from this place.

First, we call the method on this node. Once we set its height, we check to see if the node is balanced. (This simply looks up the heights of the left and right subtrees, and decides if the difference is more than 1.) In this case, the node is balanced, so we march up to the parent node, that stores 44.

We will trace through the same steps here, setting the new height of this node (this is important!) and determining that this node is balanced, since its left subtree has a height of -1 and the right subtree has a height of 0.

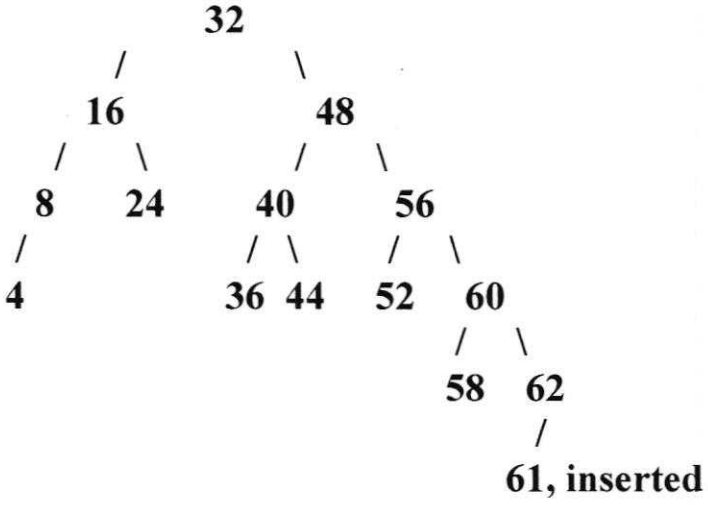
Similarly, we set the height and decide that the nodes storing 40 and 48 are balanced as well. Finally, when we reach the root node storing 32, we realize that our tree is imbalanced.

Now, we finally get to execute the code inside the if statement in the rebalance method. Here we set xPos to be the tallest grandchild of the root node. (This is the node storing 40, since its height is 2.) Thus, the restructuring occurs on the nodes containing the 32, 48 and 40. Using the method described from last lecture, we will restructure the tree as follows:

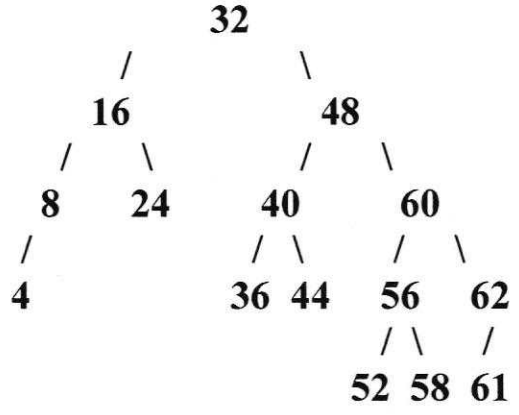


Using the variables from the last lecture, the node storing 40 is B, the node storing 32 is A, and the node storing 48 is C. T_0 is the subtree rooted at 16, T_1 is the subtree rooted at 36, T_2 is the subtree rooted at 44, and T_3 is the subtree rooted at 56.

2) ***Now, for the second example, consider inserting 61 into the following AVL Tree:***

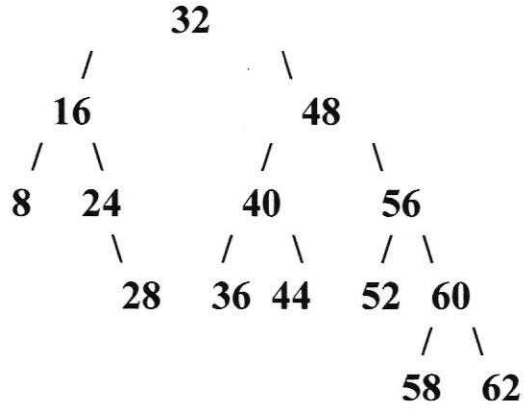


Tracing through the code, we find the first place an imbalance occurs tracing up the ancestry of the node storing 61 is at the noce storing 56. This time, we have that node A stores 56, node B stores 60, and node C stores 62. Using our restucturing algorithm, we find the tallest grandchild of 56 to be 62, and rearrange the tree as follows:



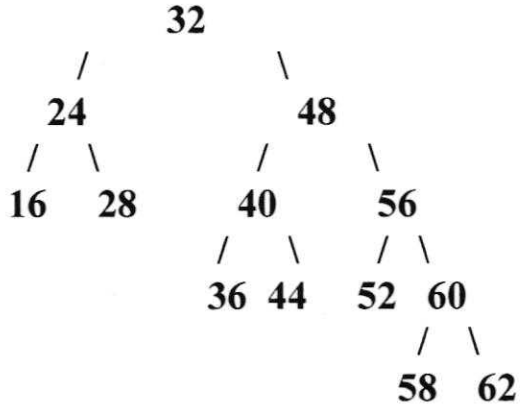
T_0 is the subtree rooted at 52, T_1 is the subtree rooted at 58, T_2 is the subtree rooted at 61, and T_3 is a null subtree.

3) ***For this example, we will delete the node storing 8 from the AVL tree below:***



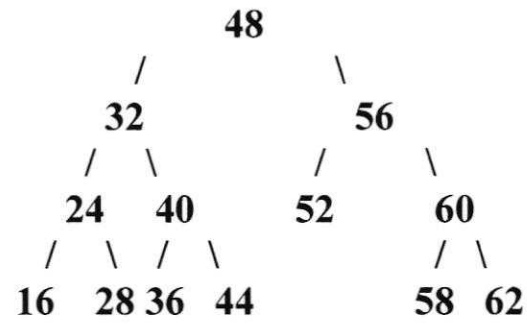
Tracing through the code, we find that we must first call the rebalance method on the parent of the deleted node, which

stores 16. This node needs rebalancing and gets restructured as follows:

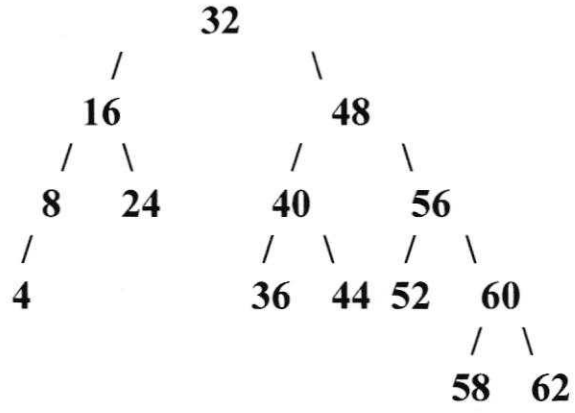


Notice that all four subtrees for this restructuring are null, and we only use the nodes A, B, and C. Next, we march up to the parent of the node storing 24, the node storing 32. Once again, this node is imbalanced. The reason for this is that the restructuring of the node with a 16 reduced the height of that subtree. By doing so, there was in INCREASE in the difference of height between the subtrees of the old parent of the node storing 16. This increase could propogate an imbalance in the AVL tree.

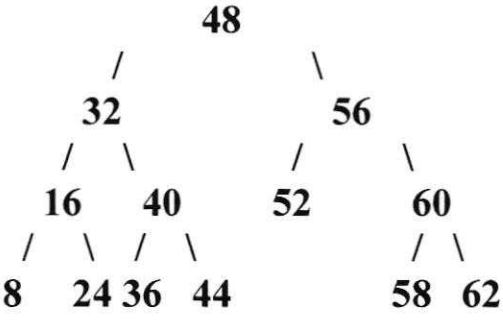
When we restructure at the node storing the 32, we identify the node storing the 56 as the tallest grandchild. Following the steps we've done previously, we get the final tree as follows:



4) *The final example, we will delete the node storing 4 from the AVL tree below:*



When we call rebalance on the node storing an 8, (the parent of the deleted node), we do NOT find an imbalance at an ancestral node until we get to the root node of the tree. Here we once again identify the node storing 32 as node A, the node storing 48 as node B and the node storing 56 as node C. Accordingly, we restructure as follows:



B+ TREES

B Trees. B Trees are multi-way trees. That is each node contains a set of keys and pointers. A B Tree with four keys and five pointers represents the minimum size of a B Tree node. A B Tree contains only data pages.

B Trees are dynamic. That is, the height of the tree grows and contracts as records are added and deleted.

B+ Trees A B+ Tree combines features of ISAM and B Trees. It contains index pages and data pages. The data pages always appear as leaf nodes in the tree. The root node and intermediate nodes are always index pages. These features are similar to ISAM. Unlike ISAM, overflow pages are not used in B+ trees.

The index pages in a B+ tree are constructed through the process of inserting and deleting records. Thus, B+ trees grow and contract like their B Tree counterparts. The contents and the number of index pages reflects this growth and shrinkage.

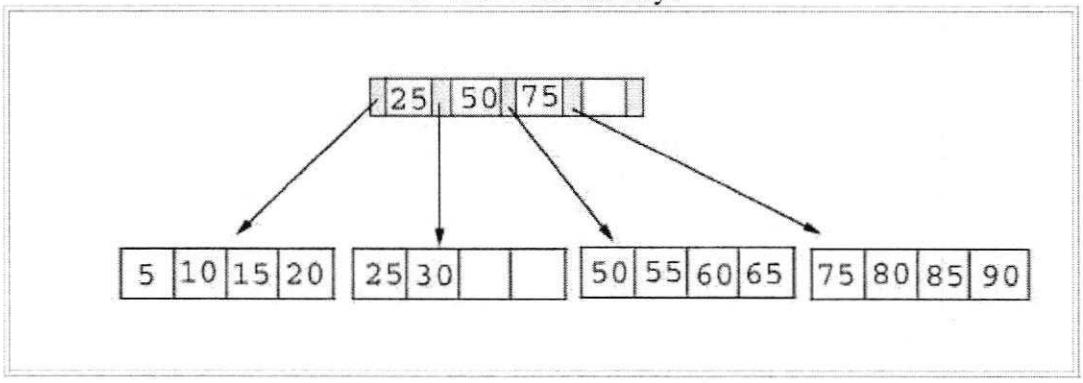
B+ Trees and B Trees use a "fill factor" to control the growth and the shrinkage. A 50% fill factor would be the minimum for any B+ or B tree. As our example we use the smallest page structure. This means that our B+ tree conforms to the following guidelines.

Number of Keys/page	4
Number of Pointers/page	5
Fill Factor	50%
Minimum Keys in each page	2

As this table indicates each page must have a minimum of two keys. The root page may violate this rule.

The following table shows a B+ tree. As the example illustrates this tree does not have a full index page. (We have room for one more key and pointer in the root page.) In addition, one of the data pages contains empty slots.

B+ Tree with four keys



Adding Records to a B+ Tree

The key value determines a record's placement in a B+ tree. The leaf pages are maintained in sequential order AND a doubly linked list (not shown) connects each leaf page with its sibling page(s). This doubly linked list speeds data movement as the pages grow and contract.

We must consider three scenarios when we add a record to a B+ tree. Each scenario causes a different action in the insert algorithm. The scenarios are:

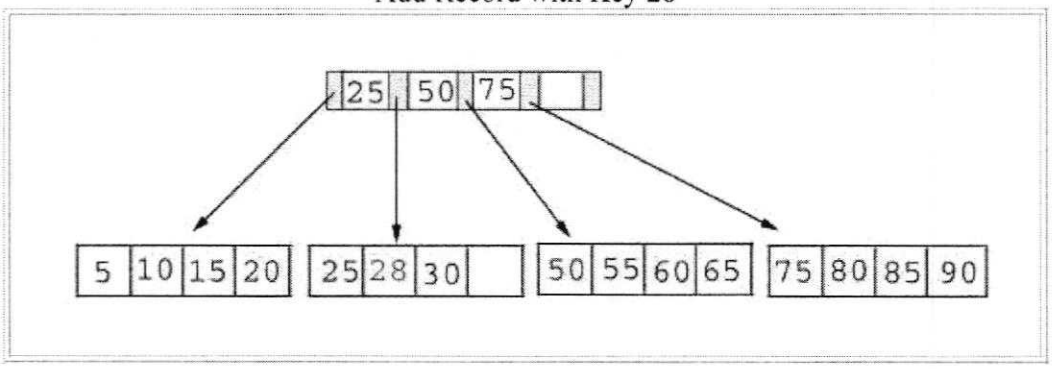
The `insert` algorithm for B+ Trees

Leaf Page Full	Index Page FULL	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. 4. Split the index page. 5. Keys < middle key go to the left index page. 6. Keys > middle key go to the right index page. 7. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

Illustrations of the insert algorithm

The following examples illlustrate each of the **insert** scenarios. We begin with the simplest scenario: inserting a record into a leaf page that is not full. Since only the leaf node containing 25 and 30 contains expansion room, we're going to insert a record with a key value of 28 into the B+ tree. The following figures shows the result of this addition.

Add Record with Key 28



Adding a record when the leaf page is full but the index page is not

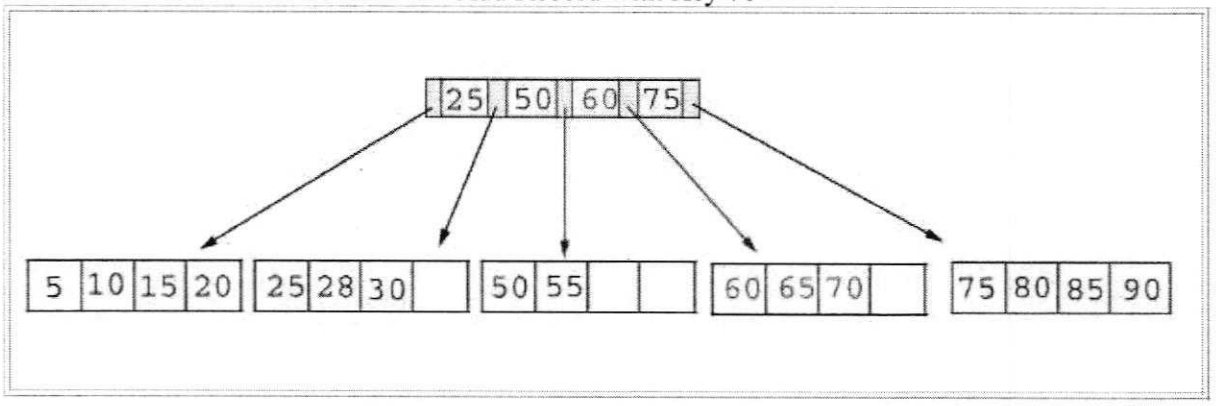
Next, we're going to insert a record with a key value of 70 into our B+ tree. This record should go in the leaf page containing 50, 55, 60, and 65. Unfortunately this page is full. This means that we must split the page as follows:

Left Leaf Page	Right Leaf Page
50 55	60 65 70

The middle key of 60 is placed in the index page between 50 and 75.

The following table shows the B+ tree after the addition of 70.

Add Record with Key 70



Adding a record when both the leaf page and the index page are full

As our last example, we're going to add a record containing a key value of 95 to our B+ tree. This record belongs in the page containing 75, 80, 85, and 90. Since this page is full we split it into two pages:

Left Leaf Page	Right Leaf Page

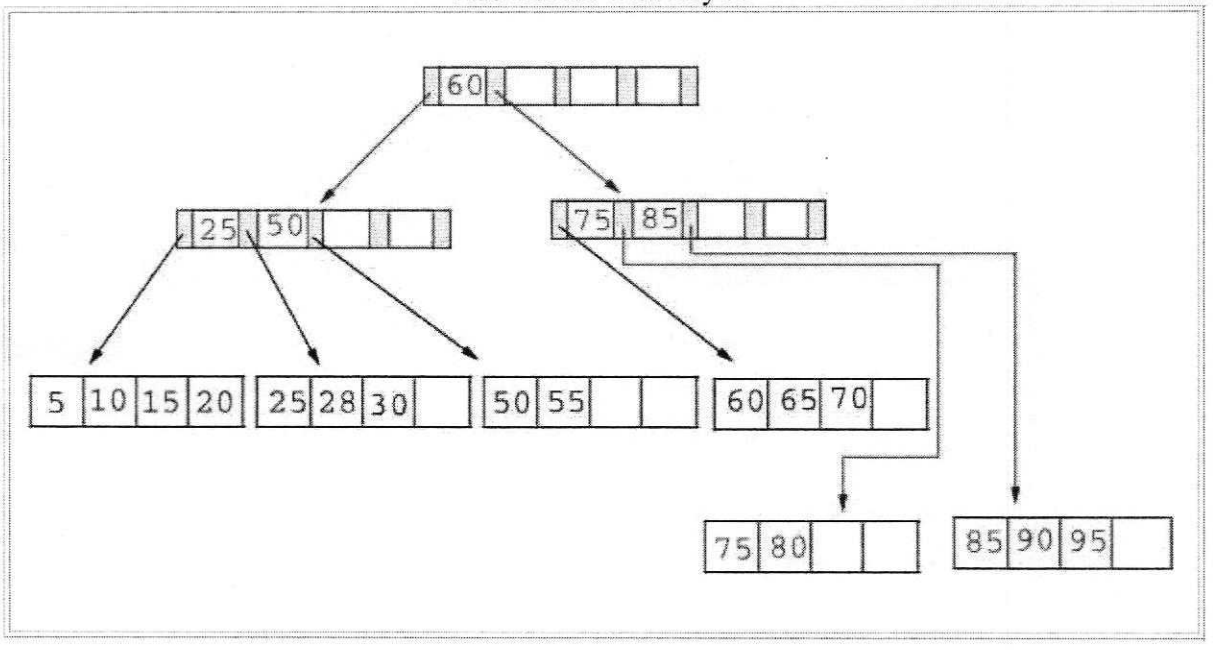
75 80	85 90 95

The middle key, 85, rises to the index page. Unfortunately, the index page is also full, so we split the index page:

Left Index Page	Right Index Page	New Index Page
25 50	75 85	60

The following table illustrates the addition of the record containing 95 to the B+ tree.

Add Record with Key 95

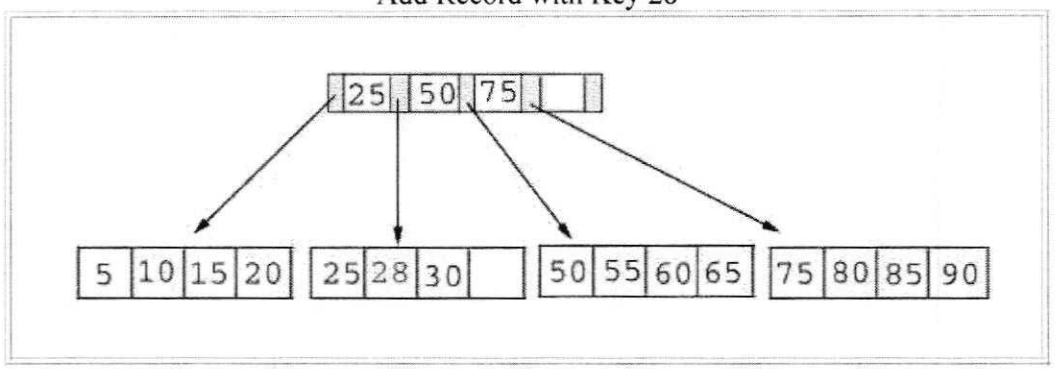


Rotation

B+ trees can incorporate rotation to reduce the number of page splits. A rotation occurs when a leaf page is full, but one of its sibling pages is not full. Rather than splitting the leaf page, we move a record to its sibling, adjusting the indices as necessary. Typically, the left sibling is checked first (if it exists) and then the right sibling.

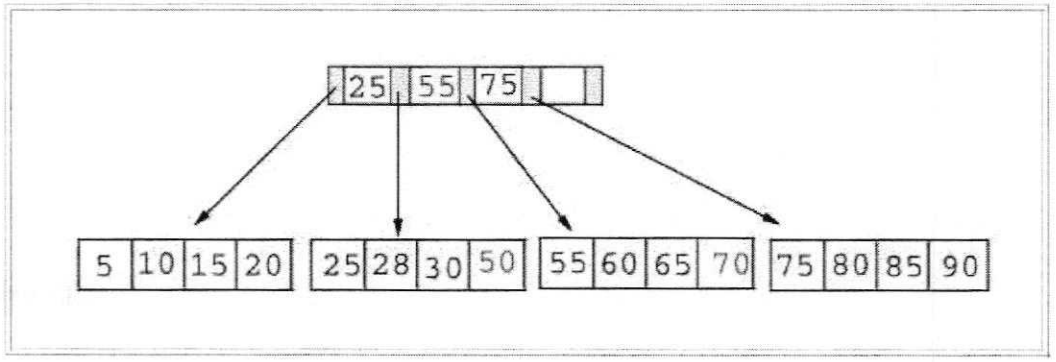
As an example, consider the B+ tree before the addition of the record containing a key of 70. As previously stated this record belongs in the leaf node containing 50 55 60 65. Notice that this node is full, but its left sibling is not.

Add Record with Key 28



Using rotation we shift the record with the lowest key to its sibling. Since this key appeared in the index page we also modify the index page. The new B+ tree appears in the following table.

Illustration of Rotation



Deleting Keys from a B+ tree

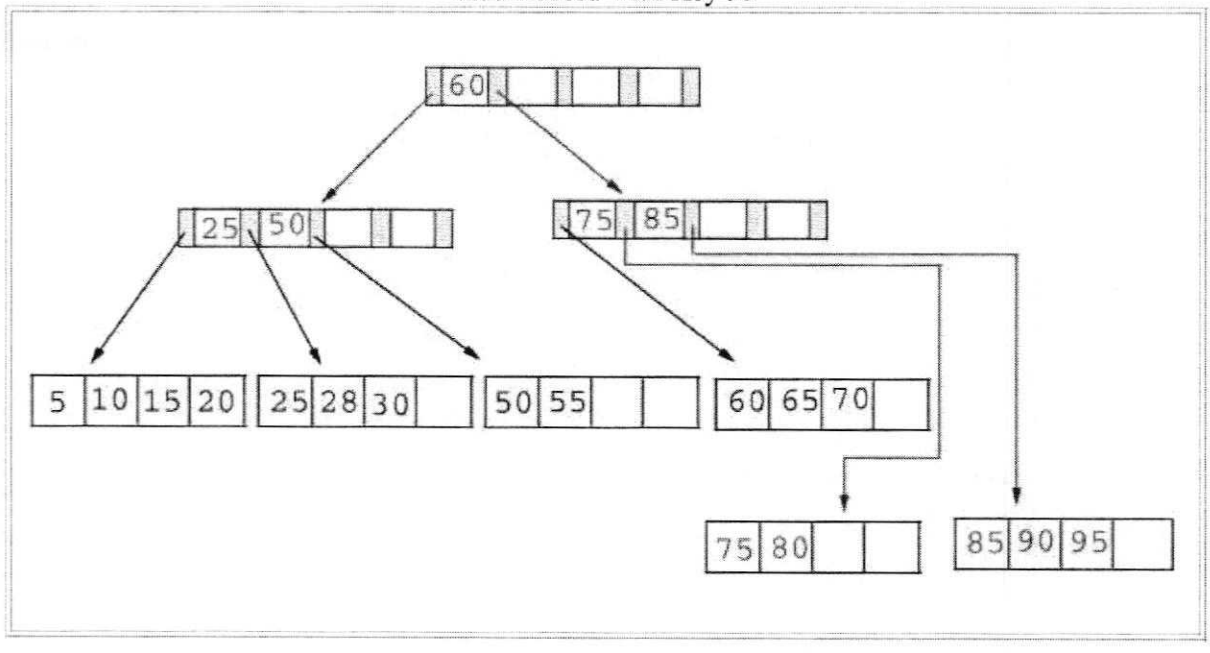
We must consider three scenarios when we delete a record from a B+ tree. Each scenario causes a different action in the delete algorithm. The scenarios are:

The `delete` algorithm for B+ Trees

Leaf Page Below Fill Factor	Index Page Below Fill Factor	Action
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

As our example, we consider the B+ tree after we added 95 as a key. As a refresher this tree is printed in the following table.

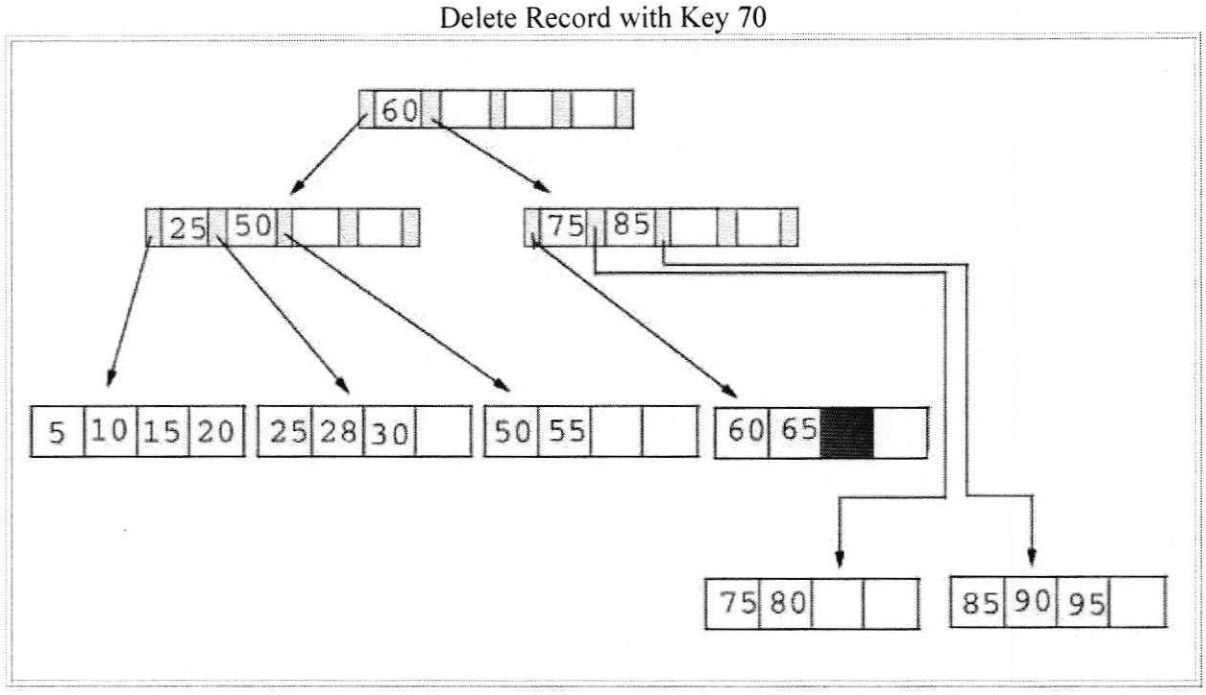
Add Record with Key 95



Delete 70 from the B+ Tree

We begin by deleting the record with key 70 from the B+ tree. This record is in a leaf page containing 60, 65 and 70. This page will contain 2 records after the deletion. Since our fill factor is 50% or (2 records) we simply delete 70 from the leaf node. The following table shows the B+ tree after the deletion.

Delete Record with Key 70

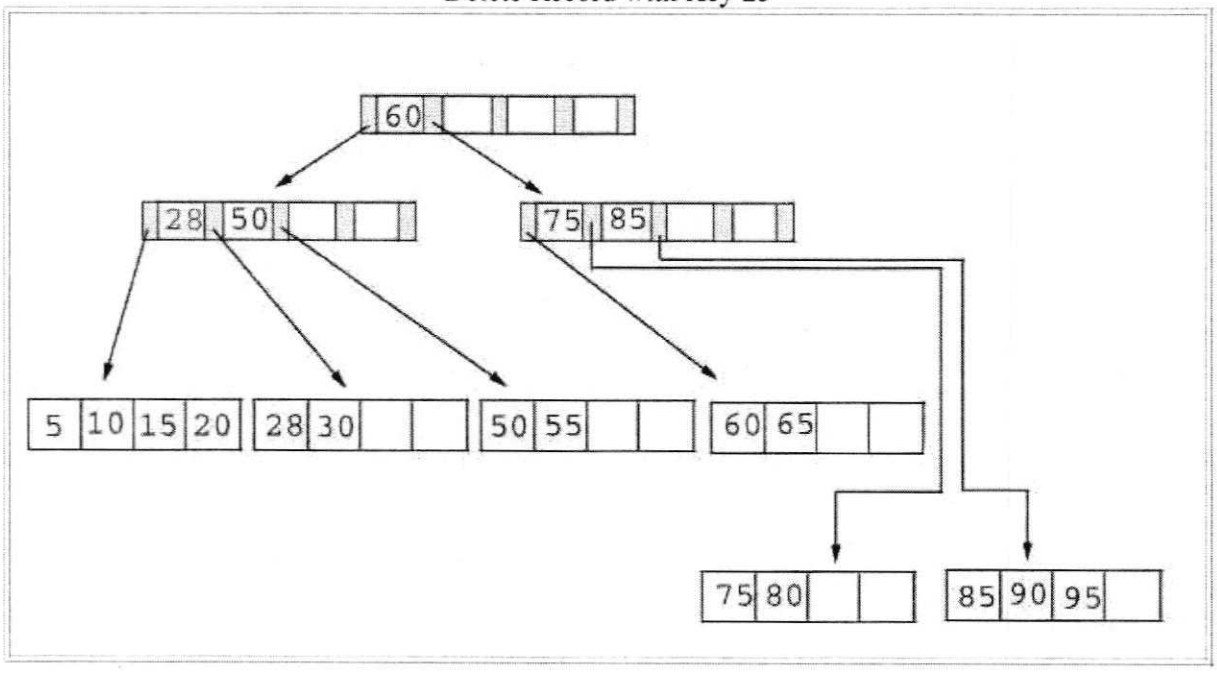


Delete 25 from the B+ tree

Next, we delete the record containing 25 from the B+ tree. This record is found in the leaf node containing 25, 28, and 30. The fill factor will be 50% after the deletion; however, 25 appears in the index page. Thus, when we delete 25 we must replace it with 28 in the index page.

The following table shows the B+ tree after this deletion.

Delete Record with Key 25



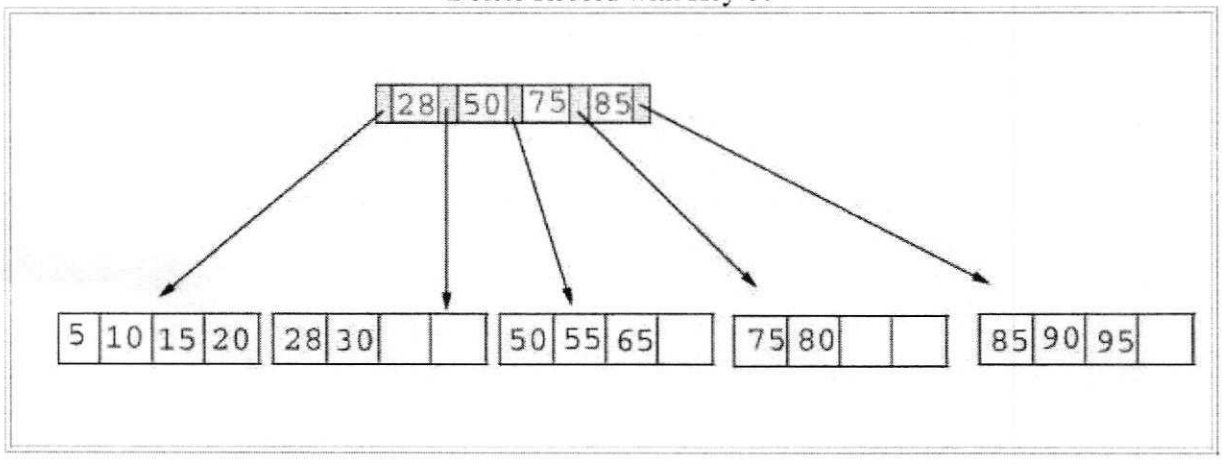
Delete 60 from the B+ tree

As our last example, we're going to delete 60 from the B+ tree. This deletion is interesting for several resasons:

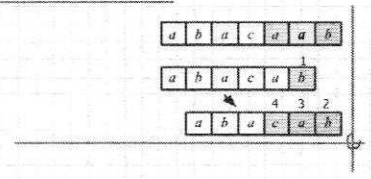
1. The leaf page containing 60 (60 65) will be below the fill factor after the deletion. Thus, we must combine leaf pages.
2. With recombined pages, the index page will be reduced by one key. Hence, it will also fall below the fill factor. Thus, we must combine index pages.
3. Sixty appears as the only key in the root index page. Obviously, it will be removed with the deletion.

The following table shows the B+ tree after the deletion of 60. Notice that the tree contains a single index page.

Delete Record with Key 60



Pattern Matching



Strings

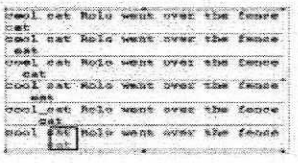
- A string is a sequence of characters
- Examples of strings:
  - C program
  - HTML document
  - DNA sequence
  - Digitized image
- An alphabet Σ is the set of possible characters for a family of strings
- Example of alphabets:
  - ASCII
  - Unicode
  - {0, 1}
  - {A, C, G, T}
- Let P be a string of size m
  - A substring $P[i..j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
  - A prefix of P is a substring of the type $P[0..i]$
  - A suffix of P is a substring of the type $P[i..m-1]$
- Given strings T (text) and P (pattern), the pattern matching problem consists of finding a substring of T equal to P
- Applications:
  - Text editors
  - Search engines
  - Biological research

Brute-Force Algorithm

- The brute-force pattern matching algorithm compares the pattern P with the text T for each possible shift of P relative to T, until either
  - a match is found, or
  - all placements of the pattern have been tried
- Brute-force pattern matching runs in time $O(nm)$
- Example of worst case:
  - $T = aaa \ldots ah$
  - $P = aaah$
  - may occur in images and DNA sequences
  - unlikely in English text

```
function BruteForceMatch(T, P, n, m)
    Input text T of size n and pattern
      P of size m
    Output starting index of a
      substring of T equal to P or −1
      if no such substring exists
    for (i = 0; i < n; i++){
      /* test shift i of the pattern */
      j = 0;
      while (j < m && T[i + j] == P[j])
        j = j + 1;
      if (j == m)
        return i; /* match at i */
    return -1;          /* no match */
```


Brute Force



Brute Force-Complexity

- Given a pattern M characters in length, and a text N characters in length...
- Worst case: compares pattern to each substring of text of length M. For example, M=5.
- This kind of case can occur for image data.

Total number of comparisons: M (N-M+1)
Worst case time complexity: O(MN)

Brute Force-Complexity(cont.)

- Given a pattern M characters in length, and a text N characters in length...
- Best case if pattern found: Finds pattern in first M positions of text. For example, M=5.

Total number of comparisons: M
Best case time complexity: O(M)

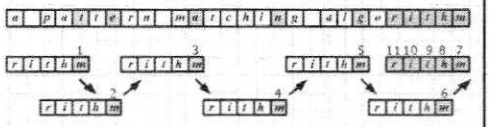
Brute Force-Complexity(cont.)

- Given a pattern M characters in length, and a text N characters in length...
- Best case if pattern not found: Always mismatch on first character. For example, M=5.

Total number of comparisons: N
Best case time complexity: O(N)

Boyer-Moore's Algorithm (1)

- The Boyer-Moore's pattern matching algorithm is based on two heuristics
  Looking-glass heuristic: Compare P with a subsequence of T moving backwards
  Character-jump heuristic: When a mismatch occurs at $T[i] = c$
  - If P contains c, shift P to align the last occurrence of c in P with $T[i]$
  - Else, shift P to align $P[0]$ with $T[i + 1]$
- Example



Last-Occurrence Function

- Boyer-Moore's algorithm preprocesses the pattern P and the alphabet Σ to build the last-occurrence function L mapping Σ to integers, where $L(c)$ is defined as
  - the largest index i such that $P[i] = c$ or
  - -1 if no such index exists
- Example:
  - $\Sigma = \{a, b, c, d\}$
  - $P = abacab$

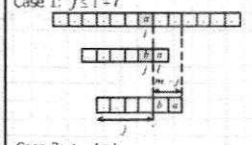
c	a	b	c	d
L(c)	4	5	3	-1

- The last-occurrence function can be represented by an array indexed by the numeric codes of the characters
- The last-occurrence function can be computed in time $O(m + s)$, where m is the size of P and s is the size of Σ

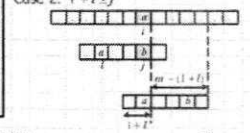
Boyer-Moore's Algorithm (2)

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L = lastOccurrenceFunction(P, Σ);
    i = m − 1;
    j = m − 1;
    repeat {
        if (T[i] == P[j])
            if (j == 0)
                return i; // match at i
            else {
                i −−;
                j −−;
            }
        else
            // character-jump
            l = L[T[i]];
            i = i + m − min(j, 1 + l);
            j = m − 1;
    }
    until (i > n − 1);
    return −1; // no match
```

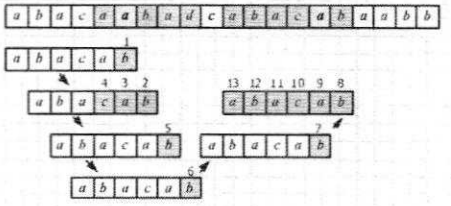
Case 1: $j \leq 1 + l$



Case 2: $1 + l \leq j$

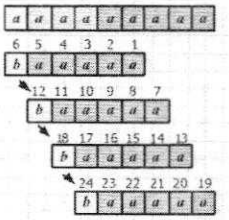


Example



Analysis

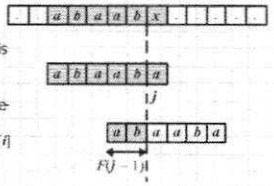
- Boyer-Moore's algorithm runs in time $O(nm + s)$
- Example of worst case:
  - $T = aaa \ldots a$
  - $P = baaa$
- The worst case may occur in images and DNA sequences but is unlikely in English text
- Boyer-Moore's algorithm is significantly faster than the brute-force algorithm on English text



KMP's Algorithm (1)

- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself
- The failure function $F(i)$ is defined as the size of the largest prefix of $P[0..j]$ that is also a suffix of $P[1..j]$
- Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm so that if a mismatch occurs at $P[j] \neq T[i]$ we set $j \leftarrow F(j-1)$

j	0	1	2	3	4	5
$P[j]$	a	b	a	a	b	a
$F(j)$	0	0	1	1	2	3



KMP's Algorithm (2)

- The failure function can be represented by an array and can be computed in $O(m)$ time

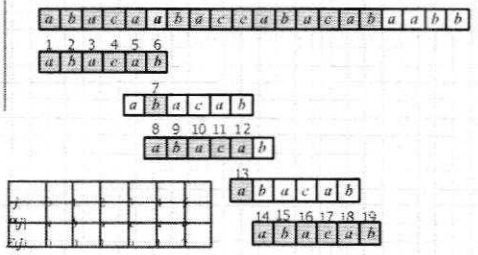
```
function FailureFunction( P)
  i = 1;
  j = 0;
  F[0] = 0;
  while ( i < m){
    if (P[i] == P[j]){
      F[i] = j + 1;
      i ++;
      j ++;
    }
    else if ( j > 0)
      j = F[j − 1];
    else{
      F[i] = 0;
      i ++;
    }
  }
  return F ;
```


KMP's Algorithm (3)

- At each iteration of the while-loop, either
  - i increases by one, or
  - the shift amount $i - j$ increases by at least one (observe that $F(j-1) < j$)
- Hence, there are no more than $2n$ iterations of the while-loop
- Thus, KMP's algorithm runs in optimal time $O(m + n)$

```
function KMPMatch(T, P)
  F = failureFunction(P);
  i = 0;
  j = 0;
  while ( i < n){
    if (T[i] == P[j])
      if ( j == m − 1)
        return i − j; /*match */
      else {
        i++;
        j++;
      }
    else
      if ( j > 0)
        j = F[j − 1];
      else
        i++;
  }
  return -1; /* no match */
```


Example



C Program To Implement Brute Force Algorithm

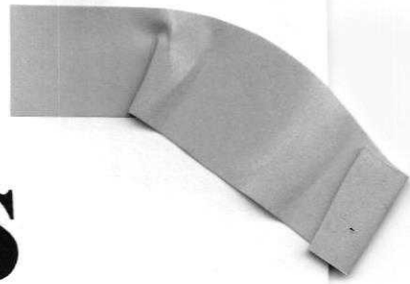
Brute-force search is a problem solving technique which is used to find the solution by systematically enumerating all possible candidates. For example, it can be used for pattern matching. Consider an input string "str" and a search string "p". We will try all possibilities to find whether there's any search string 'p' within the given input string "str".

See Also:
C Program To Implement Binary Search
C Program To Implement Linear Search On Sorted & Unsorted Array
C Program To Implement Brute Force Algorithm
C program To Implement Knuth-Morris-Pratt Algorithm

Example program to implement Brute Force Algorithm:

```
#include <stdio.h>
#include <string.h>
#define MAX 100

/* try to find the given pattern in the search string */
int bruteForce(char *search, char *pattern, int slen, int plen) {
    int i, j, k;

    for (i = 0; i <= slen - plen; i++) {
        for (j = 0, k = i; (search[k] == pattern[j]) &&
            (j < plen); j++, k++);
        if (j == plen)
            return j;
    }
    return -1;
}

int main() {
    char searchStr[MAX], pattern[MAX];
    int res;
    printf("Enter Search String:");
    fgets(searchStr, MAX, stdin);
    printf("Enter Pattern String:");
    fgets(pattern, MAX, stdin);
    searchStr[strlen(searchStr) - 1] = '\0';
    pattern[strlen(pattern) - 1] = '\0';
    res = bruteForce(searchStr, pattern, strlen(searchStr), strlen(pattern));
    if (res == -1) {
        printf("Search pattern is not available\n");
    } else {
        printf("Search pattern available at the location %d\n", res);
    }
    return 0;
```

Web References



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

Subject Code	Name of the Subject	Class/Sem	Name of the Faculty / Designation	Number of Students	Total Proposed Periods per semester/year	
					Lectures	Tutorial
123BP	Data Structures	II/I IT	Mr.Neil Gogte Assoc.Professor	60	76	10

Week Number	Lecture Number	Topic	Web References
W1	1	Introduction to Algorithm	http://nptel.iitm.ac.in/courses.php/ Introduction to Algorithm
	2	Characteristics of Algorithm	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	3	Recursive algorithms	https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion
	4	Performance analysis- time complexity and space complexity,	http://www.superwits.com/library/design-analysis-of-algorithm/course-content-daa/performanceanalysis
	5	Introduction to Asymptotic Notation Big O Notation,	http://btechsmartclass.com/DS/U1_T5.html
	6	TUTORIAL 1	http://www.geeksforgeeks.org/
W2	7	Omega and Theta notations,	http://btechsmartclass.com/DS/U1_T5.html
	8	Introduction to Linear and Non Linear data structures and Types of Linked List	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	9	Singly Linked Lists- Operation Insertion	http://www.codesters.org/
	10	TUTORIAL 2	http://www.geeksforgeeks.org/
W3	11	Singly Linked Lists-Operation Deletion	http://www.codesters.org/
	12	Traversing a linked List	http://www.geeksforgeeks.org/
	13	Inserting and Deleting from the middle of the Link List.	http://www.geeksforgeeks.org/
	14	Concatenating singly linked list.	http://www.geeksforgeeks.org/
	15	TUTORIAL 3	http://www.geeksforgeeks.org/

Signature of the Coordinator
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

W4	16	Introduction ,Operations for Circularly linked lists,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	17	Introduction to Doubly Linked List	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	18	Operations- Insertion & Deletion.	http://www.codesters.org/
	19	Sparse matrices-array & linked representation.	http://www.geeksforgeeks.org/
	20	TUTORIAL 4	http://www.geeksforgeeks.org/
W5	21	Introduction to Stack ADT. operations,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	22	Stack: array & linked implementations in C,	http://www.codesters.org/
	23	TUTORIAL 5	http://www.codesters.org/
	24	Applications-infix to postfix conversion.	http://www.geeksforgeeks.org/
	25	Postfix expression evaluation.	http://www.geeksforgeeks.org/
W6	26	Introduction to Queue ADT, operations,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	27	Queue: array & linked implementations in C,	http://www.codesters.org/
	28	TUTORIAL 6	http://www.codesters.org/
W7	29	Circular queues-Insertion and deletion operations.	http://www.geeksforgeeks.org/
	30	Introduction to Deque (Double ended queue) ADT,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	31	Deque :array implementations in C.	http://www.geeksforgeeks.org/
	32	Deque :Linked implementations in C.	http://www.geeksforgeeks.org/
	33	More examples on infix,prefix and postfix expressions	http://www.geeksforgeeks.org/
	34	TUTORIAL 7	

Signature of the Coordinator
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

W8	35	Introduction to Trees and there Terminology,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	36	Representation of Trees, Introduction to Binary Trees, Properties of Binary Trees,	http://cslibrary.stanford.edu/
	37	Representations-array and linked representations.	http://cslibrary.stanford.edu/
W9	38	Binary Tree traversals: preorder, inorder and postorder	http://cslibrary.stanford.edu/
	39	Threaded binary trees,	http://www.geeksforgeeks.org/
	40	Review Unit 3	https://www.hackerrank.com/domains/data-structures/trees/
	41	TUTORIAL 8	
W10	42	Introduction to Max Priority Queue ADT	https://www.coursera.org/learn/introduction-to-algorithms
	43	Implementation of Max Heap	https://www.coursera.org/learn/introduction-to-algorithms
	44	Insertion into a Max Heap,	https://www.coursera.org/learn/introduction-to-algorithms
W11	45	Deletion from a Max Heap.	https://www.coursera.org/learn/introduction-to-algorithms
	46	Introduction to Graphs, Terminology,	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
	47	Graph Representations- Adjacency matrix,	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
	48	Graph Representations- Adjacency List	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
	49	Graph traversals- BFS	https://visualgo.net/
	50	Introduction to Searching- Linear Search,	http://www.geeksforgeeks.org/
	51	Binary Search: Non Rec and Rec	http://www.geeksforgeeks.org/

Signature of the Coordinator **Signature of the Faculty**

Date Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

W12	52	Static Hashing-Introduction,	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	53	hash tables and hash functions,	https://visualgo.net/
W13	54	hash functions cont. Overflow Handling.	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	55	Introduction to Sorting- Selection Sort, Insertion Sort	https://visualgo.net/
	56	TUTORIAL 9	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	57	Quick sort,	https://visualgo.net/
	58	Heap Sort,	http://www.geeksforgeeks.org/
W14	59	Radix Sort, Comparison of Sorting methods.	http://www.geeksforgeeks.org/
	60	Review Unit 4	
W15	61	Introduction to Search Trees. Binary Search Tree.	http://www.geeksforgeeks.org/
	62	BST Operations- Searching, Insertion, Deletion.	https://visualgo.net/
	63	TUTORIAL 10	https://visualgo.net/
	64	Introduction to AVL Trees. Balance factor	http://www.cise.ufl.edu
	65	Insertion into an AVL Tree	http://www.cise.ufl.edu
	66	Introduction to B-Trees operations-Insertion and Searching	http://cslibrary.stanford.edu/
W16	67	Introduction to Red-Black.	http://cslibrary.stanford.edu/
	68	Introduction Splay Trees	http://www.cise.ufl.edu
	69	Comparison of Search Trees.	http://www.geeksforgeeks.org/
	70	Pattern matching algorithm The Knuth-Morris-Pratt algorithm	http://cslibrary.stanford.edu/

Signature of the Coordinator
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE – PLAN – One copy to be submitted to the HOD one week before commencement of the semester

	71	TUTORIAL 11	https://visualgo.net/
W17	72	Introduction to Tries	http://www.cise.ufl.edu
	73	Example program on BST insertion	http://www.codesters.org/
	74	examples on AVL insertion	http://www.cise.ufl.edu
	75	Example on B-Tree insertion	http://www.cise.ufl.edu
	76	Review Unit 5	

Signature of the Coordinator
Date

Signature of the Faculty
Date

*This column has to be filled-up after completion of the lecture/tutorial/practical in the copy kept with the faculty members.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
WEB REFERNCES

Subject Code	Name of the Subject	Class/Sem	Name of the Faculty / Designation	Number of Students	Total Proposed Periods per semester/year	
	Data Structures	IT- II/I	Mr. Neil Gogte	59	Lectures	Tutorial
					65	11

Week Number	Lecture Number	Topic	Web References
W1	1	Introduction to Algorithm	http://nptel.iitm.ac.in/courses.php/ Introduction to Algorithm
	2	Characteristics of Algorithm	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	3	Recursive algorithms	https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion
	4	Performance analysis- time complexity and space complexity,	http://www.superwits.com/library/design-analysis-of-algorithm/course-content-daa/performanceanalysis
	5	Introduction to Asymptotic Notation Big O Notation,	http://btechsmartclass.com/DS/U1_T5.html
	6	TUTORIAL 1 Recursion and Arrays	http://www.geeksforgeeks.org/
W2	7	Omega and Theta notations,	http://btechsmartclass.com/DS/U1_T5.html
	8	Introduction to Linear and Non Linear data structures and Types of Linked List	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	9	Singly Linked Lists- Operation Insertion	http://www.codesters.org/
	10	TUTORIAL 2 : Reverse a linked list, Find Length of a Linked List (Recursive)	http://www.geeksforgeeks.org/
W3	11	Singly Linked Lists-Operation Deletion	http://www.codesters.org/
	12	Traversing a linked List	http://www.geeksforgeeks.org/
	13	Inserting and Deleting from the middle of the Link List.	http://www.geeksforgeeks.org/
	14	Concatenating singly linked list.	http://www.geeksforgeeks.org/

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
WEB REFERNCES

	15	TUTORIAL 3 TUTORIAL 3: Merge two sorted linked lists	http://www.geeksforgeeks.org/
W4	16	Introduction ,Operations for Circularly linked lists,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	17	Introduction to Doubly Linked List	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	18	Operations- Insertion & Deletion.	http://www.codesters.org/
	19	Sparse matrices-array & linked representation.	http://www.geeksforgeeks.org/
	20	TUTORIAL 4: program to implement sparse matrix using linked list.	http://www.geeksforgeeks.org/
W5	21	Introduction to Stack ADT. operations,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	22	Stack: array & linked implementations in C,	http://www.codesters.org/
	24	Applications-infix to postfix conversion.	http://www.geeksforgeeks.org/
	25	Postfix expression evaluation.	http://www.geeksforgeeks.org/
	26	TUTORIAL 5: Towers of Hanoi problem	
W6	27	Introduction to Queue ADT, operations,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	28	Queue: array & linked implementations in C,	http://www.codesters.org/
	29	TUTORIAL 6: Implement a stack using two queues, Implement a queue using two stacks.	http://www.codesters.org/
	30	Circular queues-Insertion and deletion operations.	http://www.geeksforgeeks.org/
	31	Introduction to Deque (Double ended queue) ADT,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
WEB REFERNCES

W7	32	Deque :array implementations in C.	http://www.geeksforgeeks.org/
	33	Deque :Linked implementations in C.	http://www.geeksforgeeks.org/
W8	34	More examples on infix,prefix and postfix expressions	http://www.geeksforgeeks.org/
	35	Introduction to Trees and there Terminology,	https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
	36	Representation of Trees, Introduction to Binary Trees, Properties of Binary Trees,	http://cslibrary.stanford.edu/
	37	Representations-array and linked representations.	http://cslibrary.stanford.edu/
W9	38	Binary Tree traversals: preorder, inorder and postorder	http://cslibrary.stanford.edu/
	39	Threaded binary trees,	http://www.geeksforgeeks.org/
	40	Review Unit 3	https://www.hackerrank.com/domains/data-structures/trees/
	41	TUTORIAL 7: program to find the Total Nodes and Total Leaf Nodes of Binary Tree	
W10	42	Introduction to Max Priority Queue ADT	https://www.coursera.org/learn/introduction-to-algorithms
	43	Implementation of Max Heap	https://www.coursera.org/learn/introduction-to-algorithms
	44	Insertion into a Max Heap,	https://www.coursera.org/learn/introduction-to-algorithms
W11	45	Deletion from a Max Heap.	https://www.coursera.org/learn/introduction-to-algorithms
	46	Introduction to Graphs, Terminology,	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
	47	Graph Representations- Adjacency matrix,	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
WEB REFERNCES

	48	Graph Representations- Adjacency List	http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
	49	T8: Graph traversals- BFS	https://visualgo.net/
W12	50	Introduction to Searching- Linear Search,	http://www.geeksforgeeks.org/
	51	Binary Search: Non Rec and Rec	http://www.geeksforgeeks.org/
	52	Static Hashing-Introduction,	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	53	hash tables and hash functions,	https://visualgo.net/
W13	54	hash functions cont. Overflow Handling.	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	55	Introduction to Sorting- Selection Sort, Insertion Sort	https://visualgo.net/
	56	TUTORIAL 9: Merge Sort, Shell sort	http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=IntroToAlgorithms&video=CS161L12P1&speed=100
	57	Quick sort,	https://visualgo.net/
	58	Heap Sort,	http://www.geeksforgeeks.org/
W14	59	Radix Sort, Comparison of Sorting methods.	http://www.geeksforgeeks.org/
	60	Review Unit 4	
W15	61	Introduction to Search Trees. Binary Search Tree.	http://www.geeksforgeeks.org/
	62	BST Operations- Searching, Insertion, Deletion.	https://visualgo.net/
	63	TUTORIAL 10: program to find the height of a Binary search Tree	https://visualgo.net/
	64	Introduction to AVL Trees. Balance factor	http://www.cise.ufl.edu
	65	Insertion into an AVL Tree	http://www.cise.ufl.edu

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
WEB REFERNCES

	66	Introduction to B-Trees operations-Insertion and Searching	http://cslibrary.stanford.edu/
W16	67	Introduction to Red-Black.	http://cslibrary.stanford.edu/
	68	Introduction Splay Trees	http://www.cise.ufl.edu
	69	Comparison of Search Trees.	http://www.geeksforgeeks.org/
	70	Pattern matching algorithm The Knuth-Morris-Pratt algorithm	http://cslibrary.stanford.edu/
	71	TUTORIAL 11 : Brute Force algorithm	https://visualgo.net/
W17	72	Introduction to Tries	http://www.cise.ufl.edu
	73	Example program on BST insertion	http://www.codesters.org/
	74	examples on AVL insertion	http://www.cise.ufl.edu
	75	Example on B-Tree insertion	http://www.cise.ufl.edu
	76	Review Unit 5	

Lecture (11) Notes

Lecture Notes

16/6/16

Unit 1:

Data Abstraction:

- Referring essential features and hidding all other implementation details.

Data Type:

- It is a collection of objects (⇒ Properties/attributes/fields) and the set of operations performed on those objects.
- In C++ and Java structures, classes are used to represent the data type.

ADT (Abstract Data Type):

- The specification of objects and their operations are separated from object representations and implementation of the operations. Egs: stacks, queues, linked list, trees.

Egs: Ex1

```
Struct student
{
  int ht no;
  Char Sname[30];
  float Sub[5];
  float tot, avg;

Public:
  Void get Studint Details ( );
  Void find Total ( );
  Void find Avg ( );
  Void display studint Details ( );
};
```

Ex 2:

```
Class stack.
{
  int top;
  int S[10];
Public:
    Void Push(int x);
    ~~void~~
     int Pop();
     Void display();
};
```

~~Excamples~~

Performance Analysis

important factors are.

1. Time taken by Algorithm. (Refered to as Time Complexity)
2. Space occupied. (Space Complexity).

Time complexity

Algorithm Execution time depends on.

- Type of Machine (Type of processor)
- Type of Instruction set.
- Type of compiler.
- Type of OS (single ~~processor~~ tasking / multi processor tasking)

- Hence we do not ~~Exa~~ actually count the time taken by the algorithm instead we measure the time Complexity as count of no. of instructions

get executed before the Program/algorithm generate the output.

finding the time complexity:

Ex1: Sample code.

```
int i=1, sum=0;          → 1.
for (i=1; i<=n; i++)     → (n+1)
{
sum = sum + i;           → n.
}.
pf ("%d", sum);          → 1
                           ---
                           2n+3.
```

Ex 2

```
for (i=1; i<=m; i++)          → m+1
{
for (j=1; j<=n; j++)          → m(n+1)
{
pf ("%d", A[i][j]);           → m.n
                                ---
}                               m+1+mn+m+mn.
}.
```

2mn + 2m + 1.

complexity)

king
ussor)

e
the
tions

ex3:

```
for (i=1; i<=n; i++)                    ⟶ (n+1)
{                                       ⟶ n(n+1)
for (j=1; j<=n ; j++)                     = n²+n
  {                                     ⟶ n * n = n²
   C[i][j]=0 ;
for (k=1; k<=n; k++)                    ⟶ n²(n+1)
                                          = n³+n²
  {
  C[i][j]= C[i][j] + (A[i][k] * B[k][j]);  ⟶ n²(n)
                                             = n³
    }
  }
}
```

$2n^3 + 3n^2 + 2n + 1$

7/6/16

Asymptotic Notations.

These Notations are used to represent time complexity of the algorithm.

1) Big O (Big oh)
2) omega (Ω)
3) Theta (Θ)

Big O Notation:

- The function $f(n) = O(n)$ if and only if there exists +ve constants c, n_0 such that $f(n) \leq c \cdot g(n)$ $\forall$ $n \geq n_0$

Examples:

1) $f(n) = 3n+2$

Let $g(n) = n$.

$f(n) \leq \underset{c}{④} \cdot g(n) \quad \forall \quad n \geq \underset{n_0}{②}$

$f(n) = O(g(n)) = O(n)$

2) $f(n) = 100n + 6$.

$g(n) = n$.

$f(n) \leq c \cdot g(n) \quad \forall \quad n \geq n_0$.

$c = 101$ $\quad$ $n_0 = 6$ $\quad$ $f(n) = O(n)$

3) $f(n) = 6 \times 2^n + n^2$.

$g(n) = 2^n$ $\quad$ $c = 7$

$f(n) \leq 7 \cdot g(n)$.

$\leq 7 \cdot 2^n \cdot \forall \quad n \geq 4$.

$n_0 = 4$ $\quad$ $f(n) = O(2^n)$

~~4) $10n^2$~~

4) $f(n) = 10n^2 + 4n + 2$.

$g(n) = n^2$ $\quad$ $c = 11$

$f(n) \leq 11 \cdot g(n)$

$10n^2 + 4n + 2 \leq 11 \cdot n^2 \cdot \forall \quad n \geq 5$

$n_0 = 5$ $\quad$ $f(n) = O(n^2)$

Omega Notation:

The function $f(n) = \Omega(g(n))$ if and only if there exists +ve constants c, n_0 such that $f(n) \geq c \cdot g(n) \; \forall \; n \geq n_0$

Examples:

1) $f(n) = 2n^2 + 4n + 1$

$g(n) = n^2$

$f(n) \geq 2 \cdot g(n) \; \forall \; n \geq ?$

$2n^2 + 4n + 1 \geq 2n^2 \; \forall \; n \geq 0$. $f(n) = \Omega \cdot (n^2)$

Theta Notation:

The function $f(n) = \Theta(g(n))$ if and only if there exists +ve constants c_1, c_2, n_0 such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \; \forall \; n \geq n_0$

Example:

1) $f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

$c_1 = 10$ (can be either 10 or 9)

$c_2 = 11$

$10 \cdot n^2 \leq 10n^2 + 4n + 2 \leq 11 \cdot n^2 \; \forall \; n \geq 5$

$f(n) = \Theta \cdot (n^2)$

Space Complexity :-

- The amount of memory required by the program for its Execution.
- It is actual denoted as $S(P) = C + Sp$

where :- C is fixed Portion (fixed size).

Sp is variable size.

Examples for fixed Part :- Instructions, Variables, identifiers.

Sp → variable size → This is the memory need for variables that depend on individual Problem instance.

Ex :- Recursion stack.

21/6/16

Linked list

```
void creat()
{
struct node * newnode, * Prevnode;
int x; char choice;
do
 {
  Pf ("Enter the Number");
  Sf ("%d", &x);
newnode = (struct node *) malloc (size of (struct node));
newnode → data = x;
newnode → link = NULL;
                if (root == NULL)
                    root = newnode;
                else
                Prevnode → link = newnode;
```

```
Prevnode = new node;
pf(" do you wish to continue (y/n)");
getchar();//fflush();
choice = getchar();
}
while (choice == 'y');
}.

void display()
{
struct node *temp = root;
while (temp != NULL)
{
pf ("%d ", temp -> data);
temp = temp -> link.
}
}

void main()
{
create()
display();
length();
}
```

22/6/16

```
int
{
int
c
}
```

Line

```
voic
{
int
whi
{
```

```
If (flag =
pf ("
else
pf
}
```

22/6/16

```
int length ( )
{
int l=0; struct node *temp=root;
while (temp != NULL)
  {
    l++;
  temp = temp->link;
  }
return l;
}
```

Linear Search Using Linked list :

```
Voicl Search (int key)
{
int flag =0; Struct node *temp = root; int pos=0;
while (temp != NULL)
{
  poss ++;
   if (temp -> data == key)
      {
        flag =1; break;
      }
    temp = temp -> link;
   }.
if (flgg == 1)
  Pf (" found at %d ", Pos);
  else
    Pf (" Not found");
  },
```

```
Void search (int k, int *f, int *p)
{
Struct node *temp=root;
while (temp != NULL)
{
(*p)++;
if (temp->data == key)
{
*f = 1; break;
}
temp = temp->link;
}
}

Void append (int x)
{
Struct node *temp, *newnode; temp = root;
newnode = (Struct node *) malloc (size of (Struct node));
newnode -> data = x;
newnode -> link = NULL;
while (temp -> link != NULL)
{
temp = temp -> link;
}
temp -> link = newnode;
},
```

```
/// append (insert) at any position.

Void append (insert) (int pos, int x)
{
Struct node *temp, *newnode; temp=root; int i=2;
newnode = (Struct node*) malloc (sizeof( struct node));
newnode → data = x;
newnode → link = NULL;
  if (Pos = = 1)
    {
    newnode → link = temp;
    root = newnode;
    }
else
  {
while (i< pos && temp → link != NULL).
    { i ++;
      temp = temp → link;
    }
newnode → link = temp → link;
 temp → link = newnode;
}
}
```

23/6/16

// deletion of node based on position.

```
void delt (int pos) {
Struct node * temp = root , * temp2 , * temp3 ;
int i = 2 ;
if (Pos == 1) {
    temp2 = root ;
    root = root → link ;
    free (temp2) ;
    }
else {
  while (i < pos && temp → link != NULL) {
        i++ ; temp = temp → link ;
        temp3 = temp → link ;
  } if (temp → link != NULL) {
  temp → link = temp → link → link ;
  free (temp3) ; } else { pf ("invalid pos") ;
  }
  }
```

// deletion of node based on value given

```
void search (int k ; int * f ; int * p) {
  Struct node * temp = root ;
  while (temp != NULL) {
      * p++ ;
```

```
if (temp->data == k) {
        *f++;
        break;
    }
  temp = temp->link;
  }
  }
void main(){
  Search (key; flag, &pos);
  if (flag == 1) {
      delet (pos);
    }
else{
pf("can't delete");
  }.
```

```
// display using recursion: (in forward)
void display (struct node *temp) {.
  if (temp != NULL) {
  pf("%d", temp->data); } display (temp->link);
  }
  }
```

* interchange these statements will get display reverse.

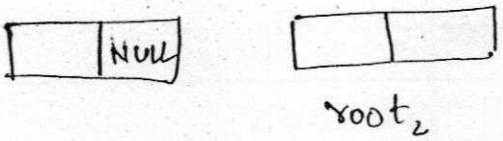
```
void main () {
  struct node *root = NULL;
        create (&root);
        display (root);
        }.
```

```
void Create (Struct node ** root) {
    .
    .
    .
if (* root == NULL) {
        * root = new node;
    }
}
```

// display using recursion (in backward):

```
void display reverse (St
```

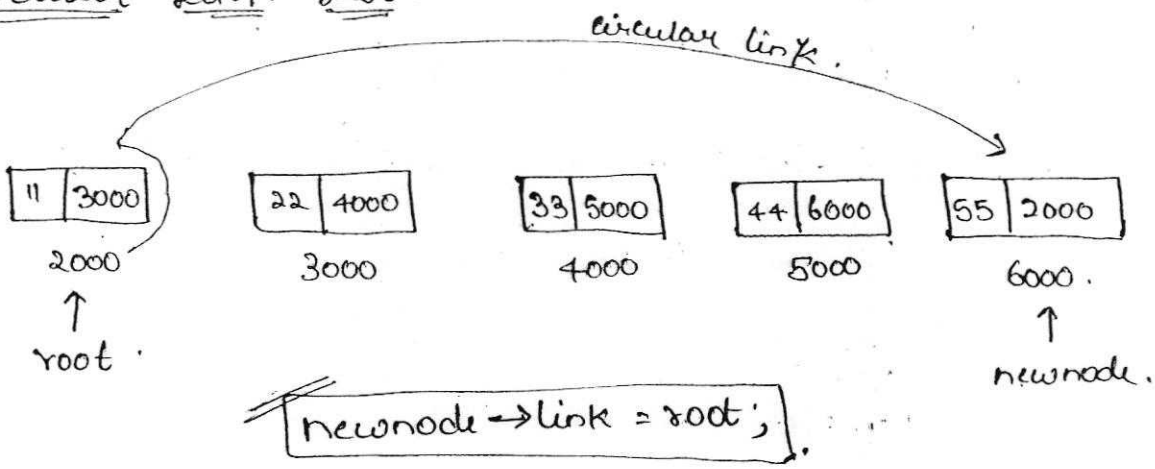
```
// Concatenate two linked lists;
void Concat (Struct node * root1, Struct node* root2)
{
  Struct node * temp = root1;
  while (temp -> link != NULL){
      temp = temp -> link;
  }
  temp -> link = root2;
}
void main ( ) {
  Struct node *root1 = NULL,
              *root2 = NULL;
  create (&root1);
  display (root1);
  create (&root2);
  display (root2);
  Concat (root1, root2);
  display (root1); }
```



$root_2$

28/6/16

Circular Link List :-



newnode → link = root;

~~void~~ creati.

```
#include <stdio.h>.
Struct node {.
 int data;
 Struct node * link;
 /* root = NULL;
```

```
void creati()
{
Struct node . * newnode , * Prevnode .
 int x , char choice ;
 do
 {
  Pf ("Enter the number");
  Sf (" %d ", & x);
  newnode = (Struct node*) malloc ( Size of (Struct node));
  new node → data = x ',
  newnode → link = NULL ;
```

```
if (root == NULL)
        root = newnode;
else
Prevnode -> link = newnode;
 Prevnode = newnode.
pf("DO you wish to continue (y/n)");
get char();
Choice = get char();
} while (Choice == 'y');
  newnode -> link = root;
}
```

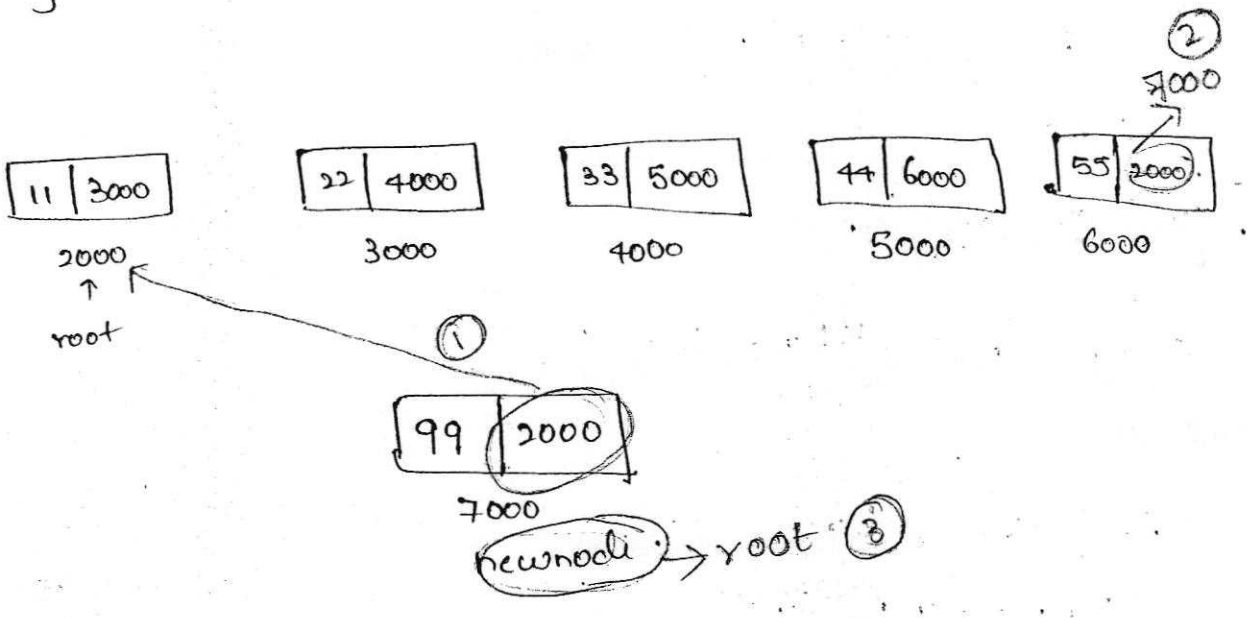
```
Void display()
{
Struct node *temp = root;
do
 {
 pf("%d", temp -> data);
   temp = temp -> link;
 }
while (temp != root);
```

```
Void insert()
temp = root , i = 2
newnode = (struct node*) malloc (Size of (struct node));
newnode -> data = x;
newnode -> link = NULL;
if (Pos == 1)
{
newnode -> Link = root; ①
```

```
while (temp->link != root).
    {
    temp = temp->link;
    }
temp->link = newnode;

root = newnode;
}
```

(2)

(3)




```
// at position (eg: 8 (or) 4...)
else{
while (i< pos && temp -> link != root)
  { i++;
    temp = temp -> link;
    }
new node -> link = temp -> link;
    temp -> link = newnode;
    }
}.
```

⇒ Void delete (int pos){.

```
Struct node *temp = root, *temp1;
int l=0; int i=2;
While (temp → link != root){.
  l ++;
  temp = temp → link;
}
if (pos == 1) {.
  {
    if (temp → link == root)
      { root = NULL;
      } .
    else
  while (temp → link != root).
      {
      temp = temp → link;
      }
temp → link = root → link;
  root = root → link;
  }
}
```

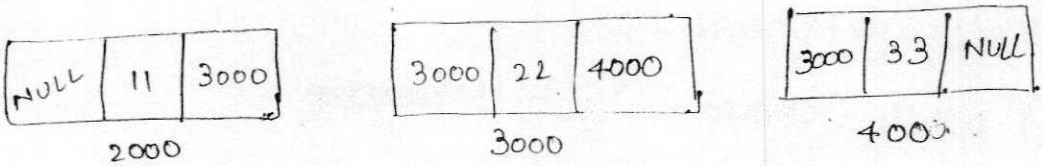
③ [at any position]

```
else if{
  while (i < pos && temp → link != root){
    i++;
    temp = temp → link;
    temp1 = temp → link;
  }
temp → link = temp → link → link;
free (temp1);
}
```

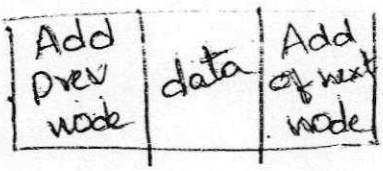
```
else {
    Pf ("Invalid pos");
    }
    }.
```

```
#in
St
{
}

S
}

→ Voic
{
Stic
int
{
Pf
sf(
newr
newr
newn
newn

el
```

Double Linked List.



root →




```
# include <stdio.h>
  Struct node.
  {
   Struct node * left ;
       int   data;
   Struct  node * right ;
   } * root = NULL ; * Last = NULL ;

⇒ Void Create ( )
   {
  Struct node. *newnode , * Prevnode , * last ;
  int x , Char Choice ;
  do {
   Pf ("Enter the Number");
   Sf ("%.d", &x) ;
 newnode = (Struct node *) malloc (Size of (Struct node)) ;
 newnode → data = x ;
 newnode → left = NULL ;
 newnode → right = NULL ;
        if (root == NULL)
              root = newnode ;
   else{
 Prevnode → right = newnode ;
 new
 Prevnode → left = Prevnode ; }.
```

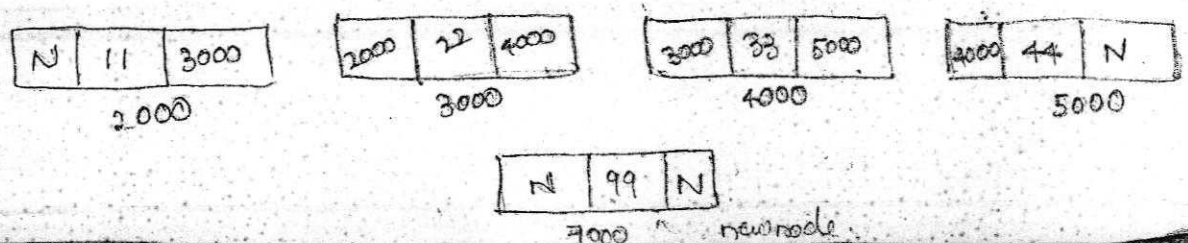
```
Prev = newnode;
Pf ("Do you wish to continue (y/n)");
getchar();
choice = getchar();
} while (choice == 'y');
last = newnode;
}
```

⇒ Void display ( ) for forward display.

```
{
struct node *temp = root;
while (temp != NULL)
{
Pf ("%d", temp → data);
temp = temp → right;
}
}
```

for backward display.

```
struct node *temp = last;
while (temp != NULL)
{
Pf("%d", temp → data);
temp = temp → lft;
}
}
```



30/6/16

⇒ Void insert (int pos, int x)

{.

Struct node *temp, *newnode, temp=root ; int i=2;

newnode = (struct node *) malloc (sizeof (struct node)) ;

newnode → left = NULL ;

newnode → data ;

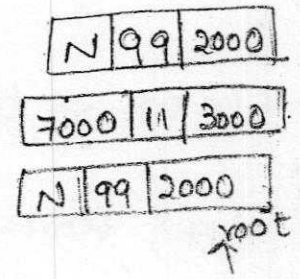
newnode → right = NULL ;

if (pos ==1) {.

newnode → right = root ;

root → left = newnode ;

root = newnode ;



}

else {

while (i< pos && temp → right != NULL) {.

i++ ;

temp = temp → right ;

}

newnode → right = temp → right ;

newnode → left = temp ;

if (temp → right != NULL)

temp → right → left = newnode ;

else.

last = newnode ; (for updating the last newnode)

temp → right = newnode ;.

}.

N
10

```
→ void delete (int pos)
  {
  struct node *temp; int i=2;
  if (pos==1)
    {
    root = root → right;
    root → left = NULL;
    }.
while (i<pos && temp → right != NULL).
    {
      i++; temp = temp → right;
    }
if (temp → right != NULL)
  {.
  temp → right = temp → right → right;
    if (temp → right != NULL).
      temp → right → left = temp;
       else
           last = temp;
    }
else
    {
    pf("invalid position");
    }
```

1 - 02/7/16

STACKS. UNIT - II

- Stack is a linear data structure which is based on LIFO (Last in First out).
- With stacks insertions and deletions happens through only one end called top of the stack.
- Inserting into the stack is called push operation.
- Deleting from the stack is called pop operation.
- If their is no space in the stack to insert a new element then it is called stack overflow.
- when the stack is empty, and we try doing any operation, then it is stack underflow.
- wan can implement the stack concept in 2 ways.
  (i) arrays
  (ii) linked list.

Applications of Stack:

- Converting an infix expression to its postfix notation.
- Evaluating the postfix expression.
- Nested function calls.
- Recursion.

Sparse Matrix Representation :-

Array Representation :-

Eg :-

	0	1	2	3
0	0	0	15	0
1	0	45	0	0
2	0	0	0	-92
3	0	37	46	0

(4 × 4)

Sparse matrix :-

row no.	column no.	non zero element
4 (no. of rows)	4 (no. of columns)	5 (no of non-zero element) → Standard
0	2	15
1	1	45
2	3	-92
3	1	37
3	2	46

(6 × 3)

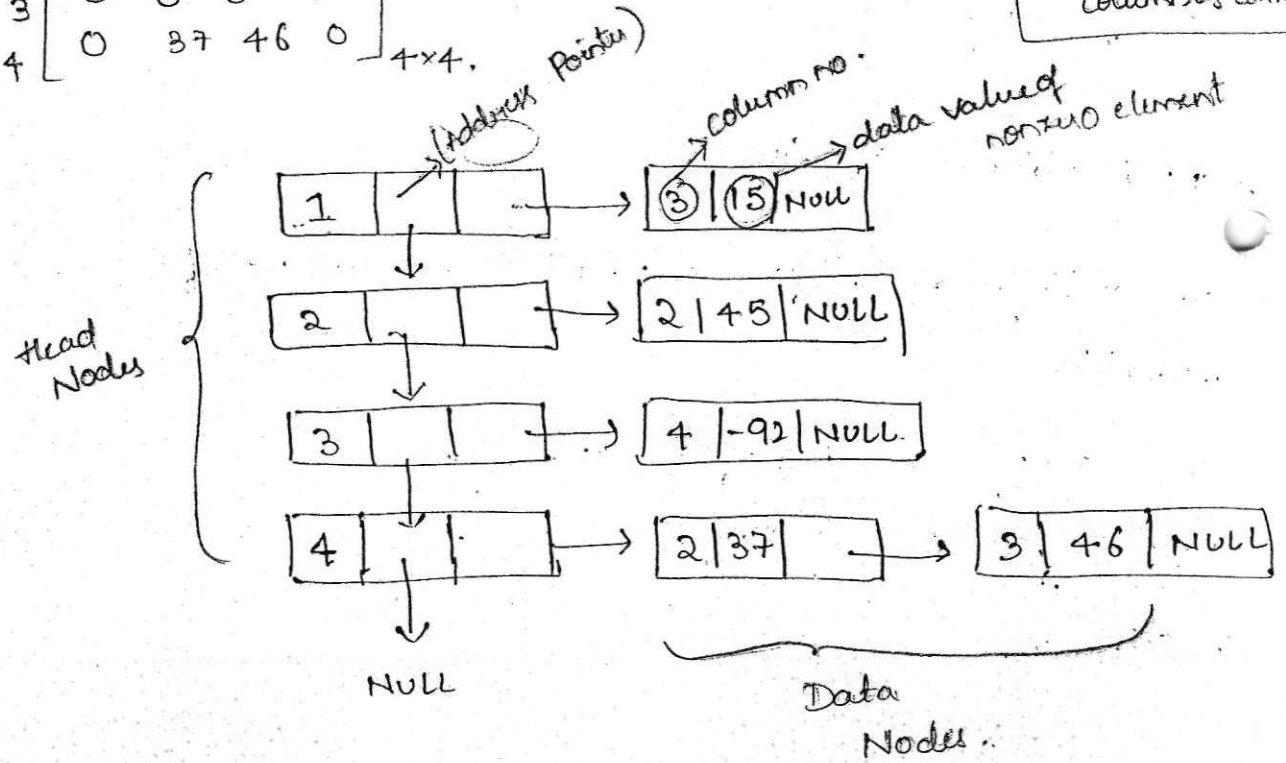
Linked Representation :-

[row is taken as common]

	1	2	3	4
1	0	0	15	0
2	0	45	0	0
3	0	0	0	-92
4	0	37	46	0

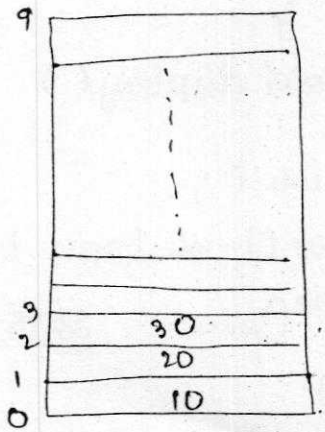
4×4.

⇒ Same linked representation can also be done by taking columns as common



UNIT – II

```
# define SIZE 10
Struct stack.
{
int S[SIZE];
int top;
} st;
int isstoverflow()
{ int x;
if (st.top >= SIZE-1)
    x=1;
else
    x=0;
return x;
}
int isstunderflow()
{
 int x;
 if (st.top == -1)
   x=1;
 else
   x=0;
  return x;
  }.
Void push (int x)
 {
 if (isstoverflow()==1)
 Pf ("Stack overflow = %d", x);
else {
    st.top++;
    St.S[st.top]=x;
    Pf ("inserted");
   }
   }.
```



```
Void Pop ()
{
 int x;
if (isstunderflow () == 1)
Pf("Stack underflow")
else
 {
 x = st. S[st. top];
 St. top--;
 Pf("Element to be delete = %d", x);
 }
 }
Void display ()
{
 int i;
for (i = st. top; i >= 0; i--)
Pf("%d", St.S[i])
}

Void main ()
{
St. top = -1;
 Push(10);
 Push (20);
 Push (30);
 display();
  Pop();
  display();
 }
```

```
Void main() {
  St. top = -1;
   int x;
   Char ch;
   do {
   Pf (" enter the elements");
   Sf (" %d ", & x);
   Push (x);
   Pf("do you wish to continue (y/n)");
  getchar();
  ch = getchar();
  } while (ch == 'y');
  display();
   POP ();
   display();
  }
```

4/7/16

Stack Using Linked List:-

```
# include <stdio.h>
# include <stdlib.h>

Struct node{
 int data;
 Struct node * link;
 }* top = NULL;

void push (){
 Struct node * newnode
 int x, char choice;
 do{
 Pf (" Enter the numbers ");
 Sf (" %d ", &x);
newnode = (Struct node *) malloc (sizeof (Struct node));
 newnode → data = x;
 newnode → link = NULL;
 if (top == NULL)
     {
        top = newnode;
    }
 else
     {
     newnode → link = top;
       top = newnode;
     }
   pf (" do you wish to continue (y/n)");
   get char();
   choice = get char();
   }
  While (choice == 'y');
  {
```

```
void display( )
 {
struct node *temp = top;
 while (temp != NULL)
  {
   pf("%d", temp->data);
   temp = temp->link;
  }
 }
```

```
Void pop ( ) {
 Struct node *temp=top;
 if (top==NULL)
 {
 pf("cannot delete")
 }
 else {
     top=top->link;
 }
}
```

```
Void main( ){
Push( );
display( );
pop( );
display( );
 }
```

Level of Priority
#
^
*, /, %
+, -
( )

5/7/11

1) Re

2) Re
in

3) Sto
inf

4) If

5)

exp

6) If
then re

w

we

- No

7)

Rem

Pre

The

8) Ret
u

9) D.
e

5/7/16

Converting Infix Expression to its Postfix form.

1) Read the infix expression.

2) Repeat steps 3, 4, 5, 6, 7, 8 for the each character in the infix expression.

3) Store current character in the ~~variable 'x'~~ infix expression in the variable 'x'.

4) If 'x' is left bracket (( ) put it in the stack.

5) If 'x' is an operand write it on the postfix expression.

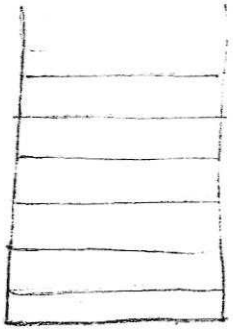
6) If 'x' is right bracket ( )) ~~for each operators~~ then remove all the operators from the stack and write them on the postfix expression until, we get the left bracket in the stack,
- Now remove left bracket the stack.

7) If 'x' is an operator
Remove all the operators from the stack whose Prescidence is greater than or equal to x:
Then, put them into postfix and then put the operator 'x' in the stack.

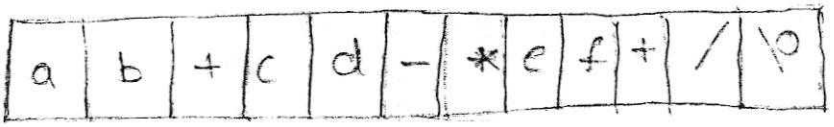
8) Retriev next character into 'x'.
Until you reach NULL character.

9) Display the characters stored in postfix expression.

Infix Expression :- ((a+b) *(c-d)/(e+f))



Stack



Postfix Expression

#
#

```
char St[20];
int top = -1;
void push (char x)
  {
    top ++;
    St [top] = x;
  }
Char pop ()
{
 Char x;
  x = St [top];
   top --;
   return x;
  }
```

```
int
 {
 if (

 else

 }
int p
  {
  if (

else if

else if

  else

  }
Void
{
Char
 char
 int

 Pf(
 gets (

 for
 { x

  if
  }
```

```
int isOperator (int Char x)
{
if (x == '+' || x == '-' || x == '*' || .....
    return 1;
else
    return 0;
}
int prec (Char x)
 {
if (x == '^')
    return 4;
else if (x == '*' || x == '/' || x == '%')
    return 3;
else if (x == '+' || x == '-')
    return 2;
  else
    return 1;
  }.
Void main ()
{
Char infix [80], postfix [80];
 char x, y;
 int i, j = 0;
 Pf ("enter infix expression");
 gets (infix);
 for (i=0 ; infix [i] != '\0' ; i++)
 { x = infix [i];
   if (x == '(')
   { push (x);
```

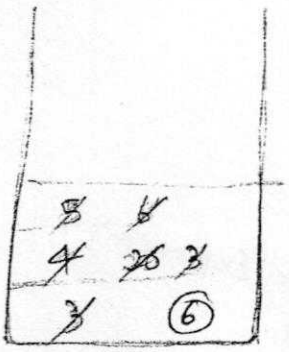
^	4
*, /, %	3
+, -	2
(,)	1

```
else if (x == ')')
  {
  while ( St[top] != '(' ).
     {
      Postfix[j] = pop();
       j++;
     }
  pop();
 }
else if (isoperator(x))
 {
 while (prec(St[top]) >= prec(x))
   {
     Postfix[j] = pop();
      j++;
   }
  Push(x);
 }
else if (x >= 'a' && x <= 'z')
  {
    postfix[j] = x;
     j++;
  }
 Postfix[j] = '\0';
  Puts (Postfix);
  }.
```

Postfix Evaluation

Postfix:

3	4	5	*	6	%	+	\0



b=5 } ①
a=4
b*a
=20

b=20 } ②
a=6
20/6 = 3....

b=3 } ③
a=3
3+3
=6

Ex 2:-

3	4	+	5	9	-	*	3	2	+	/

→ postfix.

a=97 A=65
b=98 B=
c=

'0' - 48
'1' - 49
'2' = 50
'3' - 51

```
#include <stdio.h>
int st[20];
int top = -1;

void push (int x)
{
  top ++;
 st [top] = x;
}
int pop (){
 int x;
 x = st [top];
  top --;
  return x;
}
```

```
int isoperator (char x)
{
if (x=='+' || x=='-' || x=='*' || ..... )
    return 1;
else
  return 0;
}.
void main( )
{
  char postfix[80]; int i; char x; int a, b;
    for(i=0; postfix[i] != '\0'; i++)
    { x= postfix[i];
      if (x>='0' && x<='9')
         push(x-48);
      else if (isoperator(x))
       {
          b=POP();
          a=POP();
          switch(x)
           {
             case '+' : push(a+b); break;
             case '*' : push(a*b'); break;
                  .
                  .
                  .
           }
       }
    }
  pf("%d", st[top]);
  }
```

= pf("Enter the Expression")
= gets(postfix)

11/7/16

Queue

- Queue is a linear data structure which follows first in first out (FIFO) mechanism., follows linear mechanism.
- Inserting data into the queue is called Enqueing/enqu operation
- Deleting data from the queue is called dequing/ dequ operation.
- Queue Overflow :- when their is no space in the queue to add a new element is known as queue overflow
- Queue Underflow :- when they are no elements in the queue to remove elements is known as queue underflo
- The mechanism queue is implimented in 2 ways.
  (i) Arrays.
  (ii) Linked-List.

Applications of Queue :-

(first come first serve)
- FCFS C.P.U scheduling algorithm.
- usage of a Printer in a shared networking envisonment

* Insertions into the queue we do through rear.
* Deletions through front.

Queue Using arrays

```
# define SIZE 10
int f = -1 , r = -1 ;
int q [SIZE];

int is qfull()
 {
   if (r >= SIZE - 1)
     return 1;
   else
     return 0 ;
 }
int is qempty( )
 {
   if ( f == -1)
     return 1;
   else
     return 0 ;
   }.

Void enqueue (int x)
{ if (isqfull( ) == 1)
     Pf ("Queue Overflow");
else {
     r++ ;
     q [r] = x ;
      if (f == -1)
        f = 0 ;
    }
 }.
```

```
Voic
{
int
if (
 Pf
else
for

 }

Void
{
if

else

 }
}
```

```
Void display()
{
int i;
if(isqempty() == 1)
 Pf(" No elements to display");
else {
 for(i=f; i<=r; i++)
    Pf("%d ", q[i]);

  }
}

void dqueue()
{
if (isqempty() == 1)
    Pf(" No elements to delete");
else {
    x = q[f];
    f++;
    Pf(" The element deleted = %d", x);

  }
}
```

12/7/16 Queue → Linked list

```
#include <stdio.h>
Struct node
{
 int data;
 Struct node *link;
} *f = NULL, *r = NULL;
Void enqueue()
{
Struct node *newnode;
int x;
do
{
Pf("Enter the number");
Sf("%d", &x);
newnode = (Struct node*) malloc (sizeof (Struct node));
newnode -> data = x;
newnode -> link = NULL;
if (f == NULL)
   {
    f = newnode;
    [r = newnode;] (not necessary)
   }
else
    {
      r -> link = newnode;
    }
  r = newnode;
}
```

```
{
pf ("Cant insert new element");
}
else if (f==-1)
{
f=r=0;
DQ[f]=x;
}
else {
f--;
DQ[f]=x;
}
}
```

(or)
elements can be inserted by shifting element.

```
R++;
for(i=R; i>0; i--);
DQ[i]=DQ[i-1]
DQ[0]=x
```

```
Void deleter()
{
if (f==-1)
pf ("no elements in Queue to delete");
else if (f==r)
{
f=-1
r=-1;
}
else
{
r--;
}
}.
```

u)

from

sition
ted
be

tion only
d. only
e from

```
        {
        f=r=0;
        }
    else {
        r++;
        }
    DQ[r]=x;
    }
  }
Void delet f( )
{ int x;
  if (f==-1)
   pf("Empty Can't delete");
else
   { x=DQ[f];
    if (f== r)
     {
       f=-1;
       r=-1;
     }
   else
     {
       f++;
     }
   pf("The element deleted = %d", x);
  }
}
Void insert f ( )
{
 if (f==0)
```

(Arrays) DeQueue (Double ended Queue)

- Dequeue stands for double ended queue.
- In Dequeue insertion and deletions can be done from both the ends i.e., front end and rear end.

They are 2 types of Dequeue.

① I/P restricted Dequeue :- where their is restriction only in insertion, i.e., elements must be inserted only from rear end and deletions can be done from both the ends.

② O/P restricted Dequeue :- Where their is restriction only in deletion i.e., elements must be deleted only from front end and insertions can be done from both the ends.

```
#include <stdio.h>
# define SIZE 50.
  int DQ[SIZE];
  int f=-1, r=-1;

void insertr(int x)
{
  if (r==SIZE-1)
      Pf ("Queue overflow");
  else {
      if (f==-1)
```

known
ell .
always

wed by
ve rear

```
void dequeue()
{
  int temp;
  if (f==-1)
    pf("Queue empty");
  else
  {temp=CQ[f];
    if (f==r)
    {f=r=-1;
    }
  else {f=(f+1)%SIZE;
      }
  pf("Element deleted =%d", temp);
  }

void display()
{
 int i;
 if (f==-1)
  pf("Queue empty");
 else
  {
 for(i=f; i!=r; i=(i+1)%5)
    {
    pf("%d", CQ[i]);
    }
  pf("%d", CQ[r]);
  }
 }
```

Deleting ~~into~~ from a Heap

23/7/16

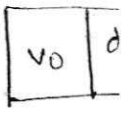
```
i=n-1;
temp= A[i]
A[i]= A[0];
k=0;
if (i==1)
    j=-1;
else
    j=1;
if (i>2 && A[2]>A[1])
    j=2;
while (j>0 && A[j]>temp)
{
    A[k]=A[j];
    k=j;
    j=(2*k)+1
if (j+1<=i-1 &&
        A[j+1] > A[j])
    j++;
if (j>i-1)
    j=-1;
}
A[k]=temp;
n--;
```

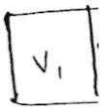
In Deletion elements are arranged in Sorted Ascending order for Max heap and in Sorted Decending order for min heap

Adjo

Adj

1st a





2nd ap

21/7/16

Heap UNIT – III

- Heap is a Complete binary tree or an almost Complete binary tree, having all the Parent node values are either greater (or) (lesser) lesser than its children.

Almost Complete binary tree:

- It is a binary tree which satisfies the following two Properties.

1) A node must have a left child, in case if it has a right child.

2) If the height of the tree is h, then all the leaf nodes must be at the level h (or) h-1

Types of Heaps.

1) Max Heap :- Parent node values are greater than its Childs

2) Min Heap :- Parent node values are lesser than its Children

Inserting into a Heap (Max Heap)

```
insert (int n, int x, int A [ ])
{
// n no. of existing elements in the Array.
// x. elements to be added;
    int i = n;
    while (i>0 && A[(i-1)/2] < x)
    {
        A[i] = A[(i-1)/2];
        i = (i-1)/2;
    }
    A[i] = x;
}
```

```
else {
  for (i=f; i<=r; i++){
    Pf ("%d ", q[i]);
    }
  }
}
void main() {
 int a;
 Pf ("Enter element");
 Sf ("%d", &a);
 enqueue (a);
 display();
 dequeue();
 display();
 }.
```

21/7/16

- Heap
binar
either

Almos

- It is
two
1) A
a
2) If
node

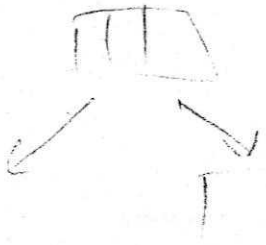
Types of

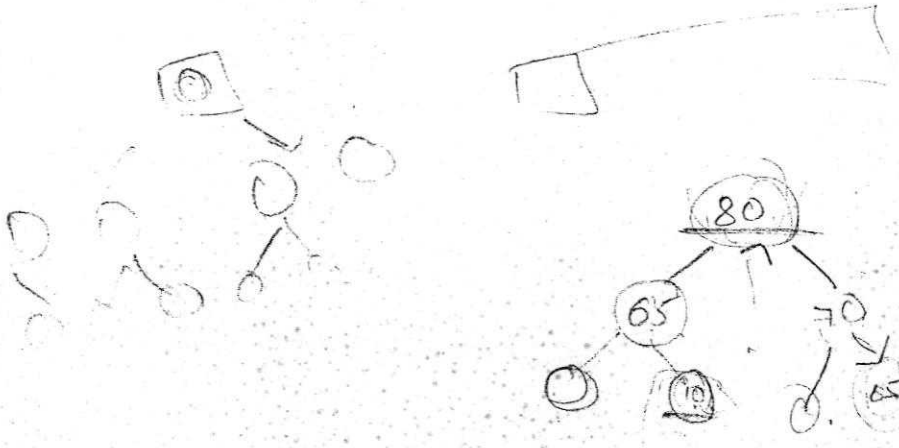
1) Max

2) Min

Insertio

insert(
{
// n
// x.
i
whil





Ascending Order Priority Queue:

```
Void enqueue (int x)
{
 int j;
 if (f == -1)
  { f++; }
j = r;
while (j >= 0 && Q[j] > x)
   {
    Q[j+1] = Q[j];
     j--;
   }
Q[j+1] = x;
r++;
}.
```

Here:

f = -1

r = 0

```
void dequeue () {
if (f == -1) {
     Pf ("can't delete);
   }
else
   f++;

void display () {
  int i;
if (f == -1) {
    Pf ("no elements to display");
   }
```

20/7/16

Priority Queues:

It is a set of ordered elements in which each element is associated with same priority.

we have two types of Priority Queues.

1. Ascending order Priority Queue.
2. Decending order Priority Queue

1. Ascending order Priority Queue.

In this insertions can happen in any order. where as we can always remove only the smallest element in the Queue.

2. Decending order Priority Queue:-

In this insertions can happen in any order where as we can always remove only the biggest element in the queue.

Applications:-

- It is used in the CPU scheduling in which all the Processes are assigned priorities and the processor will be allocated to the processes based on the priority order.

Asc

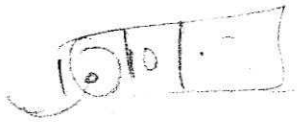
```
Void
{
  int
  if (
  {
  j=r
  while
  {
  Q
  r
  q[j
  r++
}.
void
if (f
}
else
f.
void
int
if (
```

```
        temp = temp -> right;
        if (temp == head)
        return;
        Pf("%d", temp -> data);
        }
    temp = temp -> right;
    }
}
void main()
{
create();
Pf("\n inorder non recursive : ");
inordernonrec(root);
}
```

Temp

2000
3000
5000
3000
6000
2000
4000
7000
4000
8000
1000

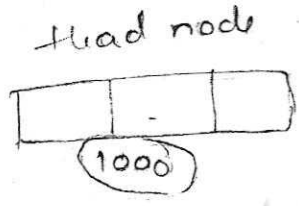
10, 20, 30, 40, 50, 60, 70



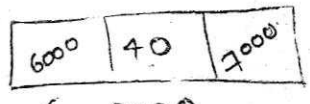
19/7/16

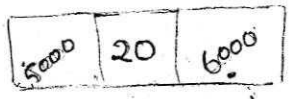
LNR

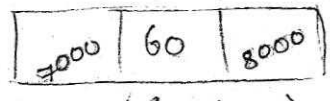
Inorder Threaded Binary Trees

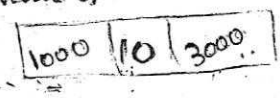


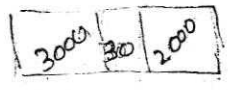
→ This is also Example for binary search tree i.e., left child should have less value than the parent node and right child should have ≥ value of Parent node

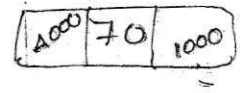








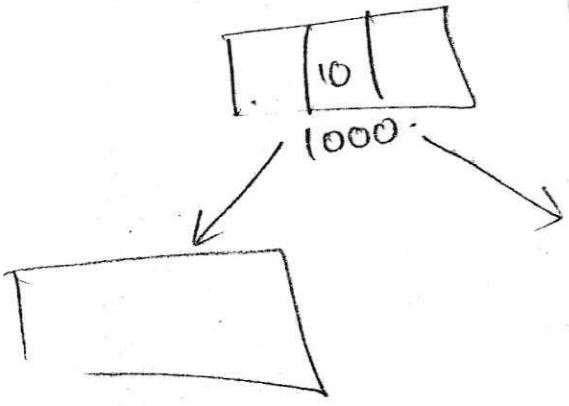




10, 20, 30, 40, 50, 60, 70

```
if (root == NULL)
  {
  root = nn;
  head = (struct tree *)malloc (sizeof (struct tree));
  head -> left = root;
  head -> data = 999;
  root -> left = head;
  root -> right = head;
  }
else
  insert (root, nn);
pf ("\n do you wish to continue (y/n)");
getchar();
ch = getchar();
}
while (ch == 'y');
}

void inordernonrec (struct tree * temp)
{
while (temp != head)
{
while (temp -> haslchild == 1)
      temp = temp -> left;
  pf("%d", temp -> data);
while (temp -> hasrchild == 0)
{
```

```
else
  insert (temp->left, nn);
 }
else
 {
  if(temp -> hasrchild ==0)
    {
     nn->right = temp->right;
     nn -> left =temp;
   temp -> hasrchild =1;
   temp -> right=nn;
   }
else
  insert (temp->right,nn);
 }
}

Void creat()
{
Struct tree *nn;
char ch; int x;
do
 {
 Pf("enter data");
 Sf ("%d", &x);
 nn = (Struct tree*) malloc (sizeof (struct tree));
 nn -> left = NULL;
 nn -> haslchild =0;
 nn -> data = x;
 nn -> hasrchild =0;
 nn -> right = NULL;
```



data);

Inorder Threaded Binary Trees

```
# include <Stdio.h>
# include <Stdlib.h>

Struct tree
{
Struct tree * left;
int haslchild;
int data;
int hasrchild;
Struct tree * right;
}* root = NULL, *head = NULL;

void insert (Struct tree *temp, Struct tree *nn)
{
char ch;
Pf("\n inser to the left or right of %d", temp->data);
getchar ();
Sf("%c", &ch);
if (ch == '1')
 {
   if (temp -> haslchild == 0)
      {
      { nn -> left = temp -> left;
        nn -> right = temp;
      { temp -> left = nn;
        temp -> haslchild = 1;
      }
```

```
// after visiting LR subtrees
while (St[top].Check==0)
{
temp = St[top].addr;
pf("%d", temp->data);
top--;
if (top = -1)
  return;
}
// after visiting left subtree
temp = St[top].addr;
temp = temp->right;
St[top].Check=0;
} // closing of infinite while loop
}
```

18/7/16

Post order Iterative Method.

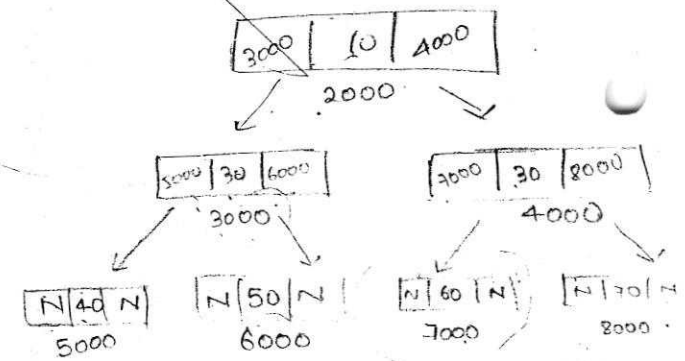
```
Struct tree
{
Struct tree *left;
int data;
Struct tree * right;
};
Struct element
{
Struct tree *addr;
    int check;
};
Struct element St[20];

Void Ipost order ()
{
Struct tree *temp = root;
while(1)
{     // for each new node
while (temp != NULL)
  {
   top++;
  St [top]. addr = temp;
  St [top]. check = 1;
   temp = temp → left;
  }
```

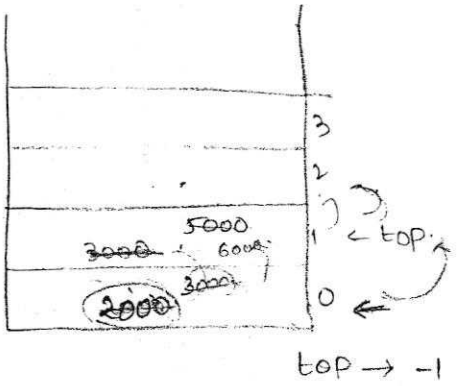
```
//
whi
{
ten
pf.
top
if (
  r
}
// aft
tem
ten
St[
},
}
```

```
pf("%d", temp->data);
    top++; st[top] = temp;
    temp = temp->left;
  }
if (top == -1)
   return;
else {
  temp = st[top]; top--;
   temp = temp->right;
  }
 }
}.
```

Inorder : 40, 20, 50, 10, 60, 30, 70

Preorder : 10, 30, 40, 50, 30, 60, 70.





Preorder : NLR

temp → ~~2000~~

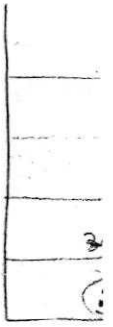
16/7/16

Without Recursion. (Iterative Travelsal Approach)

```
Void inorder( )
{
Struct tree *temp = root;
Struct tree *St[20]; int top=-1;
while (1)
{
while (temp != NULL)
{ top++; St[top] = temp;
  temp = temp → left;
}
if (top == -1)
    return;
else {
  temp = St[top]; top--;
  Pf("%d", temp→data);
  temp = temp → right;
  }
 }
}
```

→

```
Void Preorder( )
{
Struct tree *temp = root;
Struct tree *St[20]; int top=-1;
while(1)
{
while (temp != NULL)
{
```



```
newnode = (struct tree *) malloc (sizeof(struct tree));
newnode -> left = NULL;
newnode -> data = x;
newnode -> right = NULL;
if (root == NULL)
    root = newnode;
else
   insert (root, newnode);
pf ("do you wish to continue (y/n) ");
getchar();
ch=getchar();
}
while (ch == 'y');
}
void preorder (struct tree * temp)
{
if (temp 1 = NULL)
 {
  pf ("%d", temp->data);
  Preorder (temp -> left);
  preorder (temp -> right);
 }
}
```

```
void main() {
create ();
Preorder (root);
inorder (root);
Postorder (root);
}
```

```
Void insert (struct tree *temp, struct tree *newnode)
{
 Char ch;
 Pf ("insert to the left or right of %d", temp->data);
 getchar( );
 ch = getchar( );
 if (ch == 'l')
{
 if (temp->left != NULL)
        temp->left = newnode;
    else
        insert (temp->left, newnode);
 }
else {
    if (temp->right != NULL) temp->right = newnode;
    else insert (temp->right, newnode);
  }
}

Void Create ( )
{
 struct node * newnode;
 char ch; int x;
 do
 {
 Pf ("Enter data");
 Sf ("%d", &x);
```

```
newn
newr
newr
newr
if

else
 i

Pf
get
ch =
}
whil
}

void
{
if
{
T
P
}
}
```

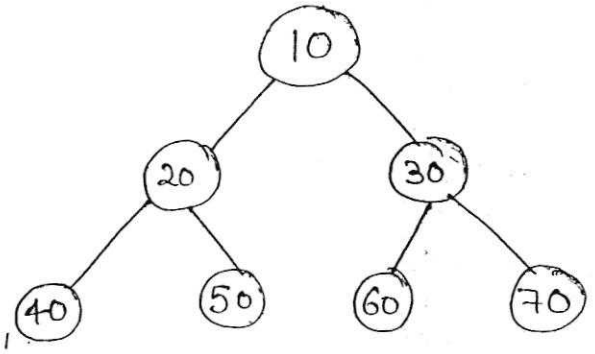
TREE TRAVERSING

(ii) TREE TRAVERSAL METHOD / TECHNIQUE.

1) Pre order : Node, Left Tree, Right tree.

2) In order : L (N) R.

3) Post order : L R (N)



Preorder : 10, 20, 40, 50, 30, 60, 70

Inorder : 40, 20, 50, 10, 60, 30, 70

Postorder : 40, 50, 20, 60, 70, 30, 10.

Binary Tree :- (Recursive Approach)

```
# include <stdio.h>
# include <stdlib.h>

struct tree → Node
{
 struct tree *left;
 int data;
 struct tree *right;
 } *root = NULL;
```

Tre

(ii)

Preor

Inor

Posto

Binc

#inc

#incl

struct

{

struc

int

Stru

};

ary Tree.

Height = (3) $\Rightarrow h$.

No. of nodes in a complete binary tree $= 2^{h+1} - 1$

$= 2^4 - 1$

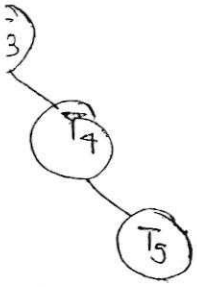
$= 16 - 1$

$= 15$

no. of leaf nodes $= 2^h$

$= 2^3$

$= 8$



1 has the children

odes are

14/7/16

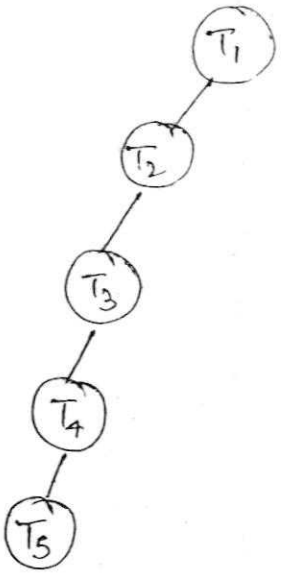
Binary Tree ADT.

```
Struct
{
instances :
    Struct tree *root ;
operations :
    create() ;
    insert() ;
    delete() ;
    Search() ;
    display() ;
}
```

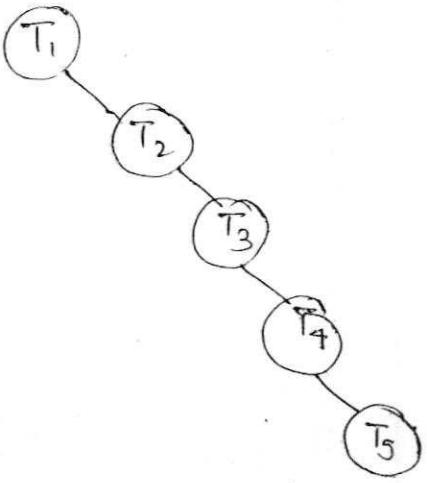
Applications of Binary Trees :-

- Binary Trees are used to construct expression trees used in evaluation of expressions by compiler.
- Tree structure is used by operating systems to organise directories and files (application of Trees not only for binary trees).
- Binary Trees are used in developing binary search trees.

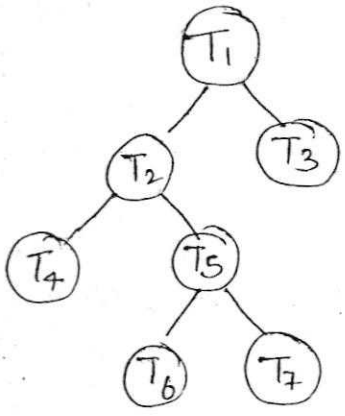
⇒ Left Skewed Binary Tree :



⇒ Right Skewed Binary Tree.

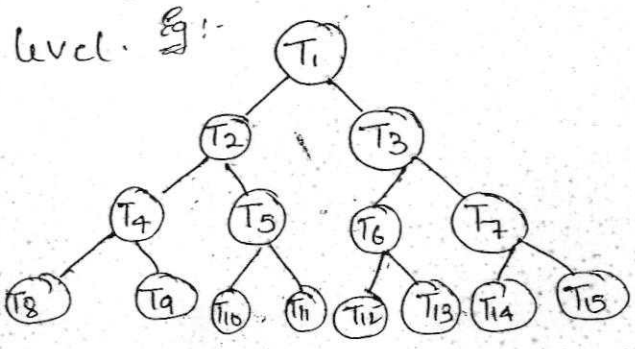


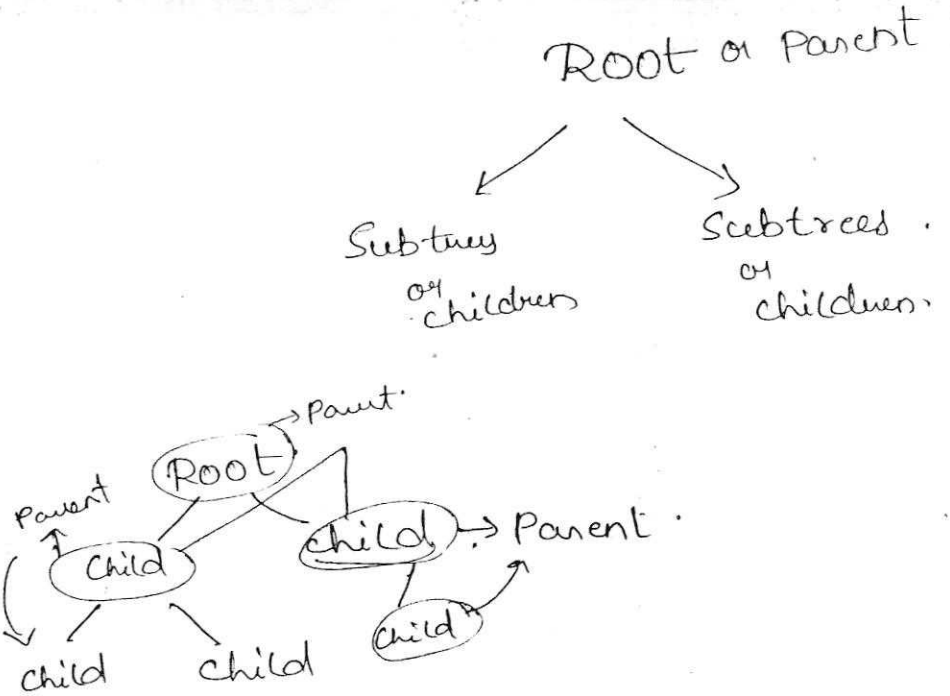
Full Binary Tree :- It is a binary tree and has the property of any node can have either 2 children or no children but not one children.



Complete Binary Tree :-

It is full binary tree in which all the leaf nodes are at same level. Eg :-

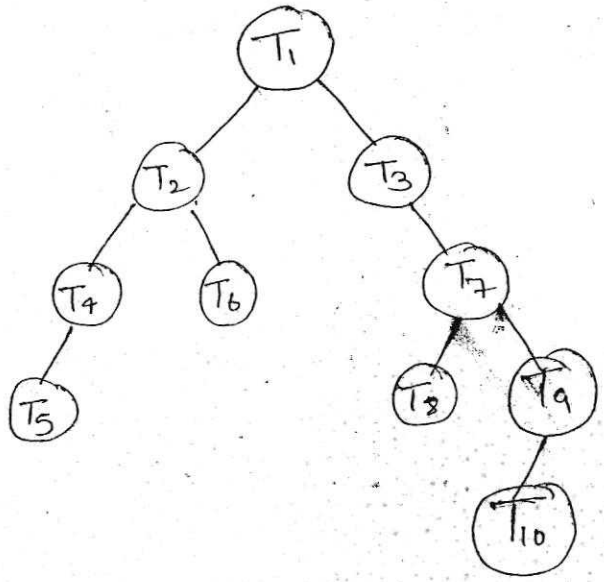




Binary Tree.

- Binary tree is a finite set of nodes which is either NULL or having one root node and a left binary tree, a right binary tree which are also called as left sub binary tree and a right sub binary tree. (or)
- A tree in which every node has atmost outdegree (children) '2'

Ex:-



13/7/16

TREES

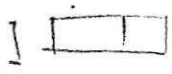
- Tree is a finite set of nodes having
  (i) A specially designated node called root of the tree.
  (ii) All other nodes can be Partitioned into disjoint sets $t_1, t_2, t_3 \ldots\ldots t_n$ (n disjoint sets)
  - Each of this sets is a sub tree of the given tree

Tree Terminologies :-

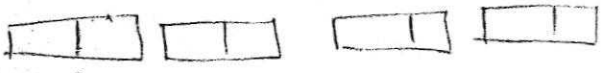
- root
- Parent node.
- child node.
- Sibling
- Degree of node / Tree.
- Level of the Tree.
- Height / Depth of the tree.
- Internal nodes
- external nodes / leaf nodes.
- Predessor & Successor :-

* A tree is a finite set of one or more nodes.

```
void delet_r( )
{
struct node * temp
if (front == NULL) {
  pf ("queue Empty");
}
elsif (r == f) {
      temp = r;
     r = f = NULL;
    free(temp);
}
else {
    temp = f;
    while (temp -> link != r) {
              temp = temp -> link;
          }
        temp -> link = NULL;
           r = temp;
    }
}

void delete _f ( ) {
     Struct node * temp;
      temp = f;
      f = f -> link;
      free (temp);
      }
```



```
void insert_r()
{
struct node* newnode;
int x;
newnode = (struct node*) malloc (sizeof (struct node));
newnode -> data = x;
newnode -> link = NULL;
if (f == NULL)
  {
  f = newnode;
  }
else
  {
  r -> link = newnode;
  }
r = newnode;
};
```



delete at r.

f → link = f.

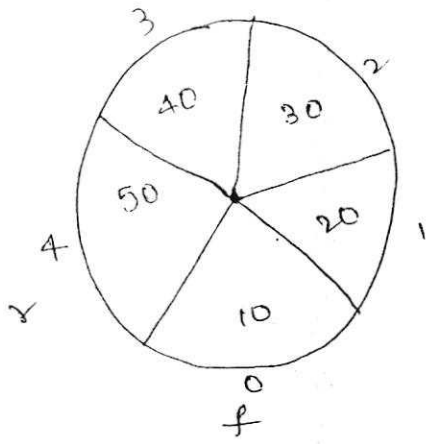
```
void insert_f()
struct node *newnode;
{
newnode =
  ——
  ——
if (f == NULL){
    f = r = newnode;
  }
else {
    newnode -> link = front;
    f = newnode;
  }
}
```

Deque (Linked list)

```
# include <Stdio.h>
# include <Stdlib.h>

Struct node{
int data;
Struct node * Link;
} * rear = NULL , * front = NULL ;

Void Create ()
{
Struct node *newnode, * Prevnode ;
int x, char choice;
do {
pf ("Enter the Numbers"),
sf (" %d ", & x);
newnode = (Struct node*) malloc (size of (Struct node));
newnode → data = x;
newnode → Link = NULL;
        if (root == NULL)
                root = new node;
        else
            Prevnode → link = newnode;
    Prevnode = newnode;
    pf (" do you wish to Continue (y/n) ");
    getchar();
    choice = get char();
    }
    While (Choice == 'y');
    }
```

```
void display( ) {
Struct node *temp = front ;
While ( temp != NULL ) {
   Pf(" %d ", temp -> data)
    temp = temp -> link ;
   }
  }.
```

Circular Queue.



→ In circular queue is to known whether the que is full. rear should be always followed by front.

→ If rear is not followed by front then we can move rear by $\boxed{(R+1) \% \text{ size}}$.

```
# define SIZE 5
void enqueue (int x)
{
 if ((r+1) % SIZE == f)
   pf ("CQ full");
 else {
      if (f == -1)
        { f = r = 0 ;
        }
      else
        {
        r = (r+1) % SIZE ;
        }
      CQ [r] = x ;
 }.
```

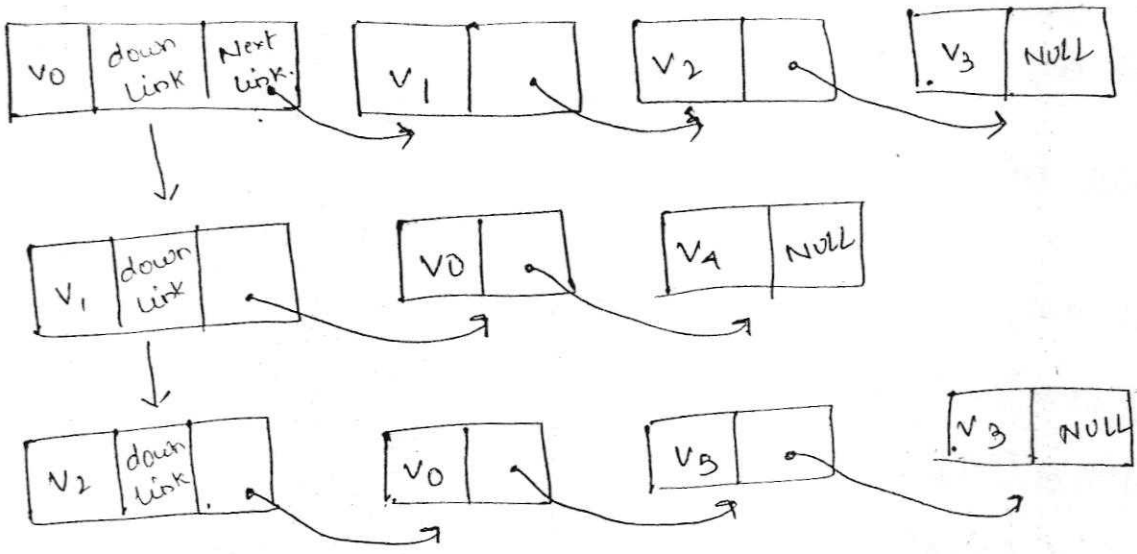
23/7/16

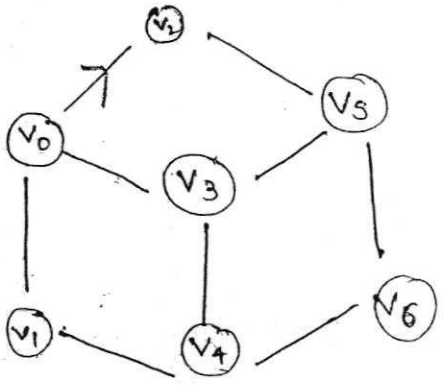
GRAPHS

Adjacency Matrix representation:

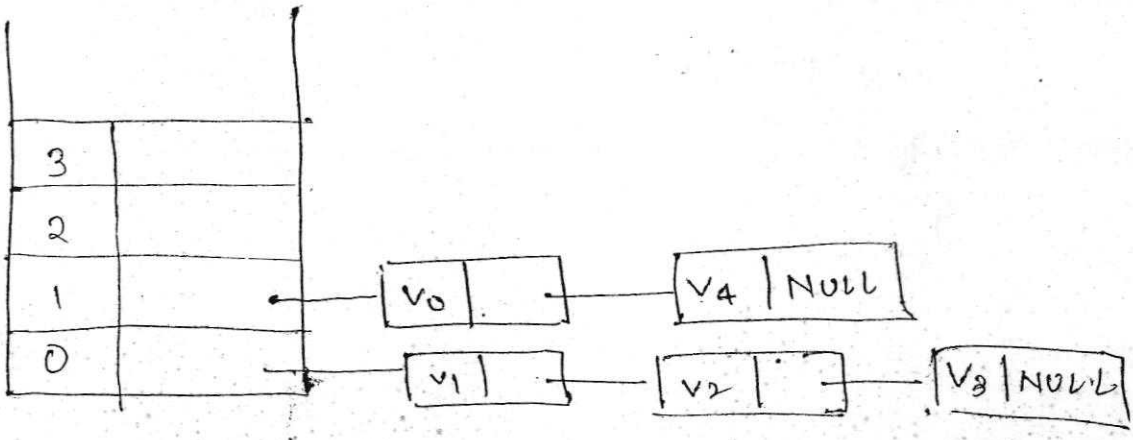
Adjacency link Representation (Linked list):

1st approach:





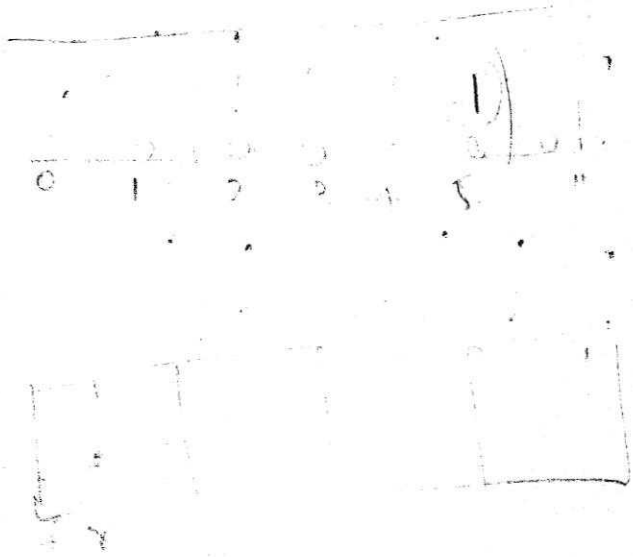
2nd approach:



Traversing Graph Using BFS and DFS:

(breadth first Search) (Depth first Search)

```
#include <stdio.h>
int Q[20], f=-1, r=-1;
int G[20][20];
int visited[20], visited2[20];
int n;

void bfs (int v1)
{
int v2;
Q[++r] = v1;
visited[v1]=1;
while (f!=r)
{
v1 = Q[++f];
pf("%d", v1);
for (v2=0; v2<n; v2++)
{ if (G[v1][v2]==1 && visited[v2]==0)
  {
    Q[++r]=v2;
    visited[v2]=1;
  }
}
}
}
```

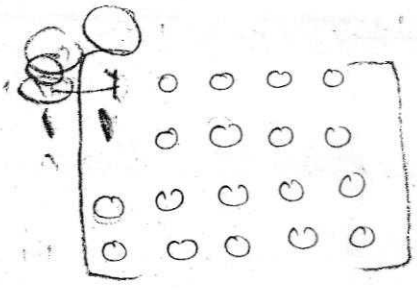


```
void
{
int v
pf("
visit
for(
{
i
}
}
}

void
{
int v
char
pf("
sf
//init
for(
for(
G[
pf(
do
{
pf("
sf("
```

DFS

Depth first Search

```
void dfs(int v1)
{
 int v2;
 pf("%d", v1);
 visited2[v1]=1;
for(v2=0; v2<n; v2++)
   { if(G[v1][v2]==1 && visited2[v2]==0)
     { dfs(v2);
     }
   }
}

void main()
{
int v1, v2, v;
char ch;
pf("enter the number of vertices:");
sf("%d", &n);
//initializing the adjacency matrix G of the graph to 0.
for(v1=0; v1<n; v1++)
for(v2=0; v2<n; v2++)
 G[v1][v2]=0;
pf("\n enter the edges details:");
do
{
pf("\n enter source vertex and destion vertex");
sf("%d %d", &v1, &v2);
```



```
    G[v1][v2]=1;
    pf("\n add more edges (y/n)");
    getchar();
    ch = getchar();
    }
while (ch=='y');
pf ("\n the adjacency matrix for the graph is :\n\n");
for (v1=0; v1<n; v1++)
 {
for(v2=0; v2<n; v2++)
{
 pf("%d", G[v1][v2]);
 }
 pf("\n");
}
// intializing visited status to not visited for all the
   vertices
for(v=0; v<n; v++)
 {
  visited[v]=0;
  visited2[v]=0;
 }
pf("\n enter the starting vertex to transverse the graph:");
sf("%d", &v);
pf("\n traversing using bfs : \n");
bfs(v);
pf("\n traversing using dfs : \n");
dfs(v);
 pf("\n");
}
```

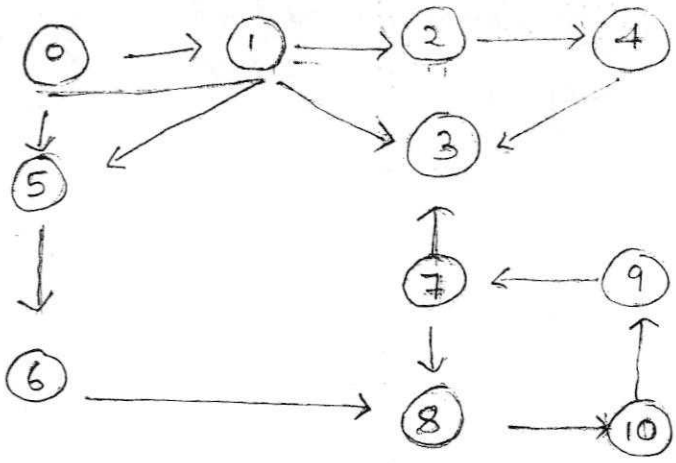
Edges

v1 v2

6 - 1
0 -
1 - 2
7 - 3
1 - 5
2 - 4
4 - 3
5 - 6
6 - 8
7 - 3
7 - 8
8 - 10
9 - 7
10 - 9

DFS

BFS

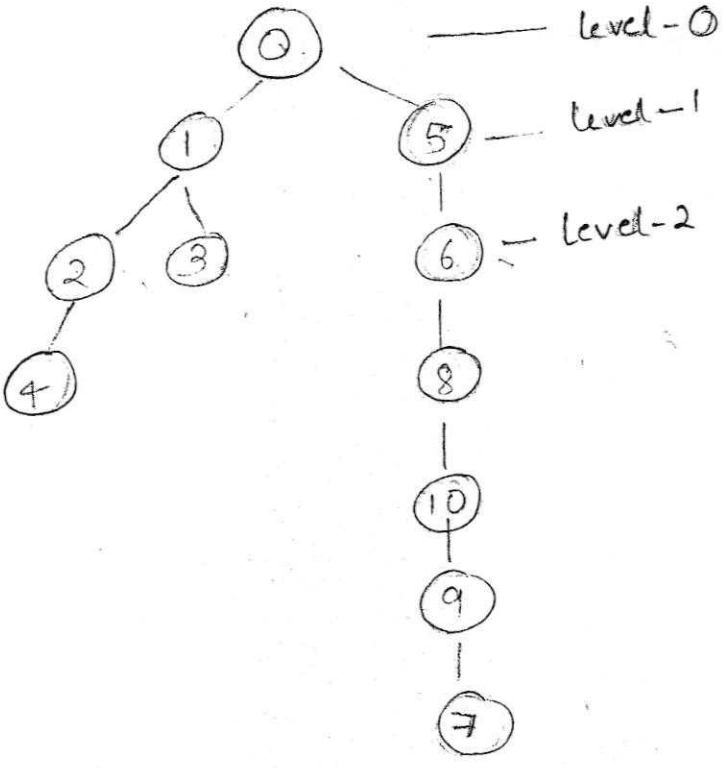


11 Vertices

/n");

Edges

V_1	V_2
0	1
0	5
1	2
1	3
1	5
2	4
4	3
5	6
6	8
7	3
7	8
8	10
9	7
10	9



BFS - Q

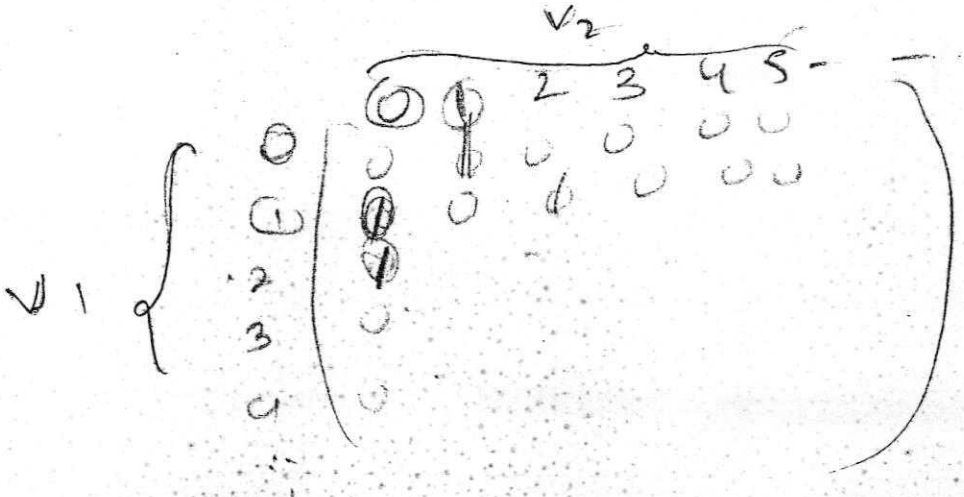
DFS - S

all the

DFS :- 0, 1, 2, 4, 3, 5, 6, 8, 10, 9, 7

graph:");

BFS :- 0, 1, 5, 2, 3, 6, 4, 8, 10, 9, 7



20/8/16

UNIT 4

Searching & Sorting.

Linear search.

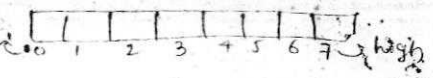
```
# include <stdio.h>
void main() {
 flag=0', pos',
for (i=0', i<n', i++)
{
 if (A[i] == key)
   {
    flag = 1
    pos = i+1',
    break',
   }
}
if (flag ==0)
 Pf (" not found");
else
  Pf(" found at = %d", pos)',
}.
```

Binary

[In th
are
whe
or
Value

```
# incl
void
flag=0
low = 0
if
else i
else
if (f
else
}
```

Binary Search:



[In this we consider lower bound and high bound and then calculate the mid value, and check whether the key is towards left array or right array and then change the mid value accordingly].

```
#include <stdio.h>
void main(){
flag=0, pos;
low=0, high=n-1;
while (low<=high)
    {
    mid = (low+high)/2;
  if (a[mid]==key)
    {flag=1;
        pos=mid+1;
        break;
    }
else if (key>a[mid])
    low=mid+1;
else
    high=mid-1;
if (flag==0)
    pf("not found");
else pf("found at=%d", pos);
}
```

23/8/16

Hashing

UNIT – IV [4]

Hash Table : This is a data structure which stores, has values associated with a hash key.

for example:

if we wanted to store & access employee records with attributes employee no:, name, designation,

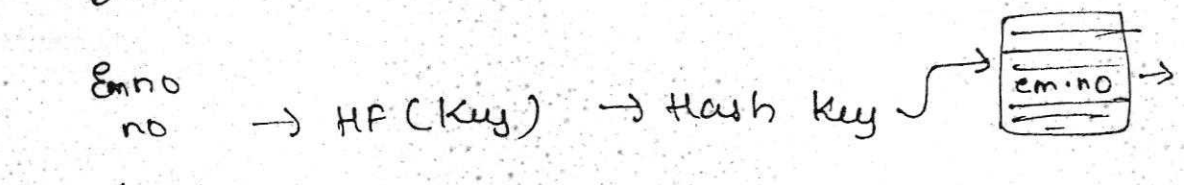
Then we consider one of the field like as key field like employee no: for above record.

Hash function takes this key as an i/p & produces an hash key which will be used as an index in to the hash table. to place the corresponding record of the given key at that location.

Same procedure is used to retrive or search a record with a given key.

This process can locate a given record with one (or) few searches.

Hash tables are used to implement, Dictionaries, consisting of key, value pair.



Hashing Methods / functions

1) Division Method.
2) Mid Square Method
3) Multiplication Method
4) Folding Method.

Division Method :

- In this method, we divide the key with the Hash table size, which gives the address of the key.

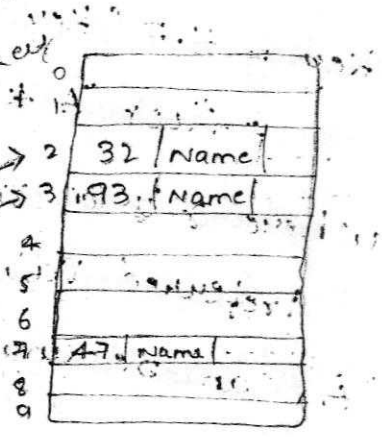
Ex 1 :-

Let the H.T size = 10

key = 47 $\quad 47 \% 10 = 7$

key = 32 $\quad 32 \% 10 = 2$

key = 93 $\quad 93 \% 10 = 3$



Ex 2 : Let H.T Size = 100

key = 12475 $\quad 12475 \% 100 = 75$

Mid Square Method

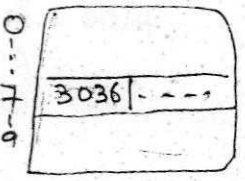
We ~~used~~ Square the key value and then find the mid number of the squared value, which will be the address of the key.

Ex :- key = 3036

$HF(key) = (3036)^2 = 9216296$

If H.T size is 10

then addres is 7



Ex 2:- 92|17296

If HT size is 100
then address / index either 17 or 72

[for 7 we can
Consider left no. or right number
but it should be same for all the values]

Ex 3:-

92|17296

If HT Size is 1000. (0-999)
Then 172 is the Hash key

Multiplication Method:-

Hash key = H.F (key)
= floor (A * (key * r))

where r is a real number.

Preferred value for r = 0.61803

'A' can be any +ve int Ex:- 5 (but should be same for all the values)

Ex:-

key = 25

Hash key = floor (5 * (25 * 0.61803))
= floor (5 * 15.451)
= floor (77.255)
= 77.

floor (3.7) } = 3
floor (3.1) }

round (3.7) = 4
round (3.1) = 3

Seal (3.7) } = 4
Seal (3.1) }

Fold

[-

key

let

Hash

Collis

If th

for

colli

Chara

1) If

2) If

no

3) The

key

4) If

collision

1) Chai

2) line

3) quad

4) Do

Folding Method:

12

[fold & Sum]

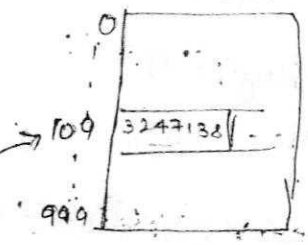
Key = 3247138

Let the HT Size is 1000 (0-999 indexes)

Hash key = 324|713|8

```
  324
  713
+   8
-----
 1045
```

104+5 = 109



Collision:

If the hash function producing same hash key for two different key values then it is called collision.

Characteristic of a good hash function:-

Id be
r all the
values)

1) It should be easy/simple to compute.
2) It should generate very few collisions; ideally no collisions at all
3) The function should evenly distribute, hash keys, across the hash table
4) It should operate on every bit of the i/p key

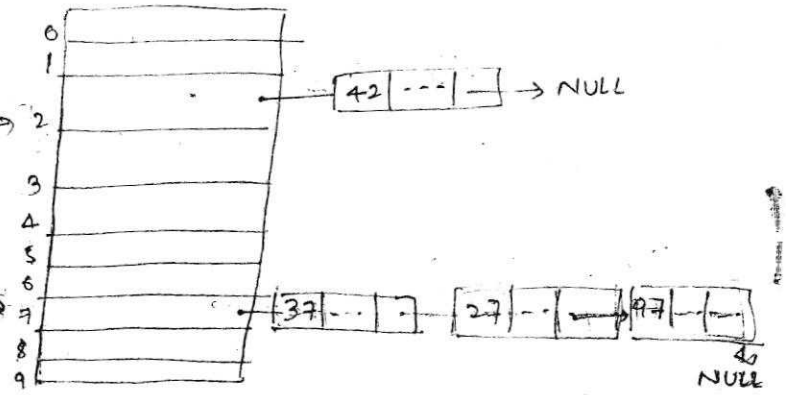
collision handling Methods:

1) Chaining
2) Linear Probing
3) quadratic Probing
4) Double hashing

Chaining :

Method = Division Method.

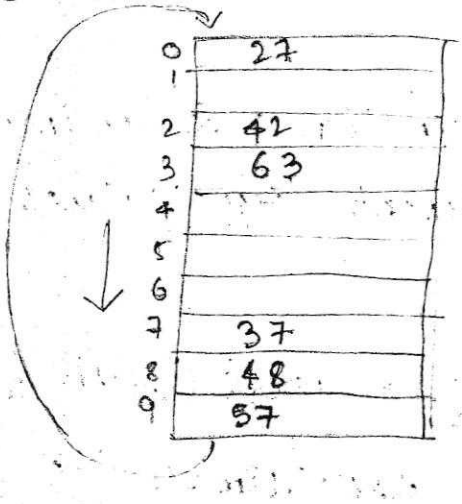
Key = 37 = 7
Key = 42 = 2
Key = 27 = 7
Key = 97 = 7



Linear Probing

Method = Division method.

Key = 37 = 7
Key = 42 = 2
Key = 48 = 8
Key = 57 = 7
Key = 63 = 3
Key = 27 = 7



Quadratic Probing

If collision occurs for given key

Then

Hash key = $(key + x^2)$ % H.T Size

where $x = 1, 2, 3, \ldots\ldots$

Ex: 1

key = 37 if collision occurs.

Step 1 :- Take $x = 1$ Hash key = $(37 + 1^2)$ % 10 = 38 % 10 = 8

Take $x = 2$ Hash key = $(37 + 2^2)$ % 10 = 41 % 10 = 1

[If this index is free use this else go to next 'x' value]

Take $x = 3$

.
.
.

Double Hashing :-

HF1 (key) = hash key

HF2 (key) = M - hash key = P

7) Now jump forward by P locations from collision index and place the record there

Key = 34

HF(34) = 34 % 10 = 4

Choose M as the biggest prime no less than Size of the table

if Size = 10

M = 7

HF 2(34) = |M - 4|

= |7-4|

= 3

collision index = 4

P = 3

jump to 7th index & Place the record.

Rehashing :-

- The Size of the hash table will be doubled to its nearest integer Prime number, whenever, either the hash table becomes full (or overflow occurs) or the hash function Producing mor collisions.

17/9/16

Sorting Techniques

BUBBLE SORT

```
for (i=1 ; i<n ; i++)
{
 for (j=0 ; j<n-1 ; j++)
 {
   if (A[j] > A[j+1])
   {
     temp = A[j]
     A[j] = A[j+1]
     A[j+1] = temp
   }
 }
}
```

SELECTION SORT :-

Eg :-

10	13	7	53	62	69	4	46	35	12
0	1	2	3	4	5	6	7	8	9

Step 1 :

10 13 7 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

7 13 10 53 62 69 4 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

4 13 10 53 62 69 7 46 35 12

Step 2:- 4 13 10 53 62 69 7 46 35 12

4 7 13 53 62 69 10 46 35 12

Step 3:- 4 7 13 53 62 69 10 46 35 12

Selection Sort

```
for (j=0 ; j< n-1 ; j++)
for (k=j+1 ; k<n ; k++)
 {
   if [A[j] > A[k]]
     {
       temp = A[j] ;
       A[j] = A[k] ;
       A[k] = temp ;
     }
 }
```

Insertion Sort :

10	13	7	53	62	69	4
0	1	2	3	4	5	6

n = 7

```
while (k != n)
{ for (i = k ; i <= 0 ; i--)
  {
    j = i + 1
    if (A[i] > A[j])
    {
      swap;
    }
    k++ ;

for (i = 1 ; i < n ; i++)
{
   j = 1 ;
 while (j > 0 && [j] < A[j-1])
   { swap;
      j-- ;
   }
}
```

Double
HF1 (k
HF2 (
Now
Key = 3.
HF (34)
Choo

Remo
- The
necess
the
occur
collis

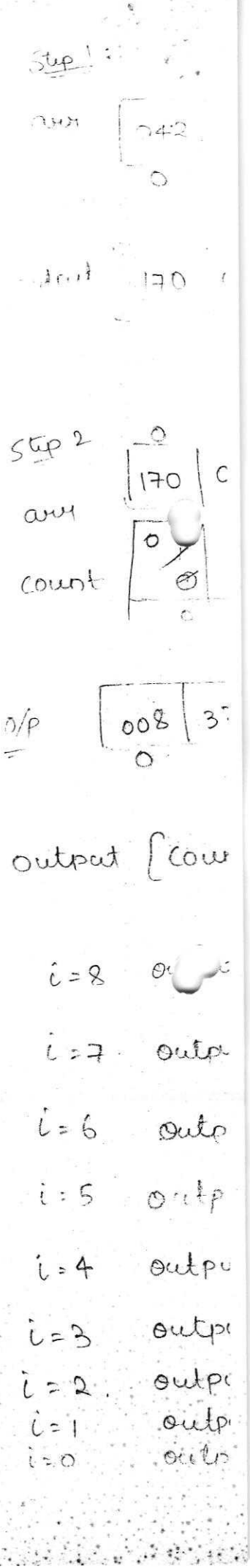
Radix Sort :-

```
# include <stdio.h>
int getMax(int arr[], int n) {
 int mx = arr[0];
 int i;
 for(i=1; i<n; i++)
   if (arr[i] > mx)
     mx = arr[i];
  return mx;
}
Void Countsort (int arr[], int n, int exp)
{
 int output[n]; // output array.
 int i, count[10] = {0};
 // Store count of occurrences in count[].
  for (i=0; i<n; i++)
   count[(arr[i]/exp) % 10]++;
  for (i=1; i<10; i++)
    count[i] += count[i-1];
// Build the output array.
for (i=n-1; i>=0; i--)
{
  output[count[(arr[i]/exp) % 10]-1] = arr[i];
  count[(arr[i]/exp) % 10]--;
}
```

a += b
a = a + b

count[i] = count[i] + count[i-1]

```
    for (i=0 ; i<n ; i++)
     arr [i] = output [i];
    }
// The main function to that sorts arr[] of size n
   using Radix sort.
Void radixsort (int arr[], int n){
   int m = get Max (arr, n);
   int exp;
   for (exp=1 ; m/exp>0 ; exp *=10)
     countSort (arr, n, exp);
   }
void print (int arr[], int n)
{
   int i;
   for(i=0 ; i<n ; i++)
    Printf(" %d", arr[i]);
   }
t main()
{
   int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
   int n = sizeof(arr)/sizeof(arr[0]);
    radixsort (arr, n);
    print(arr, n);
    return 0;
}
```



Step 1:

arr

042	170	323	894	666	047	447	170	008
0	1	2	3	4	5	6	7	8

output

170	042	323	894	666	047	008	179	447

arr[] of size n

Step 2

arr

0	1	2	3	4	5	6	7	8
170	042	323	894	666	047	008	179	449

count

0	1	2	3	4	5	6	7	8	9
1	1	2	3	3	5	5	6	8	8

O/P

0	1	2	3	4	5	6	7	8
008	323	042	047	449	666	170	179	894

output [count [(arr[i]/exp) % 10] - 1] = arr[i]

↳ last but 1 digit

$i = 8$ output [count [4] − 1] ⇒ op [5-1] = op[4] = arr[8]

$i = 7$ output [count [7] − 1] ⇒ op [8-1] = op [7] = arr[7]

$i = 6$ output [count [0] − 1] ⇒ op [1-1] = op [0] = arr[6]

$i = 5$ output [count [4] − 1] ⇒ op [4-1] = op[3] = arr[5]

$i = 4$ output [count [6] − 1] ⇒ op [6-1] = op[5] = arr[4]

$i = 3$ output [count [9] − 1] ⇒ op [9-1] = op[8] = arr[3]

$i = 2$ output [count [2] − 1] ⇒ op [2-1] = op[1] = arr[2]

$i = 1$ output [count [4] − 1] ⇒ op [3-1] = op[2] = arr[1]

$i = 0$ output [count [7] − 1] ⇒ op [7-1] = op[6] = arr[0]

Step 3:

	0	1	2	3	4	5	6	7	8
arr	008	323	042	047	449	666	170	179	894

	0	1	2	3	4	5	6	7	8	9
Count	3 ~~0~~	~~2~~ ~~0~~	0 ~~0~~	1 ~~0~~	1 ~~0~~	0 ~~0~~	1 ~~0~~	0 ~~0~~	1 ~~0~~	0 ~~0~~

	0	1	2	3	4	5	6	7	8
O/P									

~~Output~~ = [count[(arr[i]/exp) % 10] - 1] = arr[i]

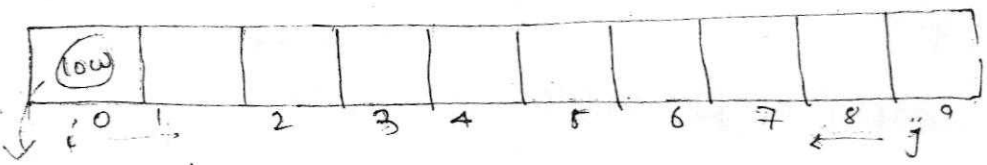
i = 8 output [Count[8] - 1] ⇒ op[

Radix So

```
#includ
int get
 int mx
int i;
for(i=1;
  if (a
    mx
  retur
}
Void Cou
{
int Out
int i;
// Sto
 for (i
  Cou
 for (
   Co
// Buil
for(i
{
  out
  Cou
}
```

27/9/16

Quick Sort :-



Pivot element

Pivot = A[low]

Increment i such that it should be greater than the Pivot element A[i] ≤ Pivot

Move j towards left (decrement) side such that the value should be less than the Pivot element A[j] > Pivot.

After the increment of i & j values.

If i < j (index values)

Swap the elements

If j < i then Swap A[j] with Pivot value or A[low]

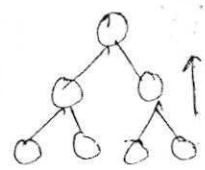
```
# include <stdio.h>
int Partition (int A[10], int low, int high)
{
 int Pivot = A[low], i, j ;
  i = low;
  j = high;
 while (i ≤ j)
 {
   while (A[i] ≤ Pivot)
       i++;
   while ( A[j] > Pivot)
       j--;
```

this index see use this go to next 'x' value

```
    if (i<j)
    swap (A[i], A[j]);
  }
swap (A[j], A[low]);
return j;
}
void quicksort (int A[10], int low, int high)
{
  int k;
if (low < high)
{
k = partition (A, low, high);
quicksort (A, low, k-1);
quicksort (A, k+1, high);
}
}
void swap (
{
  ____
  ____
  ____
}
void main()
{
  int A[10];
  for (i=0; i<j; i++)
  pf
  sf
  read A.
  display A.
  quicksort (A, 0, 9);
  display A.
}
```

29/9/16

Heap Sort:

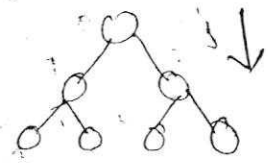


```
Void makeheap [int A[10], int n)
{
 int i, val, j, Parent;
 for(i=1; i<n; i++)
 {  val = arr[i];
    j = i;
  Parent = (j-1)/2;
  while (j>0 && A[Parent]<val)
   {
     A[j] = A [Parent];
        j = parent;
     Parent = (j-1)/2..
   }
     A[j] = val;
   }
 }
```

Here i : index value

8	7	10	26	16	45	89	43	69	39	54	17
0	1	2	3	4	5	6	7	8	9	10	11

```
Void heapsort [int A[10], int n)
{
  int i, k, temp, j;
  for (i=n-1; i>0; i--)
  {
    temp = A[i];
    A[i] = A[0];
    k=0;
    if (i==1)
    j = -1;
  else
    j = 1;
  if (i>2 && A[2] > A[1])
  j=2;
while (j>=0 && temp < A[j])
{
  A[k] = A[j];
  k=j;
  j = 2*k+1;
  if (j+1<=i-1 && A[j]<A[j+1])
  j++;
  if (j>i-1)
    j=-1;
}
```



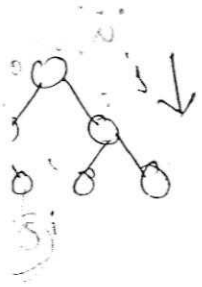
k = Parent index
j = child index

Here i = no. of elements to delete.
(and)
also for no. of elements it should consider for each iteration.

(going to left child)

A[k] = temp;

}



element to

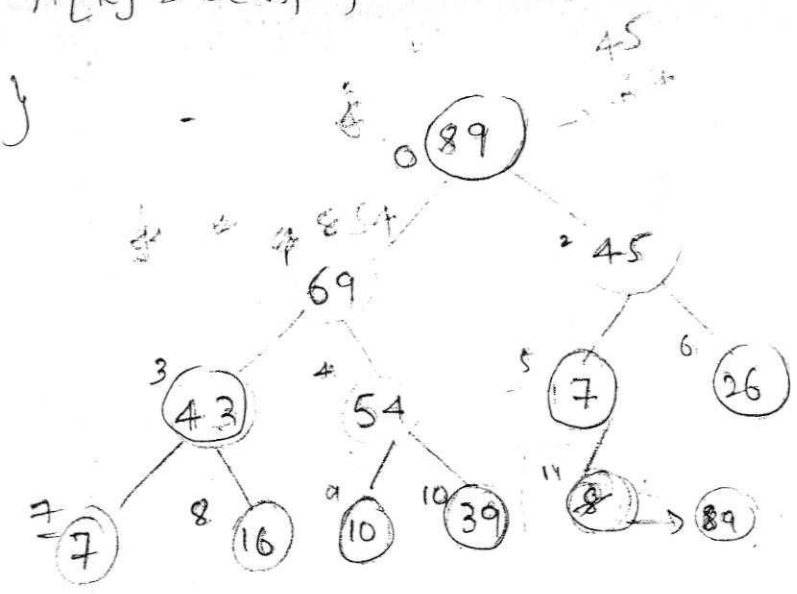
and)

or no. of

it should

for each

0.



Step 1

i = 11 ; 11 > 0

temp = A[i] = 8

A[i] = 89 ; j = 1

k = 0

11 > 2

45 > 69 ✗

i >= 0

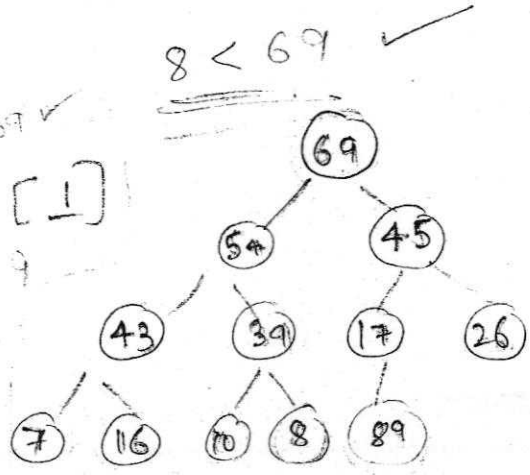
8 < 69 ✓

8 < 69 ✓

0 - A[0] = A[1]

A[0] = 69

k = 1



After Step 1

j = 0

j = (2 * 1) + 1

= 2

j = 3

A[3] < A[4]

54 < 17 ✗

43 < 54 ✓

k = 1 j = 4

4 > 11 - 1 ✗

A[9] < A[

UNIT - V

Similar to Binary tree except that the elements to the left side of any node must be smaller than the node (Parent) and the elements to the right must be greater or equal to the parent node value

```
#include<stdio.h>
#include<stdlib.h>

struct tree
{
  struct tree *left;
  char data;
  struct tree *right;
} *root = NULL;

void insert(struct tree *temp, struct tree *nn)
{
  char ch;

  if (nn->data < temp->data)
  {
      if (temp->left == NULL)
          temp->left = nn;
      else
          temp=insert(temp->left, nn);
  }
```

```
else
{
  if (temp -> right == NULL)
      temp -> right == nn;
  else
      insert (temp -> right , nn);
}
} // end of insert function.

void create ()
{
  struct tree *nn;
  char ch; char x;
  do
  {
    printf ("Enter data");
    x = getchar ();
    nn = (struct tree *) malloc ( sizeof (struct tree));
    nn -> left = NULL;
    nn -> data = x;
    nn -> right = NULL;

    if (root == NULL)
        root = nn;
    else
        insert (root, nn);
```

```
printf("\n Do you wish to continue (y/n)");
getchar();
ch = getchar();
getchar();
}
while( ch == 'y');
} // end of create function.

// inorder Traversal
void display( struct tree *temp)
{
  if (temp != NULL)
    {
      display(tem->left);
      Printf("%c\n", temp->data);
      display(temp->right);
    }
}


void main()
{
  Create();
  display(root);
}
```

(Adelson, velski & Landis)

An AVL Tree is a binary search tree in which the difference of heights of left sub tree and right subtree for any node must be either -1, 0, +1.

inserting into AVL tree

Case (i) :- inserting a node into the left child of the left sub tree
(if tree unbalanced, use LL Rotation)

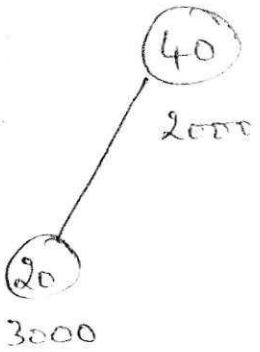
Case (ii) :- inserting a node into the right child of the left sub tree
(if tree unbalanced, use LR Rotation)

Case (iii) :- inserting a node into the right child of right sub tree
(if tree unbalanced,
use RR Rotation)

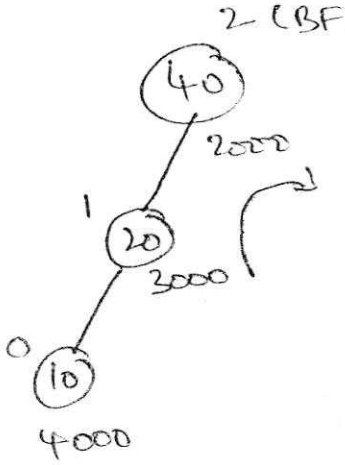
Case (iv) :- inserting a node into the left child of right subtree.
(if tree unbalanced,
use RL Rotation)

Scenario's for LL Rotation

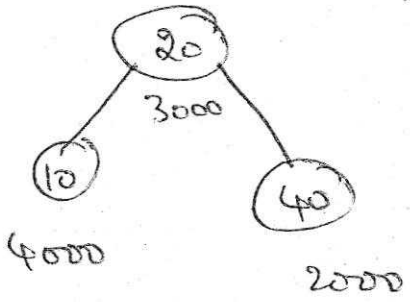
Ex 1 :-



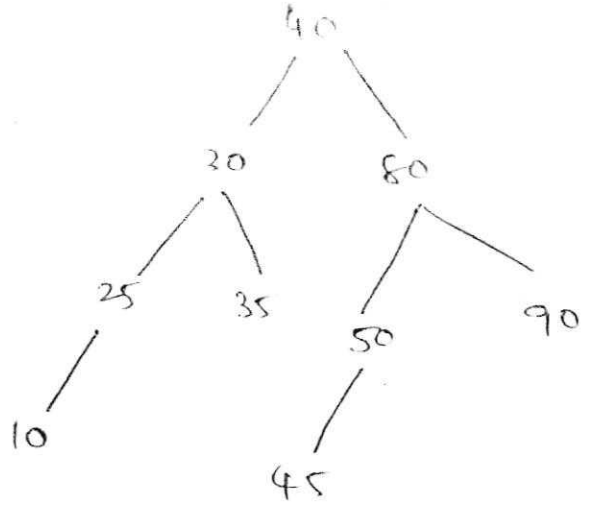
Insert 10.



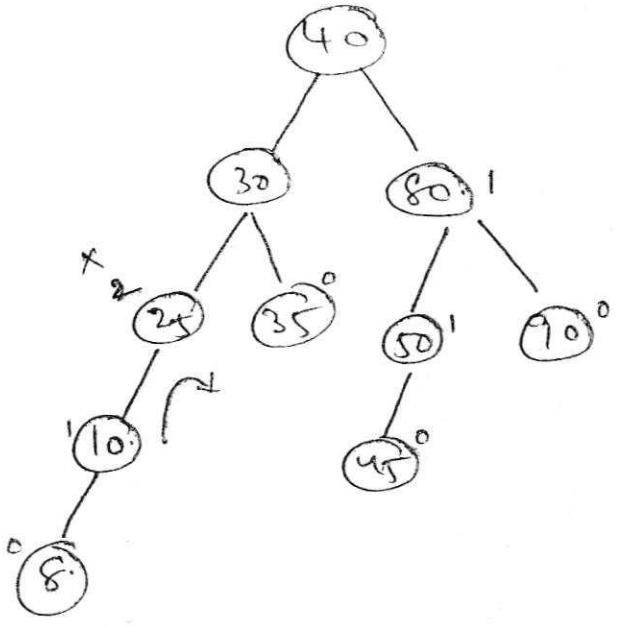
For the node at address 2000, the balance Factor = 2 which is not accept ⇒ apply LL Rotation.
(ie Right Rotation)



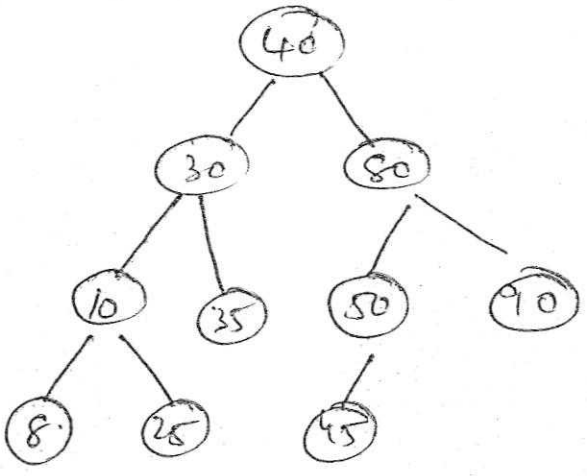
Ex:



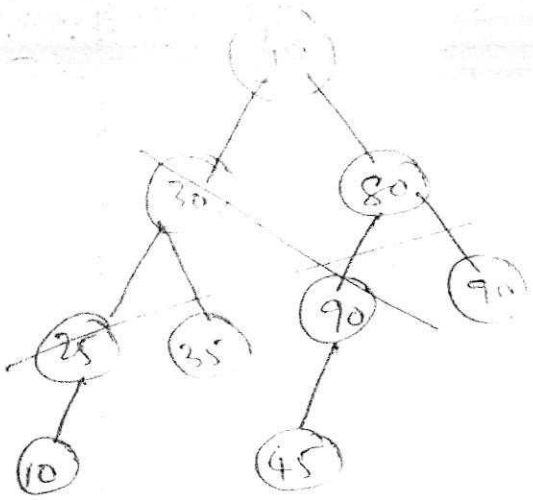
insert '8'.

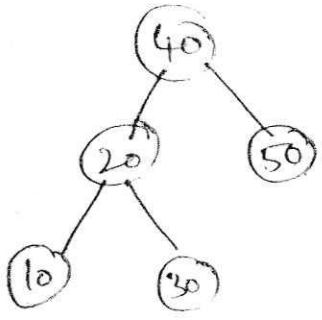


apply LL Rotation (



Ex3 -

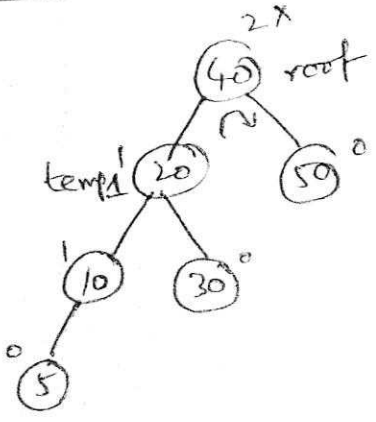




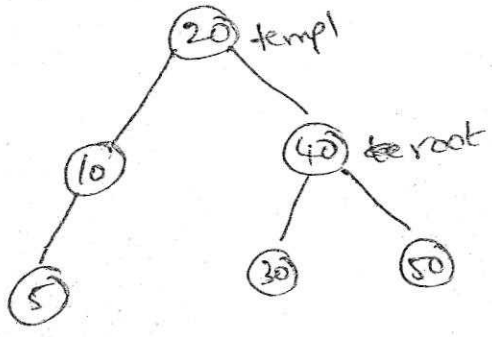
(a) insert 5

(b) insert 15

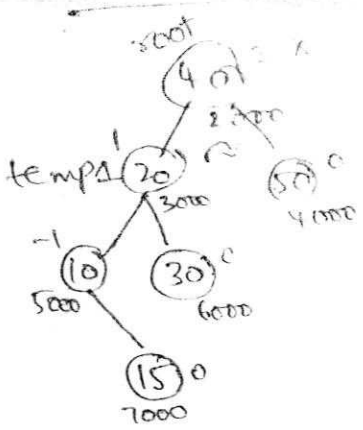
(a) insert 5



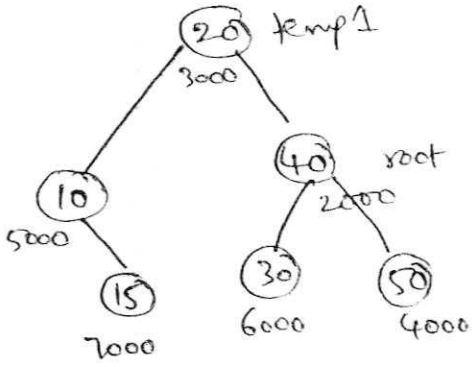
apply LL Rotation



(b) Insert 15



apply LL Rotation.



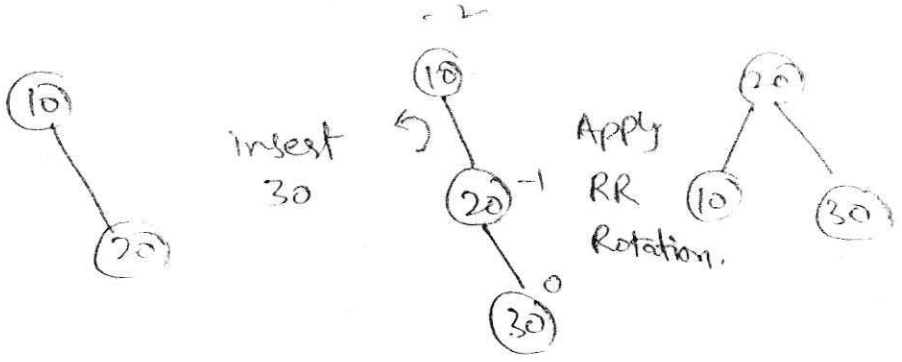
Code for LL Rotation

```
temp1 = root -> left
root -> left = temp -> right
temp1 -> right = root ,   root -> BF = 0.
root = temp1
```

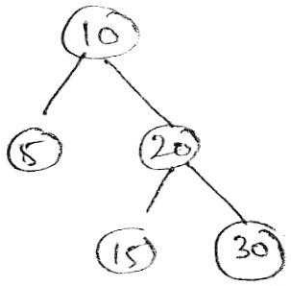
root -> BF = 0.

Examples of RR Rotation

Ex1



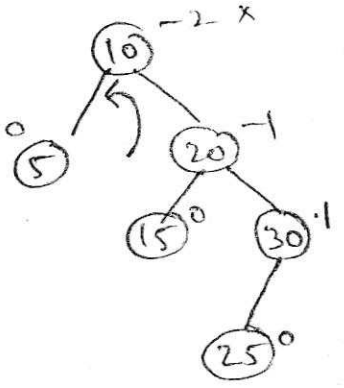
Ex2



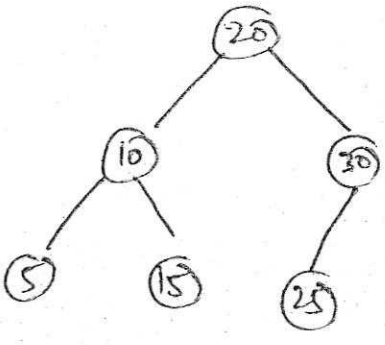
(a) insert 25

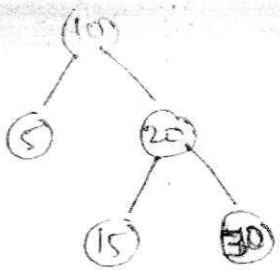
(b) insert 40

(a) insert 25

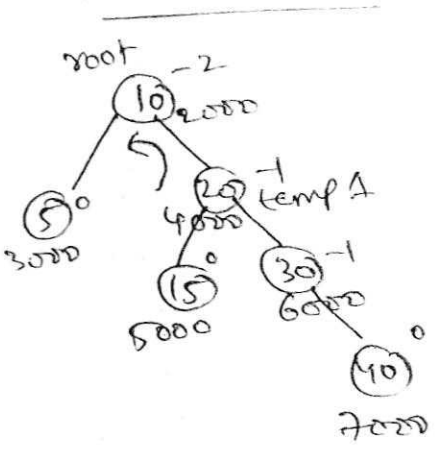


apply RR Rotation.

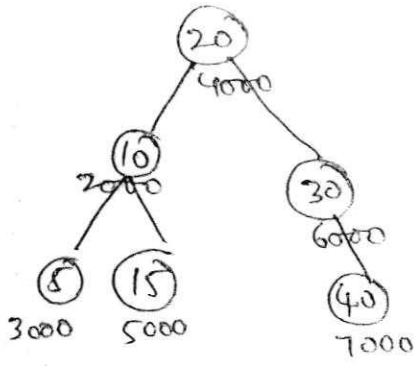




(5) Insert 40



apply RR Rotation



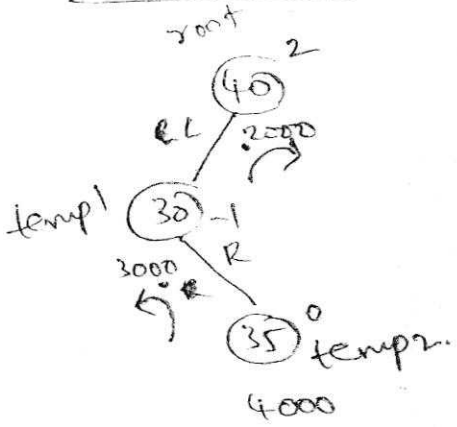
RR Rotation Code :—

```
temp1 = root -> right

root -> right = temp1 -> left;
temp1 -> left = root;

root -> BF = 0;

root = temp1;

root -> BF = 0;
```

Examples :- 1 : LR Rotation

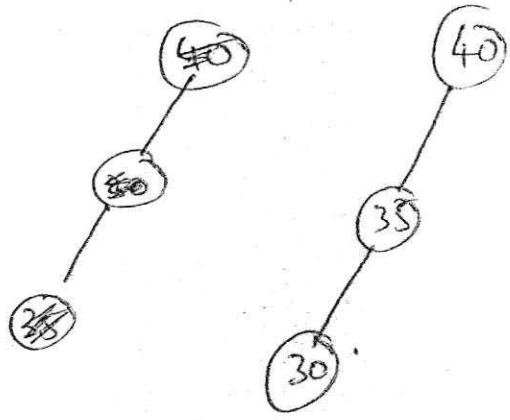
Ex1

Insert 35.

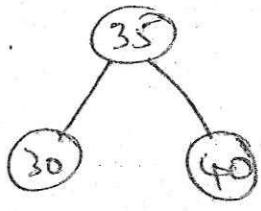


apply double Rotation. (LR)

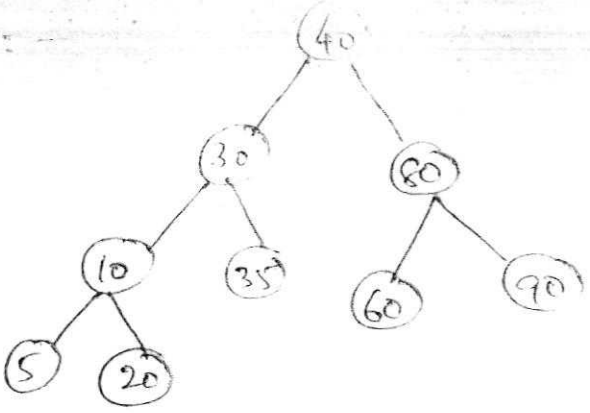
Step1 :- ~~apply~~



Step2



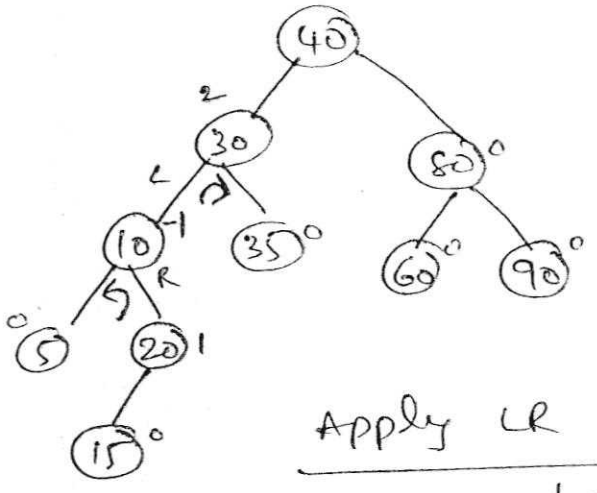
Ex 2 :-



(a) insert 15

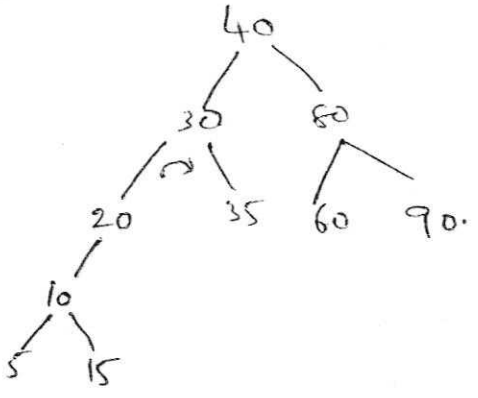
(b) insert 25

(a) insert 15.

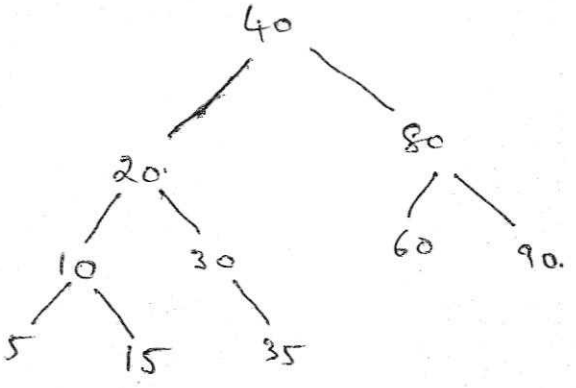


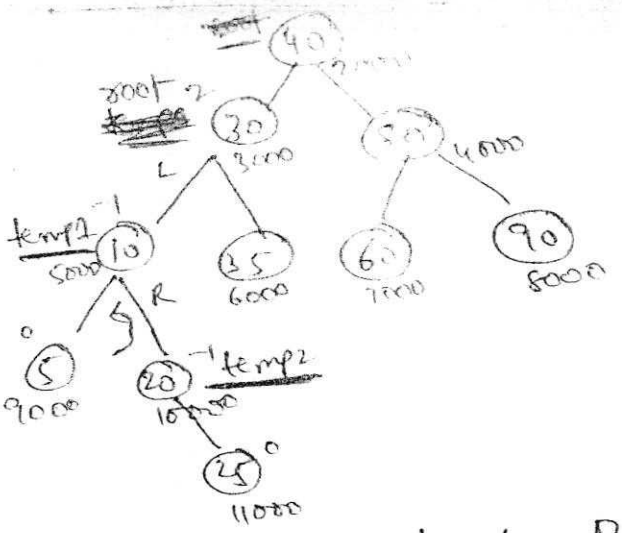
Apply LR Rotation

Step1 :-



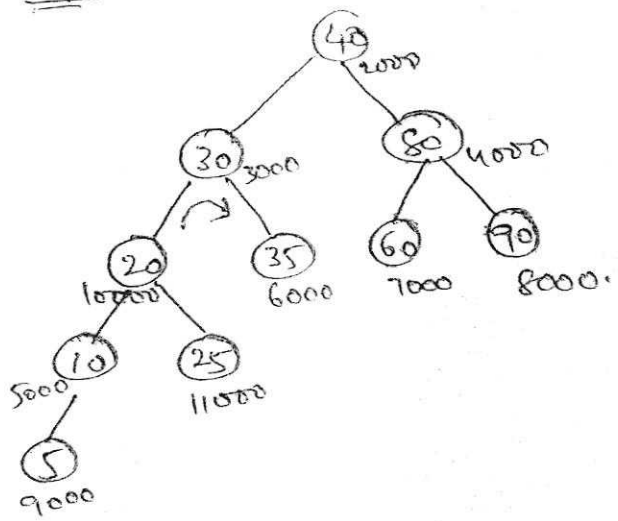
Step2 :-



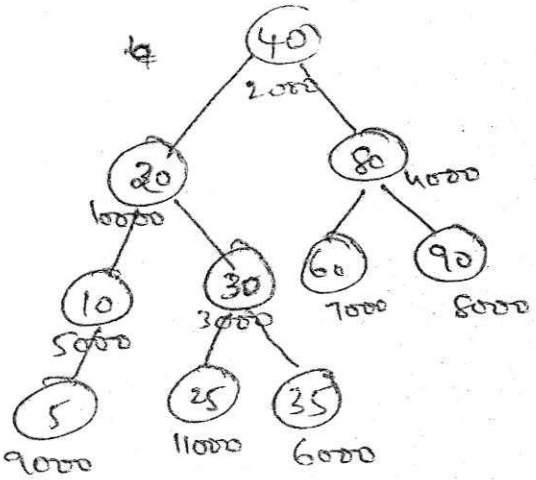


apply LR double Rotation

Step1



Step2

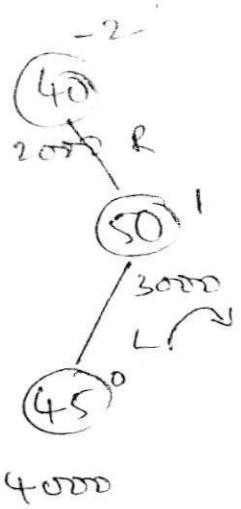


Code

```
temp2 = temp1->right
temp1->right = temp2->left
temp2->left = temp1
root->left = temp2->right
temp2->right = root;

if (temp2->BF == 1)
    root->BF = -1;
else
    root->BF = 0;

if (temp2->BF == -1)
    temp1->BF = 1;
else
    temp1->BF = 0;

root = temp2;

root->BF = 0.
```

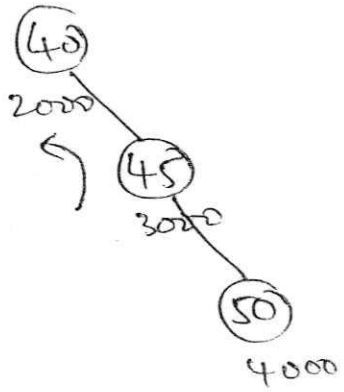
Example for RL Rotation :-

Ex 1 :-

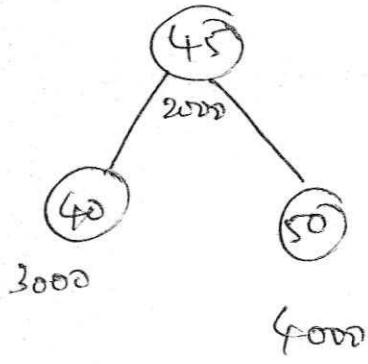


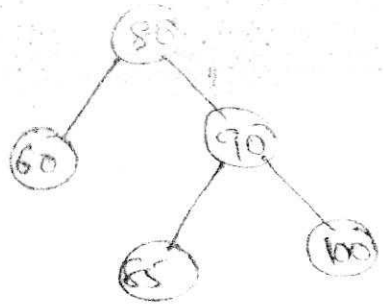
apply double Rotation.
(RL Rotation)

Step 1



Step 2

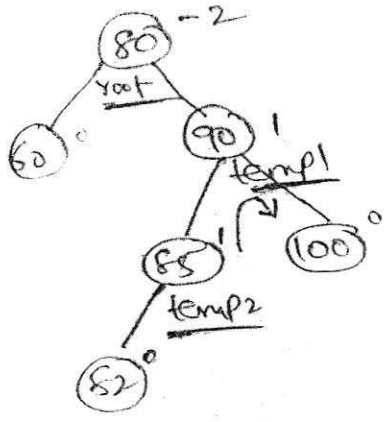




(a) Insert 82

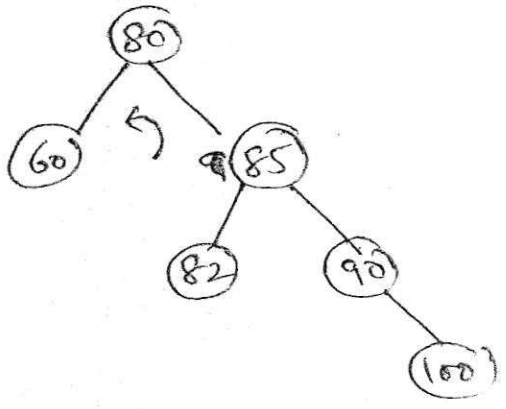
(b) Insert 86

(a) insert 82.

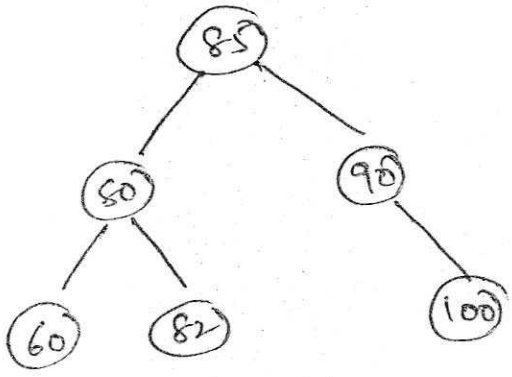


Apply RL Rotation (Double Rotation)

Step1 :-



Step2 :-



Code for RL Rotation

```
temp1 = root -> right

temp2 = temp1 -> left ;

temp1 -> left = temp2 -> right ;

temp2 -> right = temp1 ;

root -> right = temp2 -> left ;

temp2 -> left = root ;

if (temp2 -> BF == -1)
        root -> BF = 1 ;
else
        root -> BF = 0 ;

if (temp2 -> BF == 1)
        temp1 -> BF = -1 ;
else
        temp1 -> BF = 0 ;

root = temp2

root -> BF = 0 ;
```

Task 1 :-

Insert the following elements into the AVL Tree.

40, 30, 20, 60, 50, 80, 15, 28, 25.

Task 2 :-

Insert the following elements into the AVL Tree

A, V, L, T, R, E, I, S, O, K.

```
#include < stdio.h>
#include < stdlib.h>

struct avlnode
{
   struct avlnode *left;
   int data;
   int bf;
   struct avlnode *right;
};

typedef struct avlnode node;

node *root;

node *  Insert( node *root, int data, int *Curren
{
 node *temp1, *temp2;

 if(root == NULL)
    {
    root = (node *) malloc( sizeof(node));

    root->data = data;
    root->left = NULL;
   root->right =NULL;
   root->bf = 0;

   *Current =1;
    return root;
    }
```

```
if ( data > root->data)
{
 root->right = insert(root->right, data, &current);
 if (*current == 1)
 {
   switch(root->bf)
   {
Case  -1 : temp1 = root->right;
           if (temp1->bf == -1)
           {
            printf("\n Single Rotation : RR\n");
           root->right = temp1->left;
           temp1->left = root;
           root->bf = 0;
           root = temp1;
            }
          else
            {
             printf ("\n double Rotation RL\n");
             temp2 = temp1->left;
             temp1->left = temp2->right;
             temp2->right = temp1;
              root->right = temp2->left;
              temp2->left = root;

             if( temp2->bf == 1)
                  root->bf = 1;
             else
                  root->bf = 0;
```

```
if (temp2->bf ==1)
    root->bf = -1;
else
    root->bf =0;

if (temp2->bf == -1)
    temp1->bf =1;
else
    temp1->bf = 0;
root = temp2;
}
root->bf=0;
*current=0;
break;
Case 0 : root->bf =1;
         break;

Case -1 : root->bf =0;
          *current =0;

}

}

}
```

```
if ( data > root->data)
{
 root->right = insert(root->right, data, current);
 if (*current == 1)
 {
   switch (root->bf)
   {
Case -1 : temp1 = root->right;
          if (temp1->bf == -1)
          {
           printf("\n Single Rotation : RR\n");
          root->right = temp1->left;
          temp1->left = root;
          root->bf = 0;
          root = temp1;
           }
        else
          {
           printf ("\n double Rotation RL\n");
          temp2 = temp1->left;
          temp1->left = temp2->right;
          temp2->right = temp1;
          root->right = temp2->left;
          temp2->left = root;

          if ( temp2->bf == -1)
                root->bf = 1;
          else
                root->bf = 0;
```

```
if (temp2->bf == -1)
    temp1->bf = -1;
else
    temp1->bf = 0;

root = temp2;

}
root->bf = 0;
*Current = 0;
break;

Case 0 : root->bf = -1;
         break;

Case 1 : root->bf = 0;
         *Current = 0;

}
}
}
return root;
}

void display (node *temp)
{
    if (temp != NULL)
    {
        display (temp->left);
        Printf ("\n %d", temp->data);
        display (temp->right);
    }
}
```

```
Void main()
{
 int current=1; int i,x;
 root = NULL;

 for(i=1; i<=15; i++)
 {
  Printf("Enter data to insert");
 Scanf("%d", &x);
  root = insert(root, x, &current);
 }

 display(root);

 }.
```

AVL Tree Code Tracing

Insert the following elements into the AVL Tree & Display in Inorder.

40, 50, 30, 60, 70, 45.

(i) insert 40

```
main()
{
  root = NULL;
  c = 1;
  root = insert(NULL, 40, 5555); ⟶ goto (a)
                                    ↑ (b)
  // root becomes 2000 after the call
}
```

c | 1 | (5555 (Address))

(a) ⟶ insert(NULL, 40, 5555)

root = NULL

⇒ root = 2000 (create a new node
Let the address allocated is 2000)

2000 → data = 10
2000 → left = NULL
2000 → right = NULL.
2000 → BF = 0.

N	40	0	N
2000			

c = 1
return 2000 ⟶ (b)

(ii) insert 50

```
main()
{
2000 = insert (2000, 50, 5555);  → (a)
}
```

(a) → insert (2000, 50, 5555)

(50 > 40) data > root → data

root → right = insert (root → right, data, c)

2000 → right = insert (NULL, 50, 5555) → (b)

(c) →

2000 → right = 3000

2000 → bf = 0 ⇒ Case 0 ∴ 2000 → bf = -1
break.

return 2000 → (d)

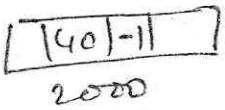
(b) → insert (NULL, 50, 5555)

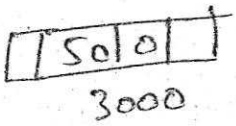
root = NULL

Create a newnode with addr '3000'

store data as 50, bf = 0, c = 1

return 3000 → (c)





```
main()
{
2000  insert (2000, 30, 5555);  ---> (a)
}                                    (d)
```

(a) → insert (2000, 30, 5555)

30 < 40 data < root → data

2000 → left = insert (2000 → left, 30, 5555) ——→ (b)

←—— (c)

2000 → left = 4000

c = 1

2000 → bf = -1, ⇒ 2000 → bf = 0.

return 2000 ——→ (d)

(b) → Insert (NULL, 30, 5555)

root = NULL

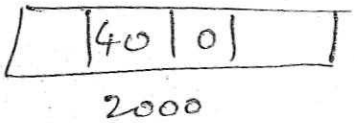
new node is created with addr 4000

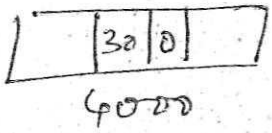
30 stored as data

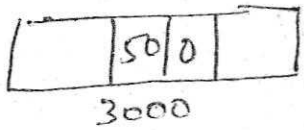
bf = 0

c = 1

return 4000 ——→ (c)







(iv) insert 60

```
main()
{
2000 = insert(2000, 60, 5555);   ——→ (a)
                                 ↑——— (f)
}
```

(a) ⟶ insert(2000, 60, 5555)

(60 > 40) data > root → data

2000 → right = insert(3000, 60, 5555) ⟶ (b)

⟵ (e)

2000 → right = 3000

c = 1, 2000 → bf = 0

Case 0 ⇒ 2000 → bf = -1,

return 2000 ⟶ (f)

(b) ⟶ insert(3000, 60, 5555)

(60 > 50) data > root → data

3000 → right = insert(3000 → right, 60, 5555) ⟶ (c)

⟵ (d)

3000 → right = 5000

c = 1, 3000 → bf = 0.

Case 0 ⇒ 3000 → bf = -1

return 3000 ⟶ (e)

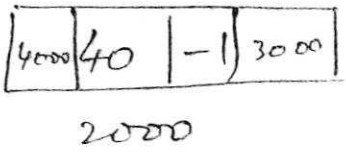
(c) insert (NULL, 60, 5555)

root = NULL

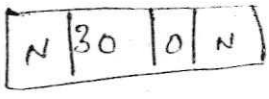
Creates a newnode with addr 5000

Put data as 60, bf = 0, c = 1

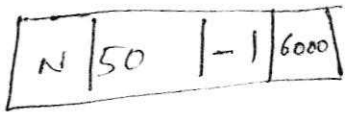
return 5000 ⟶ to (d)



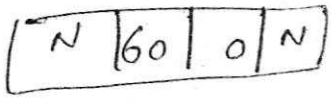
2000



4000



3000



6000

(8). insert 70

```
main()
{
2000 = insert(2000, 70, 5555) ---> (a)
                                   <--- (h)
}
```

(a) → insert(2000, 70, 5555)

data > root → data

2000 → right = insert(3000, 70, 5555) → (b)

← (g)

2000 → right = 5000

c = 0, return 2000 → (h)

(b) → insert(3000, 70, 5555)

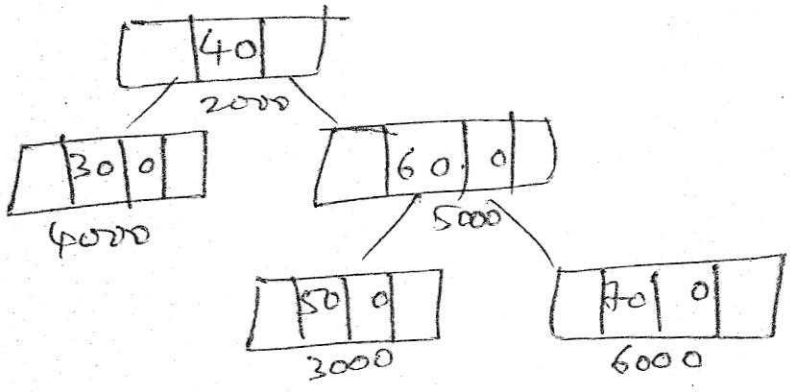
data > root → data.

3000 → right = insert(5000, 70, 5555) → (c)

← (f)

3000 → right = 5000, c = 1, 3000 → bf = -1

Case -1 ⇒ Right Rotation.



3000 → bf = 0

5000 → bf = 0 & root = 5000

return 5000 → (g).

(c) → insert (5000, 70, 5555)

data > root → data (70 > 60)

5000 → right = insert (NULL, 70, 5555) → (d)

← (e)

5000 → right = 6000

c = 1

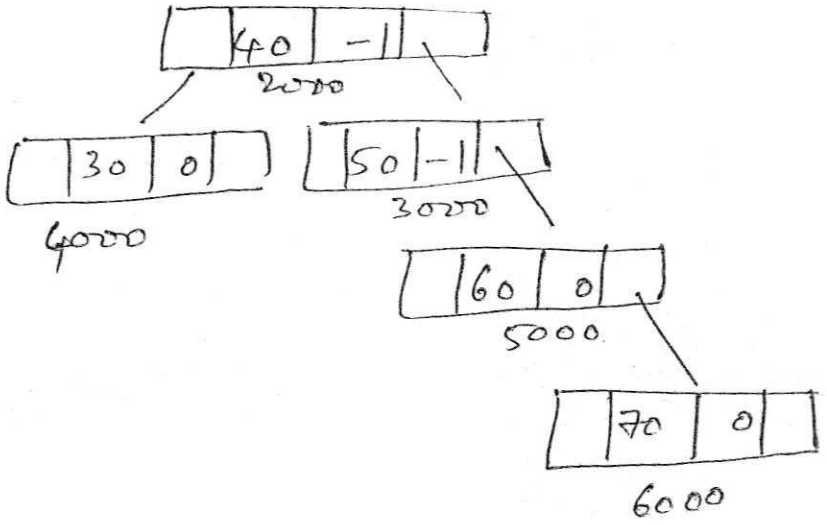
5000 → bf = 0

Case 0 ⇒ 5000 → bf = -1

return 5000 → (f)

(d) → insert (NULL, 70, 5555)

root = NULL, Create new node with address 6000 data as 70, bf = 0, c = 1.



return 6000 → (e)

(ii) Insert 45

```
main()
{
 2000 = insert(2000, 45, 5555);  ---> (a)
}                                <--- (h)
```

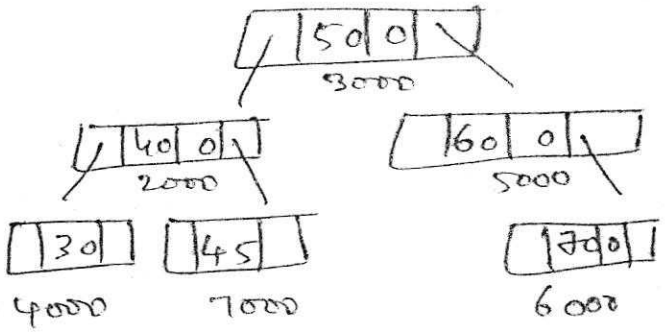
(a) insert (2000, 45, 5555)

(45 > 40) data > root → data

2000 → right = insert(5000, 45, 5555) → (b)

← (g)

c=1, 2000 → bf = -1 ⇒ Case -1 (RL Rotation)



3000 → bf = 1 ⇒ 2000 → bf = 0, 5000 → bf = -1

root = temp2 = 3000, ⇒ 3000 → bf = 0.

c = 0, return 3000 → (h)

(b) → insert (5000, 45, 5555)

45 < 60

5000 → left = insert (3000, 45, 5555) → (c)

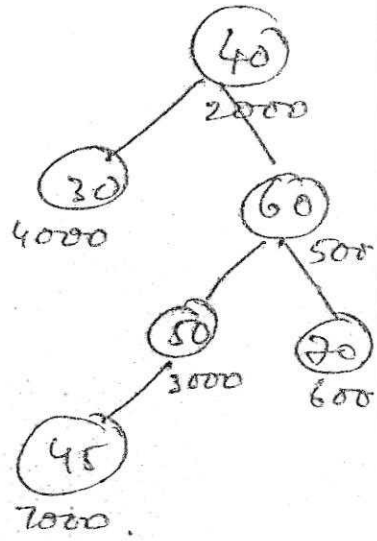
← (f)

5000 → left = 3000

c = 1, 5000 → bf = 0

⇒ 5000 → bf = 1

return 5000 → (g)

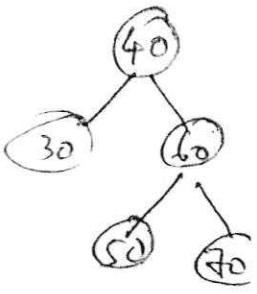


(c) → insert (3000, 45, 5555)

45 < 50

3000 → left = insert (3000 → left, 45, 5555) ——→ (d)

← (e)



3000 → left = 7000

c = 1

3000 → bf = 0 ⇒ 3000 → bf = 1

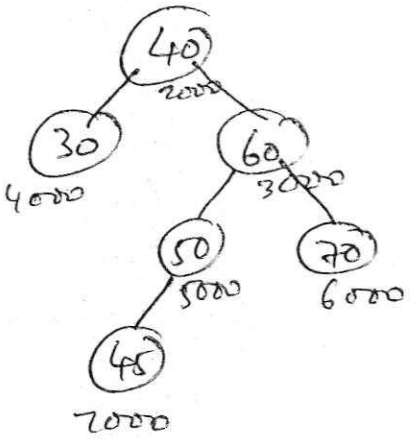
return 3000 → (f)

(d) → insert (NULL, 45, 5555)

root = NULL

Create a new node & let the addr is 7000.

data as 45, bf = 0, c = 1



return 7000 → (e)

A B-tree of order m, is an m-way search tree with the following properties

(i) Root must have atleast two children

(ii) All the leaf nodes must be on the bottom level.

(iii) All the ~~leaf~~ nod internal nodes except leaf nodes must have atleast ceil $(m/2)$ non empty Children.

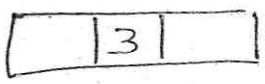
(iv) if the node has 'n' children, then it must have n-1 keys.

Task 1 :-

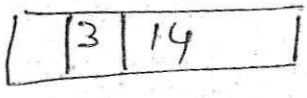
Insert the following values into the B-tree
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19.

Solutions :-

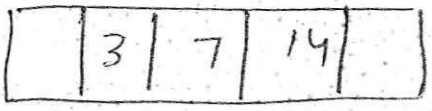
inserting 3



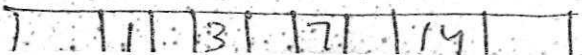
Inserting 14



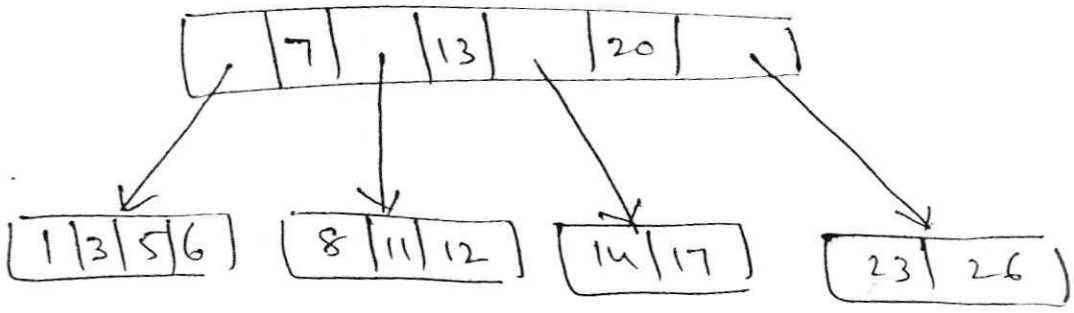
Inserting 7



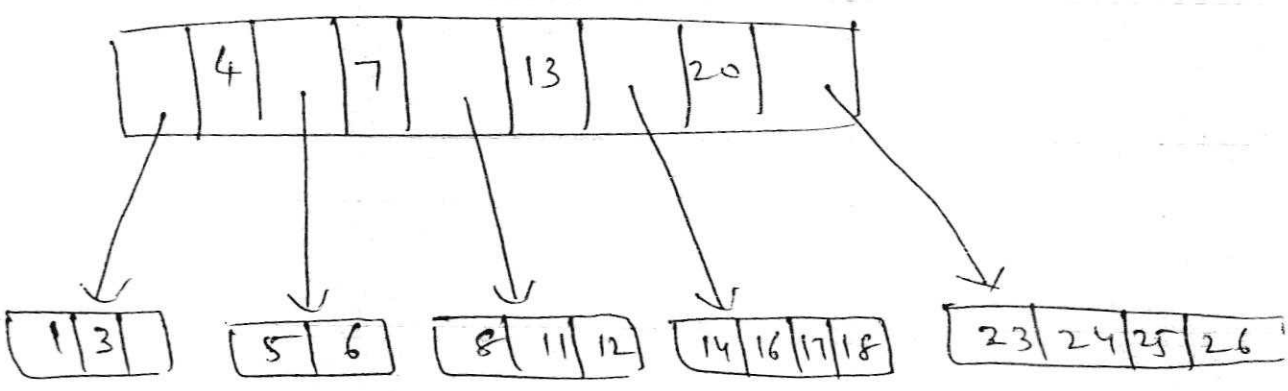
Inserting 1



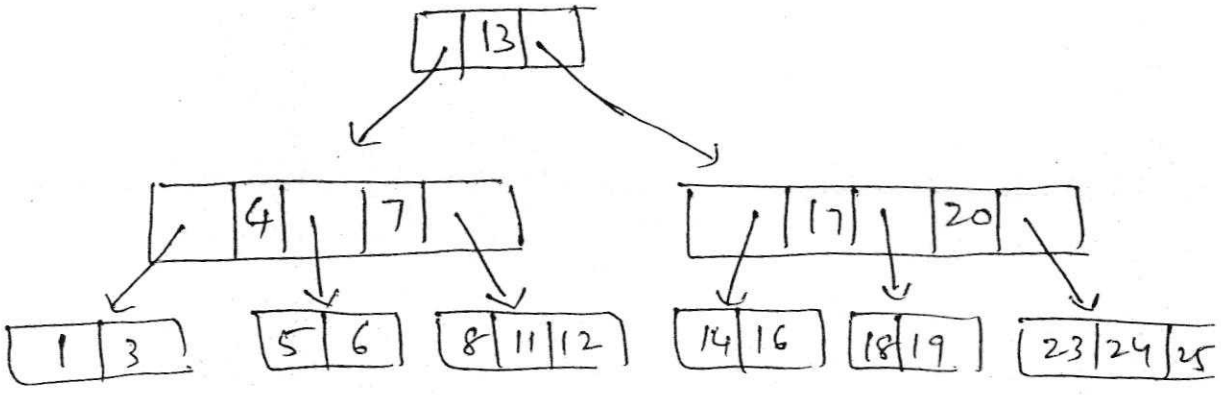
Insert 26



Insert 4



Insert 19



```
# include <stdio.h>
# include <stdlib.h>

# define  MAX   4
# define  MIN   2

struct  treenode
{
   int  Count ;
   int  Keys [MAX +1] ;
   struct  treenode  * links [MAX +1] ;
} ;

typedef  struct  treenode  node ;

int  search (int, node *, int *) ;
void  insertin (int, node *, node *, int) ;
node *  insert (int, node *) ;
int  movedown (int, node *, int *, node **) ;
void  split (int, node *, node *, int,
             int *, node **) ;
void  inorder (node *) ;
```

```
int search (int key, node *current, int *pos)
{
  if (key < current -> keys[1])
    {
      *pos = 0;
      return 0;
    }
  else {
  for (*pos = current -> count;
       key < current -> keys[*pos] && *pos > 1;
            (*pos)--);

  if (key == current -> keys[*pos])
       return 1;
  else
       return 0;
      }
  }
```

```
void insertin(int med, node *medright, node *curre
                                    int pos)
{
  int i;
  for( i = current -> count ; i > pos ; i--)
  {
current -> keys[i+1] = current -> keys[i];
current -> links[i+1] = current -> links[i];
  }
current -> keys[pos+1] = med;
current -> links[pos+1] = medright;

current -> count++;

}
```

```
node * insert (int x, node *temp)
{
  int medentry; node *medright, *newnode;
  if (movedown (x, temp, &medentry, &medright))
  {
  newnode = (node *) malloc (sizeof (struct treenode));
  newnode → cant = 1;
  newnode → keys [1] = medentry;
  newnode → links [0] = temp;
  newnode → links [1] = medright;
  return newnode;
   }
  return temp;
  }
```

```
int movedown (int x, node *current, int *med,
                                node **medright)
{
  int pos;
  if (current == NULL)
    {
      *med = x;
      *medright = NULL;
      return 1;
    }
  else
    {
      if (search (x, current, &pos))
        printf ("duplicate key");
  if (movedown(x, current→links[pos], med, medright))
  {
 if (current→ count < MAX)
   {
     insertin (*med, *medright, current, pos);
     return 0;
   }
 else {
     split (*med, *medright, current, pos, med, medright);
     return 1;
     } } return 0; } }
```

```
void split(int med, node *medright, node *current,
           int pos, int *newmedian, node **newright)
{
  int i;
  int median;

  if(pos <= MIN)
      median = MIN;
  else
      median = MIN+1;

*newright = (node*) malloc(sizeof(struct treenode));

for(i = median+1; i<= MAX ; i++)
{
(*newright) -> keys[i-median] = current -> keys[i];
(*newright) -> links[i-median] = current -> links[i];
  }

(*newright) -> count = MAX - median;
current -> count = median;

if(pos <= MIN)
    insertin(med, medright, current, pos);
else
    insertin(med, medright, *newright, pos-median);

*newmedian = current -> keys[current -> count];
(*newright) -> links[0] = current -> links[current -> count];
current -> count --;
  }
```

```
void inorder (node *temp)
{
  int pos;
  if (temp)
  {
    inorder (temp->links[0]);
    for( pos=1 ; pos <= temp->count ; pos++ )
    {
     Pf( "%d", temp->keys [ pos] );
     inorder(temp->links [ pos] );
    }
  }
}

Void  main( )
{
 node *root; int i,x ;
 root = NULL;

for(i=1; i<= 20 ; i++)
{
Pf("Enter the data to insert ");
Sf("%d", & x);
root = insert( x, root);
 }
inorder( root);
 }
```

(i) insert :

```
main()
{   node *root = NULL;
root = insert(3, NULL)  ---> (a)
}                            <--- (d)
```

(a) ⟶ insert(3, NULL)

```
if (movedown(3, NULL, &me, &mr))  ---> (b)
                                  <--- (c)
```

Creates new node, let the address = 2000

newnode → ~~entry~~ Keys[1] = me = 3

newnode → link[0] = root = NULL

newnode → link[1] = mr = NULL

return 2000 ⟶ (d)

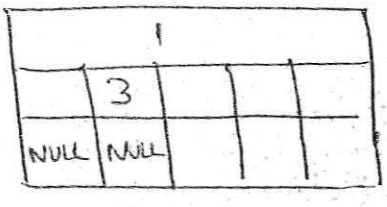
(b) → movedown(3, NULL, &me, &mr)

CUR = NULL

me = x

mr = NULL

return 1 ⟶ (c)



main()
{
root = insert (14, 2000) ——→ (a)
↑ ——→ (h)
}

(a) → insert (14, 2000)

if (movedown(14, 2000, &me, &mr)) ——→ (b)
← (g)
return 2000 ——→ (h)

(b) → movedown(14, 2000, &me, &mr)

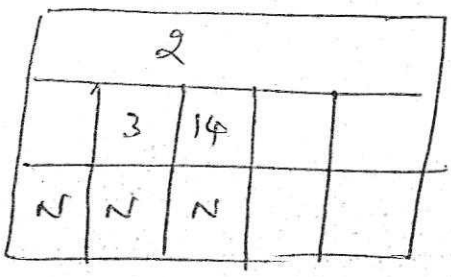
CUR != NULL
Search ~~node~~ (14, 2000, &pos) ⇒ pos = 1
if (movedown (14, 2000 → link[1], me, mr) ; ——→ (c)
← (d)
2000 → Count < MAX then
insertIn(14, NULL, 2000, 1) ——→ (e)
← (f)
return 0 ——→ (g)

(c) → movedown (14, NULL, me, mr)

CUR == NULL me = 14 mr = NULL return 1 ——→ (d)

(e) → insert in (14, NULL, 2000, 1)

place 14 as 2nd key & link[2] = NULL, Count++



return ; ——→ (f)

main()
{
root = insert(1, 2000) ——→ (a)
}
←——— (k)

insert((a) → insert(7, 2000)

if (movedown(7, 2000, &me, &mr)) ——→ (b)
← (g)

return 2000 ——→ (h)

(b) → movedown(7, 2000, &me, &mr)

CUR != NULL search(7, 2000, &pos) ⇒ pos = 1

if (movedown(7, 2000 → link[1], me, mr)) ——→ (c)
← (d)

2000 → Count < MAX Then
insertin(7, N, 2000, 1) ——→ (e)
← (f)
return 0 ——→ (g)

(c) → movedown(7, NULL, me, mr)

CUR = NULL, me = 7, mr = NULL, return 1 ——→ (d)

(e) → insertin(7, NULL, 2000, 1)

move 2000 → link[2], 2000 → Key[2] to Right

place 2000 → Key[2] = 7, 2000 → link[2] = NULL.

3				
	3	7	14	
N	N	N	N	

return ; ——→ (f)

(iv) insert 1

o/p

4				
	1	3	7	14
N	N	N	N	N

8

```
main()
{
  root = insert(8, 2000)
}
```

(a) → insert(8, 2000)

if (movedown(8, 2000, &me, &mr)) → (b)
← (i)

Creates new node, let the address is 4000, Count = 1

4000 → Keys[1] = 7, 4000 → links[0] = 2000,

4000 → links[1] = mr = 3000, return 4000 → (j)

(b) → movedown(8, 2000, &me, &mr)

CUR != NULL, search(8, 2000, &Pos) ⇒ Pos = 3

if (movedown(8, 2000 → link[3], me, mr) → (c)
← (d)

2000 → Count < MAX False split(8, NULL, 2000, 3, me, mr) → (e)
← (h)

return 1 → (i)

(c) → movedown(8, NULL, me, mr)

CUR = NULL, me = 8, mr = NULL, return 1 → (d)

(e) → split(8, NULL, 2000, 3, me, mr)

Pos <= MIN, False ⇒ median = 3

newright = new node (let the addr is 3000)

move entries after median to newright (ie 3000)

newright → Count = 1

2000 → Count = 3

Pos <= MIN False

insertin(8, NULL, 3000, Pos - median) → (f)

(9)

newmedian = (i.e. entry [count → count])

3000 → link[0] 2000 → link[3] = NULL

2000 → count = ; 2000 → count = 2

updations :- me = 7

mr = 3000

return ; ⟶ (4)

insertin(8, NULL, 3000, 0)

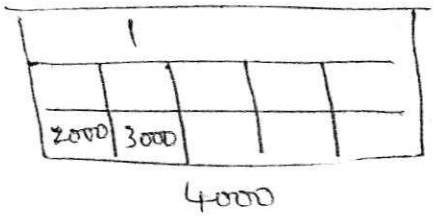
move 3000 → ~~Keys~~[i]

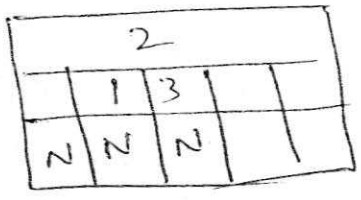
3000 → link[1] to right

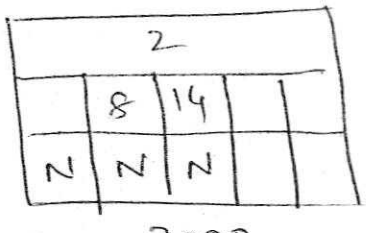
8

store 3000 → keys[i] = 8

3000 → link[i] = NULL.

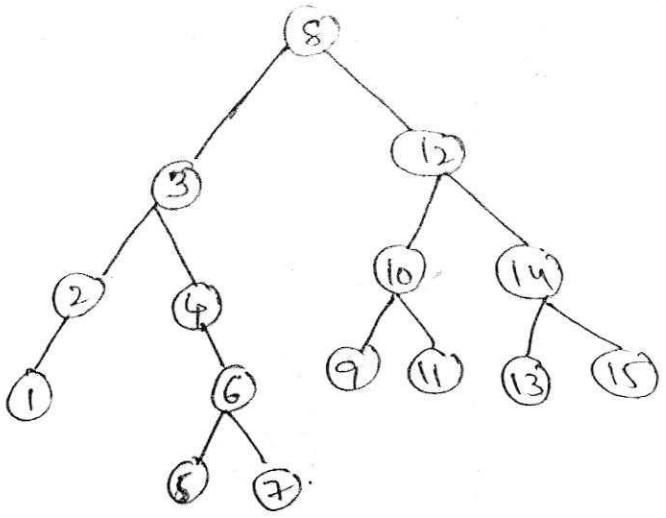




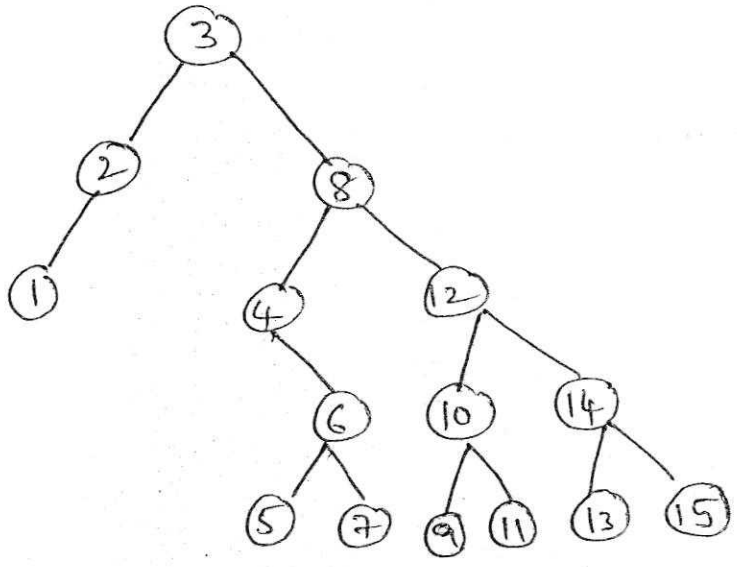


return ; ⟶ (9).

Node 3 (Parent) grand parent involved in splay operation indicate zig-zag (LR rotation)



To bring '3' to Root, we need zig (L Rotation)



Power Point Presentations (PPT) /Videos

Kmit

Data Structures through C

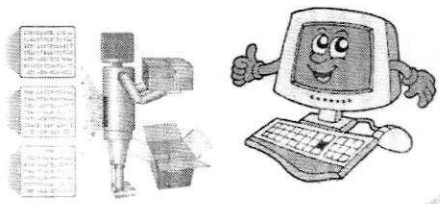
Data Structure Through C

Kmit

Data Structures through C

Data Structures:

In computer science, a data structure is a particular way of storing and organizing data in a computer's memory so that it can be used efficiently.



Kmit

Data Structures through C

Types:-

A data structure can be broadly classified into

Primitive data structure

Non-primitive data structure

Kmit

Data Structures through C

(i) Primitive data structure

The data structures, typically those data structure that are directly operated upon by machine level instructions i.e. the fundamental data types such as int, float, double etc. In case of 'c' are known as primitive data structures.

(ii) Non-primitive data structure

We can define that Non-primitive Data structure is a kind of representation of logical relationship between related data elements. In data structure, decision on the operations such as storage, retrieval and access must be carried out between the logically related data elements.

Kmit

Data Structures through C

For example, we have data player's name "Virat" and age 26. Here "Virat" is of **String** data type and 26 is of **integer** data type.

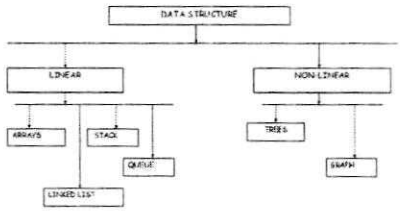
We can organize this data as a record like **Player** record. Now we can collect and store player's records in a file or database as a data structure. For example: "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

In simple language, Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily.

Kmit

Data Structures through C

Types of Data Structures



Kmit

Limitation of Arrays

Data Structures through C

1) Fixed in size :
Once an array is created, the size of array cannot be increased or decreased.
Eg: int a[100];

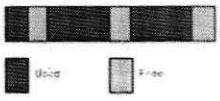
2) Wastage of space :
If no. of elements are less, leads to wastage of space.

int a[100]={1,2,3,4,5};

Kmit

Data Structures through C

3) Sequential Storage :
Array elements are stored in contiguous memory locations. At the times it might so happen that enough contiguous locations might not be available. Even though the total space requirement of an array can be met through a combination of non-contiguous blocks of memory, we would still not be allowed to create the array.

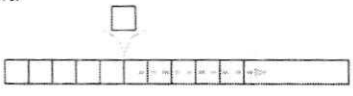


Kmit

Data Structures through C

4) Difficulty in insertion and deletion :

In case of insertion of a new element, each element after the specified location has to be shifted one position to the right. In case of deletion of an element, each element after the specified location has to be shifted one position to the left.



Kmit

Data Structures through C

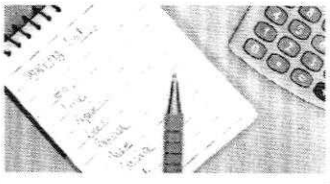
LINKED LIST

Kmit

Data Structures through C

The everyday usage of the term "list" refers to a linear collection of data items.

Example a shopping list, it contains a first element, a second element,..., and a last element.



Kmit

Data Structures through C

Linked List..

Linked List is a very common data structure often used to store similar data in memory. While the elements of an array occupy contiguous memory locations, those of a linked list are not constrained to be stored in adjacent locations. The individual elements are stored "somewhere" in memory. The order of the elements is maintained by explicit links between them.

Kmit — Data Structures through C

ADVANTAGES:

- Efficient memory utilization: In linked list (or dynamic) representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is deallocated (or removed) when it is not needed.
- Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
- Many complex applications can be easily carried out with linked list.

Kmit — Data Structures through C

TYPES OF LINKED LIST

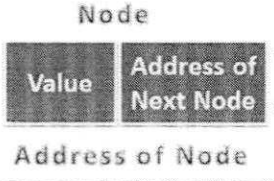
Basically we can divide the linked list into the following four types in the order in which they (or node) are arranged.

- Singly linked list
- Circular linked list
- Doubly linked list
- Circular Doubly linked list

Kmit — Data Structures through C

Single Link List:
A Single linked list or one way list is a linear collection of data elements, called ***Nodes***.

Each node is divided into two parts.

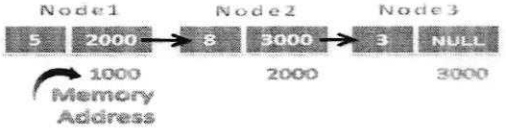


The first pa … ient.

The second … s the address of the next no…

Kmit — Data Structures through C

To see this more clearly let's look at an example:



The left part of the each node represents the information part of the node, which may contain an entire record of data (e.g. ID, name, marks, age etc). The right part represents pointer/link to the next node. The next pointer of the last node is ***null*** pointer signal the end of the list.

Kmit — Data Structures through C

OPERATION ON SINGLE LINKED LIST
Creating a Linked list:

Linked list are created dynamically. It means memory is allocated whenever it is required. And it is deallocated (or removed) when it is not needed.

Dynamic allocation is a pretty unique feature to C (amongst high level languages). It enables us to create data types and structures of any size and length to suit our programs need within the program.

Kmit — Data Structures through C

Before creating a linked list let's have a look to some pre-define functions available in 'C' to create memory dynamically.

Malloc:

The Function malloc is most commonly used to attempt to ``grab'' a continuous portion of memory.

It is defined by:

void *malloc(size_t number_of_bytes)

Kmit

Data Structures through C

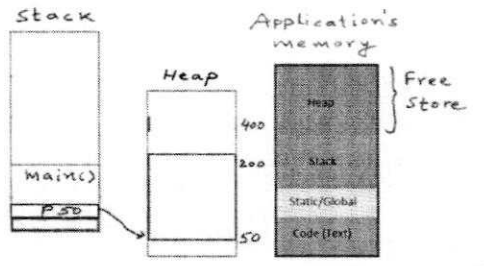
void *malloc(size_t number_of_bytes)

That is to say it returns a pointer of type void * that is the start in memory of the reserved portion of size number_of_bytes. If memory cannot be allocated a NULL pointer is returned. Since a void * is returned the C standard states that this pointer can be converted to any type. The size_t argument type is defined in stdlib.h and is an ***unsigned type***.

Kmit

Data Structures through C

e.g. int *p;
p = (int *) malloc(50*sizeof(int));



Kmit

Data Structures through C

C compilers may require casting the type of conversion. The (int *) means coercion to an integer pointer. Coercion to the correct pointer type is very important to ensure pointer arithmetic is performed correctly. It is good practice to use sizeof() even if you know the actual size you want -- it makes for device independent (portable) code. sizeof can be used to find the size of any data type, variable or structure.

Kmit

Data Structures through C

Free

When you have finished using a portion of memory you should always free() it. This allows the memory ***freed*** to be available again.
The function free() takes a pointer as an argument and frees the memory to which the pointer refers.

e.g. free(p);

Kmit

Data Structures through C

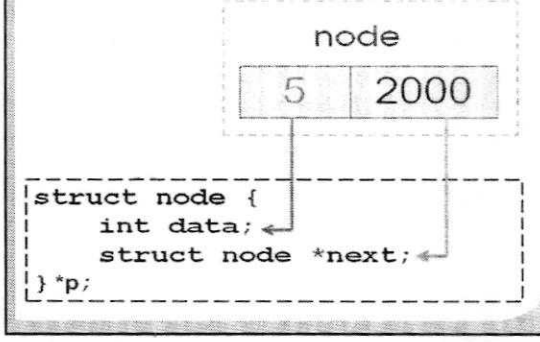
Let us now return to our linked list example:

Linked list is a collection of linked nodes. A node is a structure with at least a data field and a reference to a node of the same type.

A node is called a self-referential object, since it contains a pointer to a variable that refers to a variable of the same type.

Kmit

Data Structures through C



Kmit

Data Structures through C

void *malloc(size_t number_of_bytes)

That is to say it returns a pointer of type void * that is the start in memory of the reserved portion of size number_of_bytes. If memory cannot be allocated a NULL pointer is returned. Since a void * is returned the C standard states that this pointer can be converted to any type. The size_t argument type is defined in stdlib.h and is an ***unsigned type***.

Kmit

Data Structures through C

We can now try to grow the list dynamically:

new_node = (struct Node *) malloc(sizeof(struct Node));

This will allocate memory for a new link.

Kmit

Data Structures through C

OPERATIONS ON LINKED LISTS

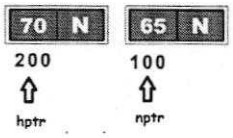
- ***Creation*** operation is used to create a linked list.
- ***Insertion*** operation is used to insert a new node at any specified location in the linked list. A new node may be inserted.
  - (*a*) At the beginning of the linked list
  - (*b*) At the end of the linked list
  - (*c*) At any specified position in between in a linked list
- ***Deletion*** operation is used to delete a node from the linked list. A node may be deleted from the
  - (*a*) Beginning of a linked list
  - (*b*) End of a linked list
  - (*c*) Specified location of the linked list
- ***Traversing*** is the process of going through all the nodes from one end to another end of a linked list

Kmit

hptr=NULL;
create(hptr,70);

Data Structures through C

```
struct node *create(struct
node *hptr,int x)
{
struct node *nptr,*ptr;
nptr=(struct
node*)malloc(sizeof(struct
node));
nptr->data=x;
nptr->next=NULL;
if(hptr==NULL)
{
hptr=nptr }
```



create(200,65);

Kmit

Data Structures through C

Stack

A **Stack ADT** is one of the most important and useful non-primitive linear data structure in computer science. It is an ordered collection of items into which new data items may be added/inserted and from which items may be deleted at only one end, called the *top* of the stack. As all the addition and deletion in a stack is done from the top of the stack, the last added element will be first removed from the stack. That is why the stack is also called ***Last-in-First-out (LIFO)***.

Kmit

Data Structures through C

There are many real life examples of stack. Consider the simple example of plates stacked over one another in canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order

Kmit

Data Structures through C

Examples of Stacks:

Stack of books

Stack of Coins

Kmit

Data Structures through C

In Computers function calls uses mechanism of pushing and popping information concerning the local variables on a stack.
When you make a function call, it is like pushing the current state (arguments and Return address) on the stack and when we return to the calling routine, it is like popping.

Kmit

Data Structures through C

When you make a function call, it is like inserting the current state (arguments and Return address) on the stack.

Add(4,8)

```
int add(int a , int b )
{
    int c;
    c = a + b;
    return c;
}
```

Stack

Kmit

Data Structures through C

OPERATIONS PERFORMED ON STACK
The primitive operations performed on the stack are as follows:

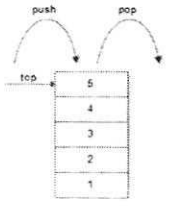
PUSH: The process of adding (or inserting) a new element to the top of the stack is called PUSH operation. Pushing an element to a stack will add the new element at the top. After every push operation the top is incremented by one. If the array is full and no new element can be accommodated, then the stack overflow condition occurs.

Kmit

Data Structures through C

POP: The process of deleting (or removing) an element from the top of stack is called POP operation. After every pop operation the stack is decremented by one. If there is no element in the stack and the pop operation is performed then the stack underflow condition occurs.

PEEK: Get the topmost item.



Kmit

Data Structures through C

STACK IMPLEMENTATION

Stack can be implemented in two ways:

1. Static implementation (using arrays)
2. Dynamic implementation (using pointers)

Kmit

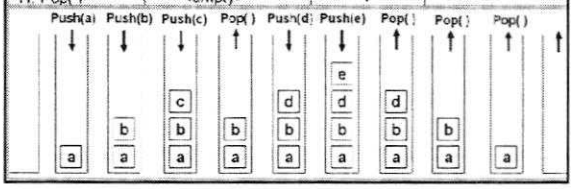
Data Structures through C

STACK USING ARRAYS

Implementation of stack using arrays is a very simple technique. The idea is that the array contains the element of the sequence stored in the stack, and a single index has to be maintained in order to indicate where is currently the top of the stack. It is this position where the element will be added to or deleted from.
This is briefly illustrated below

Operations on Stack

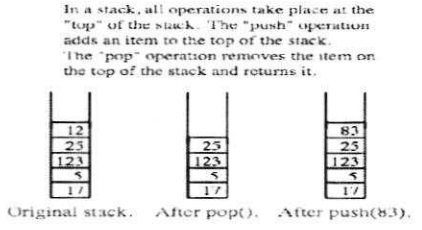
	Stack's contents	TOP value	Output
1. Init_stack( )	<empty>	-1	
2. Push('a')	a	0	
3. Push('b')	a b	1	
4. Push('c')	a b c	2	
5. Pop( )	a b	1	c
6. Push('d')	a b d	2	c
7. Push('e')	a b d e	3	c
8. Pop( )	a b d	2	c e
9. Pop( )	a b	1	c e d
10. Pop( )	a	0	c e d b
11. Pop( )	<empty>	-1	c e d b a



Kmit

Data Structures through C

This is briefly illustrated below.



Kmit

Data Structures through C

Observe that in the structure representing a stack we have one field, called top which indicates the top element in the stack. For simplicity, we assume that actually top indicates what is the position of the array which comes after the top element—this allows us to use top directly for inserting new elements and allows us to avoid designing special values for top in the case of an empty stack.

Kmit

Data Structures through C

OPERATIONS PERFORMED ON STACK

The primitive operations performed on the stack are as follows:

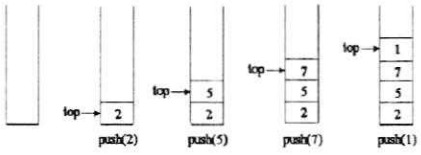
PUSH Operation:

The process of adding (or inserting) a new element to the top of the stack is called PUSH operation. Pushing an element to a stack will add the new element at the top. After every push operation the top is incremented by one. If the array is full and no new element can be accommodated, then the stack overflow condition occurs.

Kmit

Data Structures through C

Consider the stack given below



Initially the stack is set to empty; Suppose TOP is a pointer to the top element in a stack. After every push operation, the value of TOP is incremented by one. When the stack is full the status of the stack is full and this condition is called stack overflow. In such a case, no further element can be inserted.

Kmit

Data Structures through C

Algorithm for pushing (or add or insert) a new element at the top of the stack is given below.

Push(stack, top, max, item)Inserts an item onto the stack
If top = max-1 then : print : OVERFLOW and **return.**
Set top : = top +1 . [increases top by 1]
Set stack[top] := item [inserts item in **new** TOP position]
END

Kmit

Data Structures through C

Code:

```
void push(int x)
{
if(top==MAX-1)
{
printf("Stack Overflow");
return;
}
top++;
stack[top]=x;}
```

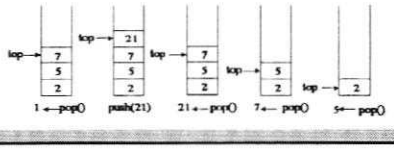
Kmit

Data Structures through C

POP Operation:

The process of deleting (or removing) an element from the top of stack is called POP operation. After every pop operation the stack is decremented by one. If there is no element in the stack and the pop operation is performed then the stack underflow condition occurs.

Consider the stack given below.



Kmit

Data Structures through C

Non-Linear Data Structures

Kmit

Data Structures through C

Non-Linear Data Structures:

So far, we studied the properties of linear (sequential) types of data structures: arrays, lists, stacks and queues that could be used to efficiently represent and manipulate linear arrangements of data.

The Non-Linear data structures, are those that are either explicitly arranged in a hierarchical manner, or result in a hierarchical structure when processed.

Kmit

Data Structures through C

Introducing...

TREES:

A **Tree** is a Non-Linear Data Structure consisting of nodes organized as a hierarchy.

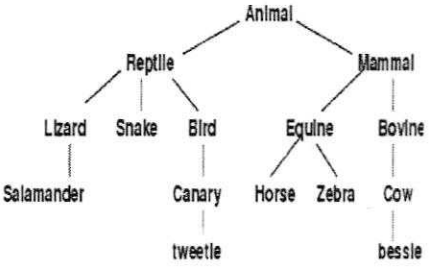
Trees are very flexible, versatile and powerful non-liner data structure that can be used to represent data items possessing hierarchical relationship between nodes/elements.

E.g. Family trees, Species trees, tables of contents, etc.

Kmit

Data Structures through C

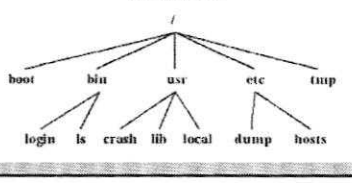
Here is an example of a tree of species, from zoology



Kmit

Data Structures through C

In the context of computer science a typical application of general trees is that for instance of a file hierarchy structure of a unix account, where documents (plain files) are organised into folders (directories), and folders can themselves include documents or other folders. In such a tree documents are modeled by the leaves of the tree and folders are modeled by parent nodes.

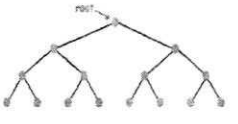


Kmit

Data Structures through C

TREE BASIC TERMINOLOGIES

- **Trees** are sets of points, called **nodes**, and lines, called **edges**
- A **node** is a structure which may contain a value or condition, or represent a separate data structure.
- The top node in the tree is called the **Root** and all other nodes branch off from this node.

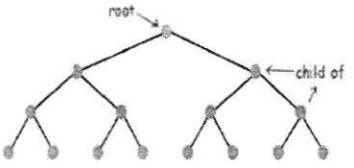


Kmit

Data Structures through C

➢**Root** is a special designed node (or data items) in a tree. It is the first node in the hierarchical arrangement of the data items.

➢Each node in a tree has zero or more **child nodes.**



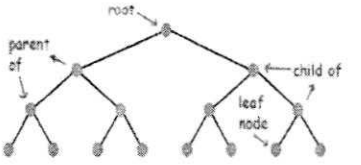
Kmit

Data Structures through C

➢ A node that has a child is called the child's **parent node**

➢There is a unique directed path from one node to another called **link** or an **edge**

➢Node having no children are called **leaf Node** (also known as **External node, or terminal node**).

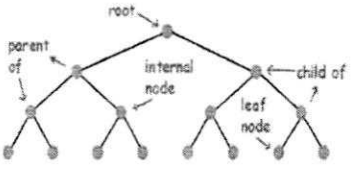


Kmit

Data Structures through C

➢Any node which is neither a root, nor a leaf is called an **interior node.**

➢Nodes with the same parent are called **siblings**



Kmit

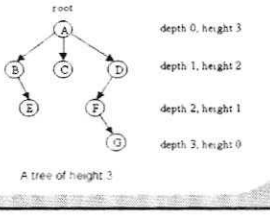
Data Structures through C

The **height** of a node is defined to be the length of the longest path from the node to a leaf in that tree

The **height** of the root is the height of the tree

The **depth** of a node is the length of the path to its root (i.e., its *root path*).

Note: The root node has depth zero; leaf nodes have height zero.
And a tree with only a single node
(hence both a root and leaf) has depth and height zero.



Kmit

Data Structures through C

The number of subtrees of a node is called its **degree.**

In-degree of a node is the number of edges arriving at that node.

Out-degree of a node is the number of edges leaving that node.

The root is the only node in the tree with In-degree = 0.

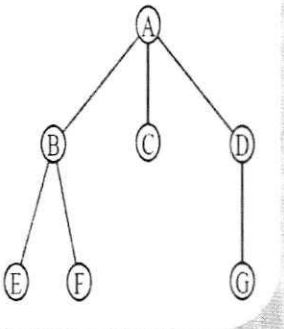
All the leaf nodes have Out-degree = 0.

The ***degree of a tree*** is the maximum degree of a node in the tree.

Kmit

Data Structures through C

Consider the following tree



A is the root node.
- B is the parent of E and F.
- C is the sibling of B & D
- E and F are the children of
- E, F, C, G are external node or leaves.
- A, B, D are internal nodes.
- The depth (level) of C is 1
- The height of the tree is 2.
- The degree of node A is 3.

Kmit

Data Structures through C

Binary Tree:

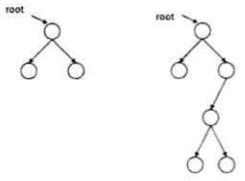
Definition:

A ***Binary Tree*** consists of a finite set of nodes,
that is either empty,
or
consists of one specially designated node called the *root* of the binary tree, and the elements of two disjoint binary trees called the *left subtree* and *right subtree* of the root.

Kmit

Data Structures through C

A binary tree is a tree in which each node can have maximum two children. Thus each node can have no child, one child or two children.

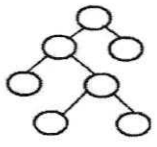


Kmit

Data Structures through C

Types of binary trees:

A **full binary tree** (sometimes proper binary tree or 2-tree or strictly binary tree) is a tree in which every node other than the leaves has two children.

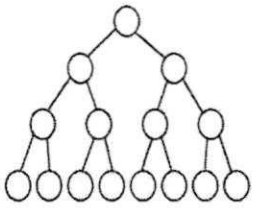


full tree

Kmit

Data Structures through C

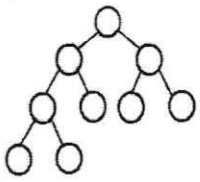
A **perfect binary tree** is a full binary tree in which all leaves are at the same depth or same level, and in which every parent has two children.



Kmit

Data Structures through C

A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

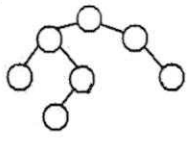


complete tree

Kmit

Data Structures through C

A **balanced binary tree** is commonly defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1



A height-balanced Tree

Kmit

Data Structures through C

Basic Properties

- Every node except the root has exactly one parent.
- A tree with n nodes has n-1 edges (every node except the root has an edge to its parent).
- There is exactly one path from the root to each node.

Kmit

Data Structures through C

•Minimum Number Of Nodes

Minimum number of nodes in a binary tree whose height is h.

At least one node at each level.

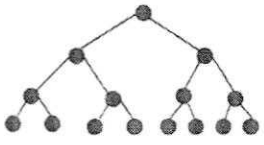
minimum number of nodes is h + 1

Kmit

Data Structures through C

•Maximum Number Of Nodes

All possible nodes at first h levels are present



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^h = 2^{h+1} - 1$

Kmit

Data Structures through C

•Relationship Between Number of Nodes (Internal - External)

Binary tree with N internal nodes has N+1 external nodes

Kmit

Data Structures through C

•Number of edges

A binary tree with N nodes (internal and external) has N-1 edges

Kmit

Data Structures through C

Representation of Binary Trees

Binary trees are implemented in two ways:

➢Array implementation of Binary Trees

➢Linked List implementation of Binary Trees

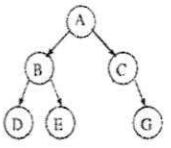
Kmit

Data Structures through C

Array implementation of Binary Trees

In an array implementation of binary trees we use one-dimensional array. The nodes stored in an array are accessible sequentially.

The array representation of the binary tree is shown below.



[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	E	.	G

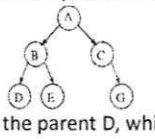
Kmit

Data Structures through C

To perform any operation often we have to identify the parent, the left child and right child of an arbitrary node.

1. The parent of a node having index n can be obtained by $(n - 1)/2$. For example to find the parent of D, where array index $n = 3$. Then the parent nodes index can be obtained

$= (n - 1)/2$
$= 3 - 1/2$
$= 2/2$
$= 1$



[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	E	.	G

i.e., Array[1] is the parent D, which is B.

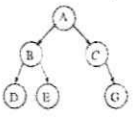
Kmit

Data Structures through C

2. The left child of a node having index *n* can be obtained by (2n+1).

For example to find the left child of C, where array index $n = 2$. Then it can be obtained by

$= (2n + 1)$
$= 2*2 + 1$
$= 4 + 1$
$= 5$



[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	E	.	G

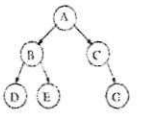
i.e., Array[5] is the left child of C, which is NULL. So no left child for C.

Kmit

Data Structures through C

3. The right child of a node having array index *n* can be obtained by the formula $(2n+ 2)$. For example to find the right child of B, where the array index $n = 1$. Then

$= (2n + 2)$
$= 2*1 + 2$
$= 4$



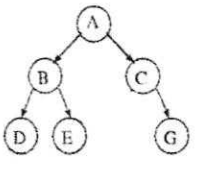
[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	E	.	G

i.e., Array[4] is the right child of B, which is E.

Kmit

Data Structures through C

4. If the left child is at array index *n*, then its right brother is at $(n + 1)$. Similarly, if the right child is at index *n*, then its left brother is at $(n - 1)$.

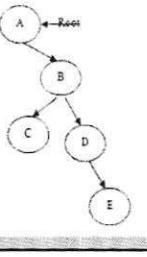


[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	E	.	G

Kmit

Data Structures through C

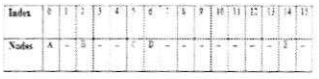
The situation is pleasant so far because it is complete binary tree. But if the binary tree is not complete then it definitely results in unnecessary wastage of memory



Kmit

Data Structures through C

space because array is static in nature. For example let we have a binary tree as shown in figure



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Nodes	A	-	B	-	-	C	D	-	-	-	-	-	-	-	E	-

From this table it is clear that if a tree is far from its complete behavior then the array implementation would waste a lot of memory space. This type of problem is overcome by implementing the binary tree in linked list

Kmit

Data Structures through C

Advantages of linear(Array) representation:

1. Simplicity.
2. Given the location of the child (say, k), the location of the parent is easy to determine (k / 2).

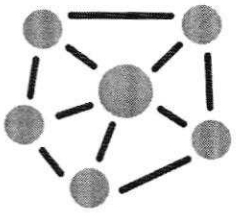
Disadvantages of linear(Array) representation:

1. Additions and deletions of nodes are inefficient, because of the data movements in the array.
2. Space is wasted if the binary tree is not complete. That is, the linear representation is useful if the number of missing nodes is small

Kmit

Data Structures through C

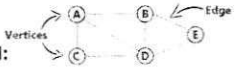
Introduction to Graphs



Kmit

Data Structures through C

INTRODUCTION:



A Graph is a set of nodes (or vertices) and edges (or arcs)which connect them.

G= (V,E)

V: set of vertices

E: set of edges connecting the vertices in V

• An edge e = (u,v) is a pair of vertices

Graphs are mathematically structures and are found very useful in problem solving.

Graphs provide an excellent way to describe the essential features of many applications.

Kmit

Data Structures through C

Example:

A Graph could be viewed as a map of what cities are connected by train routes. Viewing in this way ,each vertex represents a particular city and each edge represents whether there is a train route for one city to another.

Graphs (sometimes referred to as networks) offer a way of expressing relationships between pairs of items.

Kmit

Data Structures through C

Graphs Terminologies

Like trees, ***graphs*** are made up of nodes and the connections between those nodes.

In graph terminology, we refer to the nodes as ***vertices*** and refer to the connections among them as ***edges***

Vertices are typically referenced by a name or label

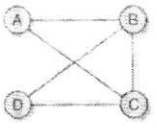
Edges are referenced by a pairing of the vertices (A, B) that they connect

Kmit

Data Structures through C

Graphs Terminologies

➢An ***undirected graph*** is a graph in which all edges are "undirected edges".



Kmit

Data Structures through C

An edge in an undirected graph can be traversed in either direction

Two vertices are said to be ***adjacent*** if there is an edge connecting them

Adjacent vertices are sometimes referred to as ***neighbors***

An edge of a graph that connects a vertex to itself is called a ***self-loop*** or ***sling***

An undirected graph is considered ***complete*** if it has the maximum possible number of edges connecting vertices. How many edges?

Kmit

Data Structures through C

Number of edges:

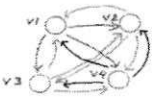
if $|V| = n$

then,

$0 \leq |E| \leq n(n-1)$, if directed

$0 \leq |E| \leq \frac{n(n-1)}{2}$, if undirected

↳ assuming no self-loop or multiedge

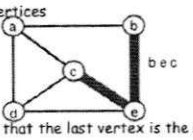


Kmit

Data Structures through C

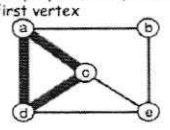
A ***path*** is a sequence of edges that connects two vertices in a graph

- simple path: no repeated vertices

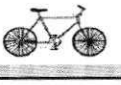


bec

- cycle: simple path, except that the last vertex is the same as the first vertex



acda



Kmit

Data Structures through C

The ***length*** **of a path** in is the number of edges in the path (or the number of vertices minus 1)

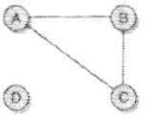
An undirected graph is considered ***connected*** if for <u>any</u> two vertices in the graph there is a path between them

Kmit

Undirected Graphs

Data Structures through C

- An example of an undirected graph that is not connected:

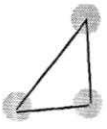


Kmit

Cycles

Data Structures through C

- A ***cycle*** is a path in which the first and last vertices are the same and none of the edges are repeated
- A graph that has no cycles is called ***acyclic***



Cyclic

Acyclic

Kmit

Directed Graphs

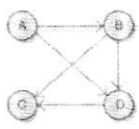
Data Structures through C

- A ***directed graph***, sometimes referred to as a ***digraph***, is a graph where the edges are ordered pairs of vertices (edges have arrows)

Directed Graphs

Kmit — Data Structures through C

- A directed graph with
  - vertices A, B, C, D
  - edges (A, B), (A, D), (B, C), (B, D) and (D, C)



Directed Graphs

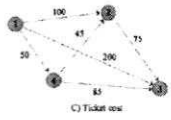
Kmit — Data Structures through C

- Previous definitions change slightly for directed graphs
  - a **path** in a direct graph is a sequence of directed edges that connects two vertices in a graph
  - a directed graph is **connected** if for any two vertices in the graph there is a path between them
- if a directed graph has no cycles, it is possible to arrange the vertices in a sequence such that vertex A precedes vertex B in the sequence, if an edge exists from A to B

Weighted Graphs

Kmit — Data Structures through C

- A ***weighted graph***, sometimes called a ***network***, is a graph with **weights (or costs)** associated with each edge
- The **weight of a path** in a weighted graph is the sum of the weights of the edges in the path
- Weighted graphs may be either undirected or directed



Kmit — Data Structures through C

➢The "Degree" of a node is the number of edges connected directly to that node , i.e., the number of edges incident on it.

➢In a directed graph,the "in degree" of a node is the number of edges beginning from the node .The "out degree" of a node is the number of edges terminating at that node.

Kmit — Data Structures through C

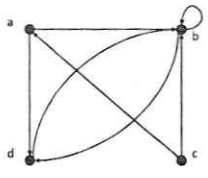
•What are the in-degrees and out-degrees of the vertices a, b, c, d in this graph:

deg⁻(a) = 1
deg⁺(a) = 2

deg⁻(b) = 4
deg⁺(b) = 2

deg⁻(d) = 2
deg⁺(d) = 1

deg⁻(c) = 0
deg⁺(c) = 2



17

Kmit — Data Structures through C

Graph Representation

To represent a graph we have to represent two things:nodes and edges.

Graphs are generally represented either in "sequential representation" or "linked representation".

Sequential representation uses a two-dimentional array where as linked representation uses a linked list.

Note: We refer graph means directed graphs unless specified otherwise

Kmit

Data Structures through C

Sequential Representation:-

A graph is conveniently represented by a matrix (two-dimentional array) called adjacency matrix (or incidence matrix).

A graph containing n nodes can be represented by a matrix containing n rows and n columns.

Adjacency matrix:-

The Adjacency matrix A of a graph G=(V,E) with n nodes is an n*n matrix such that:

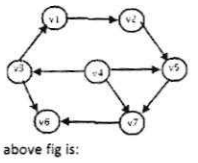
Aij={1 – if there is an edge between vi and vj.

0 – otherwise

}

Kmit

Data Structures through C



Adjacency matrix for the above fig is:

	A	B	C	D
A	0	0	1	0
B	1	0	0	1
C	0	1	0	1
D	1	0	1	0

Each position of adjacency matrix represents whether one node is connected to another (value 1 or true) or not (value 0 or false).Note that it encodes the direction of the edges.For example,since there is a 1 row B,column A,there is an edge from B to A.

Since there is no edge in the reverse direction(from A to B),row A,column B has a 0.

Kmit

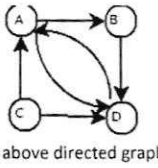
Data Structures through C

Linked Representation:-

In this representation we maintain an adjacency list.In adjacency list,for each node,we keep a list of all adjacent nodes.

Kmit

Data Structures through C



The adjacency list for above directed graph is

A	B D
B	D
C	A B D
D	A

Adjacing list structures using two dimensional array and single linked lists.

Kmit

Data Structures through C

Kmit

Data Structures through C

Graph Traversing

Traversal of a graph means that systematically visiting all nodes exactly once in the graph. The two important traversal methods are Breadth-first Search" and "Depth-first Search". The graph traversal start at an arbitrary vertex since there is no node as special. Assume that each node in a graph will be in one of three states while traversing the graph: ready state, waiting state, visited state.

Kmit

Data Structures through C

Depth-first traversal:-

The depth-first traversal of a graph is much similar to preorder traversal of an ordered tree. The traversal of a graph start at any arbitrary node, say A. Suppose b,C,D and E be the nodes adjacent to A. Then we will next visit b and keep C,D, and E waiting. After visiting B we traverse all the vertices to which it is adjacent before returning to traverse C,D and E.In the depth-first traversal, we backtrack on a path once it reached the end of that path. We consider the data structure stack instead of a queue as in breadth-first traversal.

Kmit

Data Structures through C

Algorithm:-
1) All nodes are initialized to ready state and initialize stack to empty.
2) Begin with any node which is in ready state and push into stack. mark the status of that node to waiting.
3) While stack is not empty
do begin
4) pop the top node k of stack and process it. Mark the status of that node to visited.
5) Push all the adjacent nodes of k which are in ready state into stacj and mark the status of those nodes to waiting.
end.
6) If the graph still contains nodes which are in ready state then goto step2
7) return.

Kmit

Data Structures through C

Breadth-first Traversal:-

This strategy is much similar to level-by-level traversal of an ordered tree. Breadth-first search operates by processing nodes in layers. The breadth-first search can begin at any arbitrary node. The nodes which are adjacent to start node are processed first,and proceeds to adjacent nodes of that nodes just visited. This process starts until all nodes are visited. If the traversal just visited a node A,then it next visits all the nodes adjacent to A,keeping the node adjacent tothese in waiting list to be traversed after all nodes adjacent to A have been visited.

A data structure queue is used to place all waiting nodes.This queue is also convenient to keep the track of nodes that are already visited, so that, a node is visited only once.

Kmit

Data Structures through C

The general Breadth-first traversal algorithm is as follows:

1) All nodes are initialized as ready states and initialize queue to empty.
2) Begin with any node which is in ready state and put into queue.mark the status of that node to waiting.
3) While queue is not empty do begin
4) Delete the first node k from queue and process it.Mark the status of that node to visit.
5) Add all the adjacent nodes of K which are in ready state to the rear side of the queue and mark the status of those nodes to waiting.
6) If the graph still contains nodes which are in ready state then goto step2.
7) return.

Analysis of Algorithms
CS 477/677

Sorting – Part A
Instructor: George Bebis

(Chapter 2)

The Sorting Problem

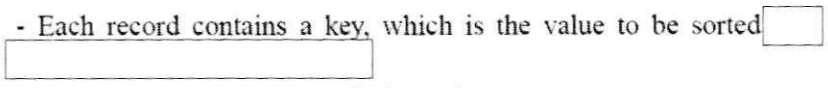
- **Input:**
  - A sequence of n numbers $a_1, a_2, \ldots, a_n$
- **Output:**
  - A permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$

2

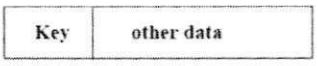
Structure of data

- Usually, the numbers to be sorted are part of a collection of data called a record

- Each record contains a key, which is the value to be sorted



example of a record



- Note that when the keys must be rearranged, the data associated with the keys must also be rearranged (time consuming !!)

- Pointers can be used instead (space consuming !!)

3

Why Study Sorting Algorithms?

- There are a variety of situations that we can encounter
  - Do we have randomly ordered keys?
  - Are all keys distinct?
  - How large is the set of keys to be ordered?
  - Need guaranteed performance?
- Various algorithms are better suited to some of these situations

4

Some Definitions

- Internal Sort
  - The data to be sorted is all stored in the computer's main memory.
- External Sort
  - Some of the data to be sorted might be stored in some external, slower, device.
- In Place Sort
  - The amount of extra space required to sort the data is constant with the input size.

5

Stability

- A STABLE sort preserves relative order of records with equal keys

Sorted on first key:

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Sort file on second key:

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Furia	3	A	766-093-9873	22 Brown
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Gazsi	4	B	665-303-0266	113 Walker
Aaron	4	A	664-480-0023	097 Little

Records with key value 3 are not in order on first key!!

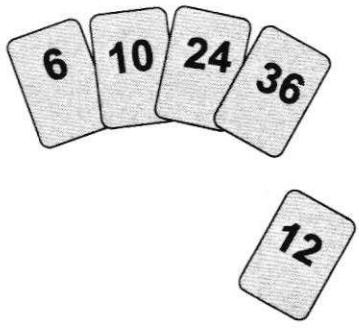
6

Insertion Sort

- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
    - compare it with each of the cards already in the hand, from right to left
  - The cards held in the left hand are sorted
    - these cards were originally the top cards of the pile on the table

7

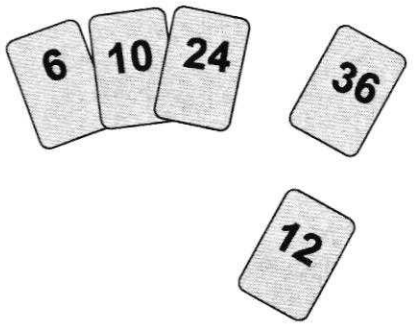
Insertion Sort



To insert 12, we need to make room for it by moving first 36 and then 24.

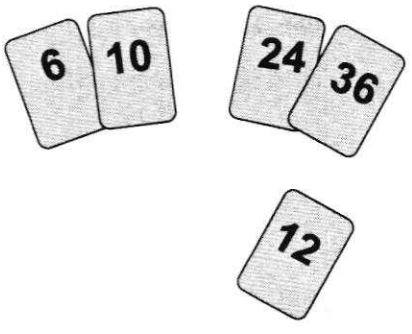
8

Insertion Sort



9

Insertion Sort



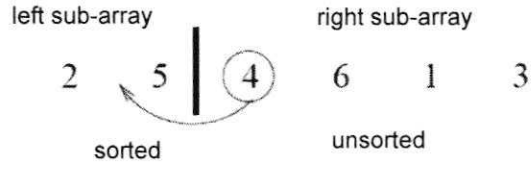
10

Insertion Sort

input array

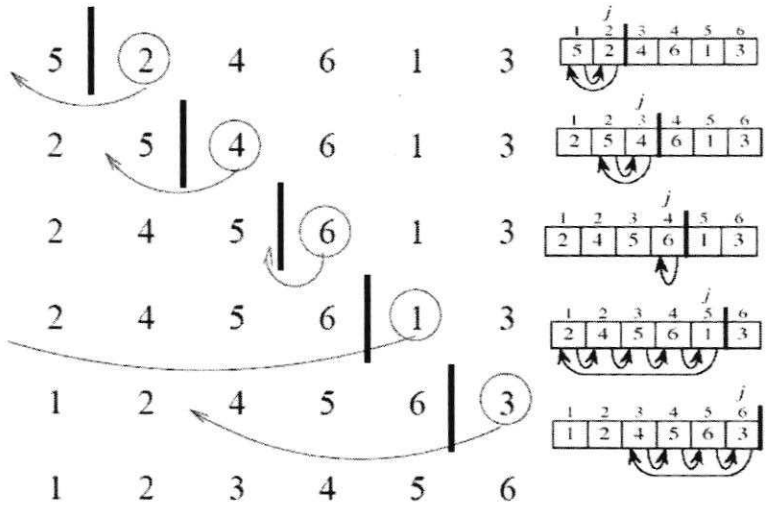
5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:



11

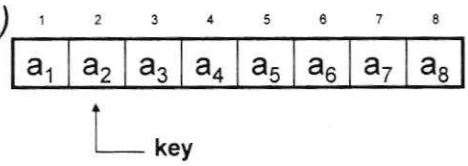
Insertion Sort



12

INSERTION-SORT

Alg.: INSERTION-SORT*(A)*



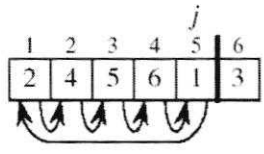

```
for j ← 2 to n
    do key ← A[ j ]
       ▷Insert A[ j ] into the sorted sequence A[1 . . j -1]
       i ← j - 1
       while i > 0 and A[i] > key
          do A[i + 1] ← A[i]
             i ← i - 1
       A[i + 1] ← key
```

- Insertion sort – sorts the elements in place

13

Loop Invariant for Insertion Sort

Alg.: INSERTION-SORT*(A)*




```
for j ← 2 to n
    do key ← A[ j ]
       Insert A[ j ] into the sorted sequence A[1 . . j -1]
       i ← j - 1
       while i > 0 and A[i] > key
          do A[i + 1] ← A[i]
             i ← i - 1
       A[i + 1] ← key
```

Invariant: at the start of the **for** loop the elements in *A*[1 . . j-1] are in sorted order

14

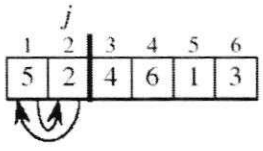
Proving Loop Invariants

- Proving loop invariants works like induction
- **Initialization (base case):**
  - It is true prior to the first iteration of the loop
- **Maintenance (inductive step):**
  - If it is true before an iteration of the loop, it remains true before the next iteration
- **Termination:**
  - When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct
  - Stop the induction when the loop terminates

15

Loop Invariant for Insertion Sort

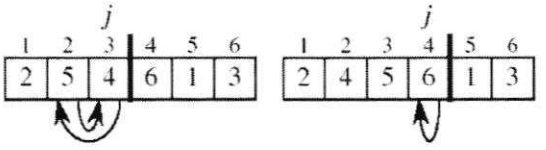
- **Initialization:**
  - Just before the first iteration, j = 2: the subarray $A[1 . . j-1] = A[1]$, (the element originally in $A[1]$) – is sorted



16

Loop Invariant for Insertion Sort

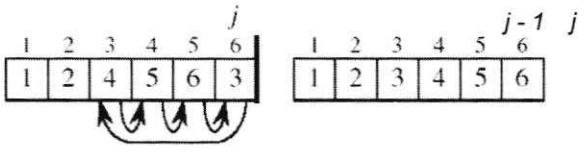
- **Maintenance:**
  - the **while** inner loop moves $A[j-1]$, $A[j-2]$, $A[j-3]$, and so on, by one position to the right until the proper position for key (which has the value that started out in $A[j]$) is found
  - At that point, the value of key is placed into this position.



17

Loop Invariant for Insertion Sort

- **Termination:**
  - The outer **for** loop ends when $j = n + 1 \Rightarrow j-1 = n$
  - Replace n with j-1 in the loop invariant:
    - the subarray $A[1 . . n]$ consists of the elements originally in $A[1 . . n]$, but in sorted order



- The entire array is sorted!

Invariant: at the start of the **for** loop the elements in $A[1 . . j-1]$ are in sorted order

18

Analysis of Insertion Sort

INSERTION-SORT*(A)*	cost	times
for j ← 2 **to** n	c_1	n
do key ← A[j]	c_2	n-1
▷Insert A[j] into the sorted sequence A[1 . . j -1]	0	n-1
i ← j - 1	c_4	n-1
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
do A[i + 1] ← A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i ← i – 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] ← key	c_8	n-1

t_j: # of times the while statement is executed at iteration j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

19

Best Case Analysis

- The array is already sorted **"while** i > 0 and A[i] > key"
  - A[i] ≤ key upon the first time the **while** loop test is run (when $i = j - 1$)
  - $t_j = 1$
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

  $= (c_1 + c_2 + c_4 + c_5 + c_8)n + (c_2 + c_4 + c_5 + c_8)$

  $= an + b = \Theta(n)$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

20

Worst Case Analysis

- The array is in reverse sorted order"**while** i > 0 and A[i] > key'
  - Always A[i] > key in **while** loop test
  - Have to compare key with all elements to the left of the j-th position ⇒ compare with j-1 elements ⇒ $t_j = j$

using $\sum_{j=1}^{n} j = \frac{n(n+1)}{2} \Rightarrow \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \Rightarrow \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$ we have:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8(n-1)$$

$= an^2 + bn + c$ a quadratic function of n

- $T(n) = \Theta(n^2)$ order of growth in n^2

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

21

Comparisons and Exchanges in Insertion Sort

INSERTION-SORT*(A)*	cost	times
for j ← 2 **to** n	c_1	n
do key ← A[j]	c_2	n-1
Insert A[j] into the sorted sequence A[1 . . j -1]	0	n-1
i ← j - 1 ≈ $n^2/2$ comparisons	c_4	n-1
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
do A[i + 1] ← A[i]	c_6	$\sum_{j=2}^{n}(t_j - 1)$
i ← i – 1 ≈ $n^2/2$ exchanges	c_7	$\sum_{j=2}^{n}(t_j - 1)$
A[i + 1] ← key	c_8	n-1

22

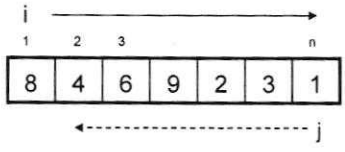
Insertion Sort - Summary

- Advantages
  - Good running time for "almost sorted" arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and exchanges

23

Bubble Sort (Ex. 2-2, page 38)

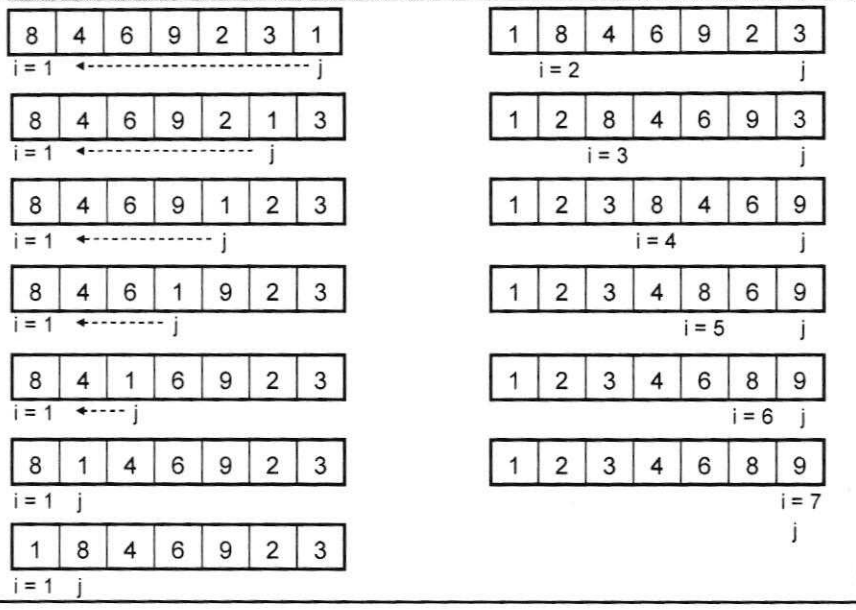
- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

24

Example

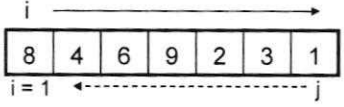


25

Bubble Sort

Alg.: BUBBLESORT(A)

for i ← 1 **to** length[*A*]
  do for j ← length[*A*] **downto** i + 1
    do if *A*[j] < *A*[j -1]
      then exchange *A*[j] ↔ *A*[j-1]



26

Bubble-Sort Running Time

Alg.: BUBBLESORT(A)

for i ← 1 **to** length[A] c_1

do for j ← length[A] **downto** i + 1 c_2

Comparisons: ≈ $n^2/2$ **do if** A[j] < A[j -1] c_3

Exchanges: ≈ $n^2/2$ **then** exchange A[j] ↔ A[j-1] c_4

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(n-i+1) + c_3\sum_{i=1}^{n}(n-i) + c_4\sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_2 + c_4)\sum_{i=1}^{n}(n-i)$$

$$where \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n}n - \sum_{i=1}^{n}i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Thus, $T(n) = \Theta(n^2)$

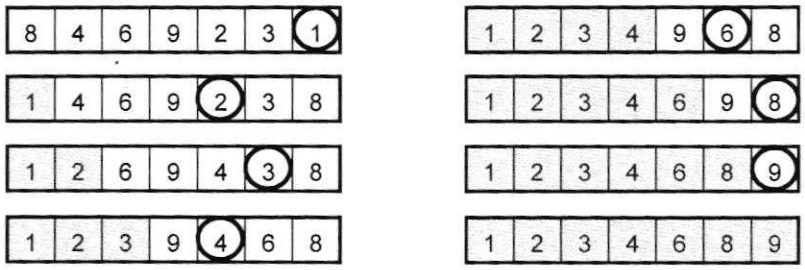
27

Selection Sort (Ex. 2.2-2, page 27)

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted
- Disadvantage:
  - Running time depends only slightly on the amount of order in the file

28

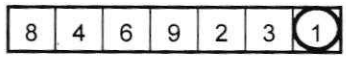
Example



29

Selection Sort

Alg.: SELECTION-SORT*(A)*




```
n ← length[A]
for j ← 1 to n - 1
    do smallest ← j
        for i ← j + 1 to n
            do if A[i] < A[smallest]
                then smallest ← i
        exchange A[j] ↔ A[smallest]
```

30

Analysis of Selection Sort

Alg.: SELECTION-SORT(*A*)	cost	times
n ← length[*A*]	c_1	1
for j ← 1 **to** n - 1	c_2	n
do smallest ← j	c_3	n-1
for i ← j + 1 **to** n	c_4	$\sum_{j=1}^{n-1}(n-j+1)$
do if *A*[i] < *A*[smallest]	c_5	$\sum_{j=1}^{n-1}(n-j)$
then smallest ← i	c_6	$\sum_{j=1}^{n-1}(n-j)$
exchange *A*[j] ↔ *A*[smallest]	c_7	n-1

≈$n^2/2$ comparisons

≈n exchanges

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4\sum_{j=1}^{n-1}(n-j+1) + c_5\sum_{j=1}^{n-1}(n-j) + c_6\sum_{j=2}^{n-1}(n-j) + c_7(n-1) = \Theta(n^2)$$

31

Knuth-Morris-Pratt Algorithm

Prepared by: Mayank Agarwal
Nitesh Maan

The problem of String Matching

Given a string 'S', the problem of string matching deals with finding whether a pattern 'p' occurs in 'S' and if 'p' does occur then returning position in 'S' where 'p' occurs.

.... a O(mn) approach

One of the most obvious approach towards the string matching problem would be to compare the first element of the pattern to be searched 'p', with the first element of the string 'S' in which to locate 'p'. If the first element of 'p' matches the first element of 'S', compare the second element of 'p' with second element of 'S'. If match found proceed likewise until entire 'p' is found. If a mismatch is found at any position, shift 'p' one position to the right and repeat comparison beginning from first element of 'p'.

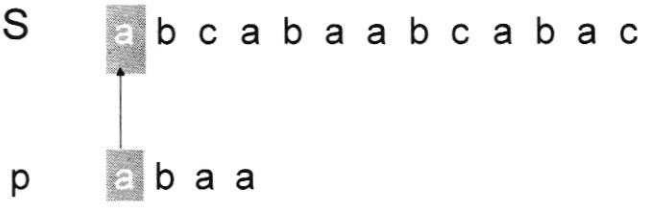
How does the O(mn) approach work

Below is an illustration of how the previously described O(mn) approach works.

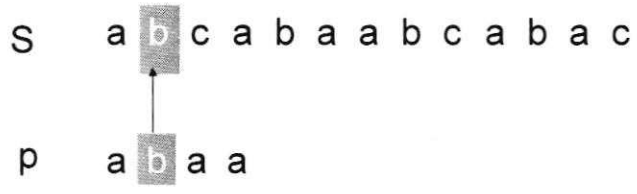
String S a b c a b a a b c a b a c

Pattern p a b a a

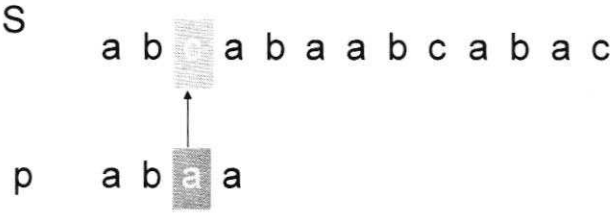
Step 1:compare p[1] with S[1]



Step 2: compare p[2] with S[2]

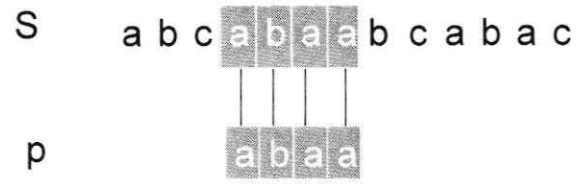


Step 3: compare p[3] with S[3]



Mismatch occurs here..

Since mismatch is detected, shift 'p' one position to the left and perform steps analogous to those from step 1 to step 3. At position where mismatch is detected, shift 'p' one position to the right and repeat matching procedure.



Finally, a match would be found after shifting 'p' three times to the right side.

Drawbacks of this approach: if 'm' is the length of pattern 'p' and 'n' the length of string 'S', the matching time is of the order O(mn). This is a certainly a very slow running algorithm.
What makes this approach so slow is the fact that elements of 'S' with which comparisons had been performed earlier are involved again and again in comparisons in some future iterations. For example: when mismatch is detected for the first time in comparison of p[3] with S[3], pattern 'p' would be moved one position to the right and matching procedure would resume from here. Here the first comparison that would take place would be between p[0]='a' and S[1]='b'. It should be noted here that S[1]='b' had been previously involved in a comparison in step 2. this is a repetitive use of S[1] in another comparison.
It is these repetitive comparisons that lead to the runtime of O(mn).

The Knuth-Morris-Pratt Algorithm

Knuth, Morris and Pratt proposed a linear time algorithm for the string matching problem.

A matching time of O(n) is achieved by avoiding comparisons with elements of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

Components of KMP algorithm

- The prefix function, Π

The prefix function,Π for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid useless shifts of the pattern 'p'. In other words, this enables avoiding backtracking on the string 'S'.

- The KMP Matcher

With string 'S', pattern 'p' and prefix function 'Π' as inputs, finds the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrence is found.

The prefix function, Π

Following pseudocode computes the prefix fucnction, Π:

Compute-Prefix-Function (p)

```
1  m ← length[p]            //'p' pattern to be matched
2  Π[1] ← 0
3  k ← 0
4      for q ← 2 to m
5          do while k > 0 and p[k+1] != p[q]
6              do k ← Π[k]
7             If p[k+1] = p[q]
8               then k ← k +1
9             Π[q] ← k
10     return Π
```

Example: compute Π for the pattern 'p' below:

p a b a b a c a

Initially: m = length[p] = 7
Π[1] = 0
k = 0

Step 1: q = 2, k=0
Π[2] = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
Π	0	0					

Step 2: q = 3, k = 0,
Π[3] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
Π	0	0	1				

Step 3: q = 4, k = 1
Π[4] = 2

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	A
Π	0	0	1	2			

Step 4: q = 5, k =2
Π[5] = 3

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
Π	0	0	1	2	3		

Step 5: q = 6, k = 3
Π[6] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
Π	0	0	1	2	3	1	

Step 6: q = 7, k = 1
Π[7] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
Π	0	0	1	2	3	1	1

After iterating 6 times, the prefix function computation is complete: →

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
Π	0	0	1	2	3	1	1

The KMP Matcher

The KMP Matcher, with pattern 'p', string 'S' and prefix function 'Π' as input, finds a match of p in S. Following pseudocode computes the matching component of KMP algorithm:

```
KMP-Matcher(S,p)
1 n ← length[S]
2 m ← length[p]
3 Π ← Compute-Prefix-Function(p)
4 q ← 0                                   //number of characters matched
5 for i ← 1 to n                          //scan S from left to right
6    do while  q > 0 and p[q+1] != S[i]
7            do  q ← Π[q]                 //next character does not match
8       if p[q+1] = S[i]
9         then q ← q + 1                  //next character matches
10      if q = m                          //is all of p matched?
11        then print "Pattern occurs with shift" i – m
12             q ← Π[ q]                  // look for the next match
```

Note: KMP finds every occurrence of a 'p' in 'S'. That is why KMP does not terminate in step 12, rather it searches remainder of 'S' for any more occurrences of 'p'.

Illustration: given a String 'S' and pattern 'p' as follows:

S b a c b a b a b a b a c a c a

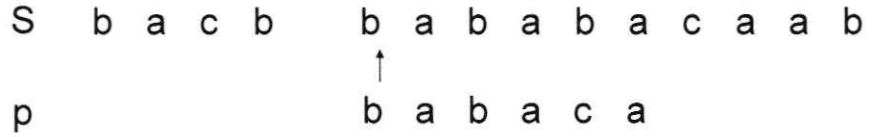
p a b a b a c a

Let us execute the KMP algorithm to find whether 'p' occurs in 'S'.

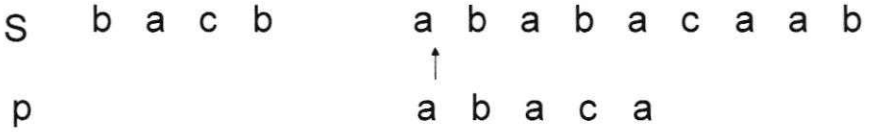
For 'p' the prefix function, Π was computed previously and is as follows:

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
Π	0	0	1	2	3	1	1

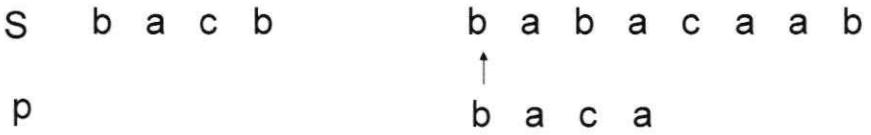
Step 6: i = 6, q = 1
Comparing p[2] with S[6] p[2] matches with S[6]



Step 7: i = 7, q = 2
Comparing p[3] with S[7] p[3] matches with S[7]

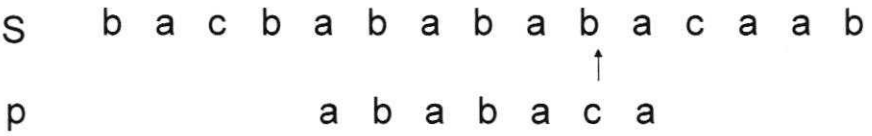


Step 8: i = 8, q = 3
Comparing p[4] with S[8] p[4] matches with S[8]



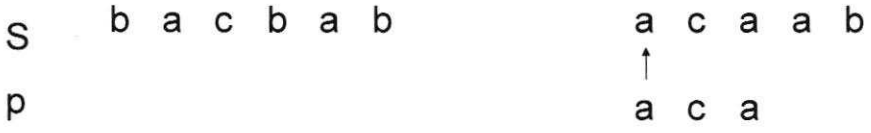
Step 9: i = 9, q = 4
Comparing p[5] with S[9] p[5] matches with S[9]

Step 10: i = 10, q = 5
Comparing p[6] with S[10] p[6] doesn't match with S[10]



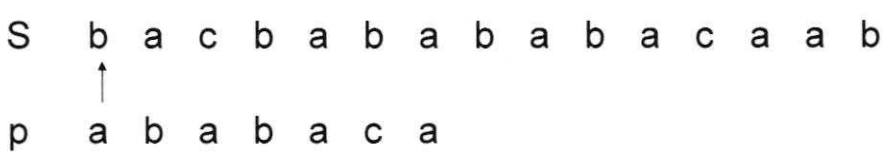
Backtracking on p, comparing p[4] with S[10] because after mismatch q = Π[5] = 3

Step 11: i = 11, q = 4
Comparing p[5] with S[11] p[5] matches with S[11]



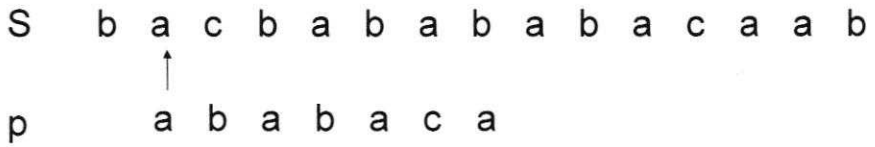
Initially: n = size of S = 15;
m = size of p = 7

Step 1: i = 1, q = 0
comparing p[1] with S[1]



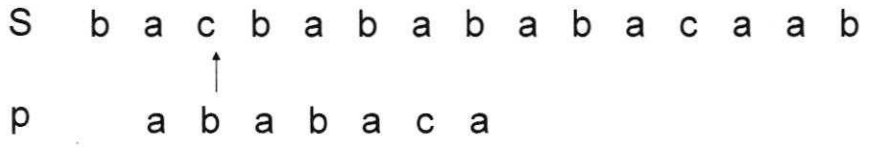
P[1] does not match with S[1]. 'p' will be shifted one position to the right.

Step 2: i = 2, q = 0
comparing p[1] with S[2]



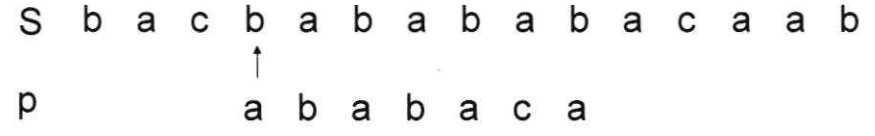
P[1] matches S[2]. Since there is a match, p is not shifted.

Step 3: i = 3, q = 1
Comparing p[2] with S[3] p[2] does not match with S[3]

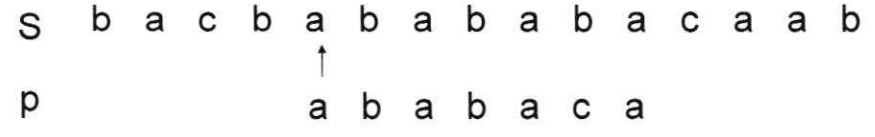


Backtracking on p, comparing p[1] and S[3]

Step 4: i = 4, q = 0
comparing p[1] with S[4] p[1] does not match with S[4]



Step 5: i = 5, q = 0
comparing p[1] with S[5] p[1] matches with S[5]

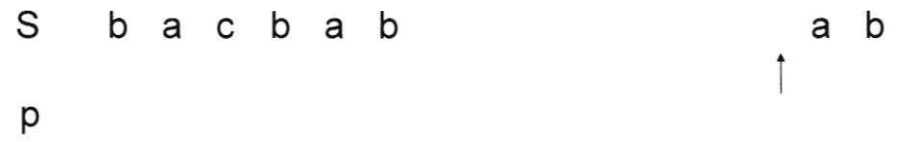


Step 12: i = 12, q = 5

Comparing p[6] with S[12] p[6] matches with S[12]

Step 13: i = 13, q = 6

Comparing p[7] with S[13] p[7] matches with S[13]



Pattern 'p' has been found to completely occur in string 'S'. The total number of shifts that took place for the match to be found are: i – m = 13 – 7 = 6 shifts.

Running - time analysis

➢ Compute-Prefix-Function (Π)

```
1  m ← length[p]          //'p' pattern to be matched
2  Π[1] ← 0
3  k ← 0
4       for q ← 2 to m
5            do while k > 0 and p[k+1] != p[q]
6               do k ← Π[k]
7               If p[k+1] = p[q]
8                 then k ← k +1
9               Π[q] ← k
10      return Π
```

In the above pseudocode for computing the prefix function, the for loop from step 4 to step 10 runs 'm' times. Step 1 to step 3 take constant time. Hence the running time of compute prefix function is Θ(m).

➢ KMP Matcher

```
1 n ← length[S]
2 m ← length[p]
3 Π ← Compute-Prefix-Function(p)
4 q ← 0
5 for i ← 1 to n
6    do while  q > 0 and p[q+1] != S[i]
7          do  q ← Π[q]
8     if p[q+1] = S[i]
9        then q ← q + 1
10    if q = m
11     then print "Pattern occurs with shift" i – m
12          q ← Π[ q]
```

The for loop beginning in step 5 runs 'n' times, i.e., as long as the length of the string 'S'. Since step 1 to step 4 take constant time, the running time is dominated by this for loop. Thus running time of matching function is Θ(n).

University Question Papers

Code No: 113BP

R13

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B.Tech II Year I Semester Examinations, December-2014
DATA STRUCTURES
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit.
Each question carries 10 marks and may have a, b, c as sub questions.

Part- A **(25 Marks)**

1.a) Find upper bound for f(n) = 3n+8. [2M]
b) Write a recursive function which converts any integer value to its equivalent Binary value. [3M]
c) Give a postfix notation of the following infix expression
A * (B + C / (D + E)) – F * (G / H + I). [2M]
d) A letter means enqueue and an asterisk means dequeue in the following sequence. Give the sequence of values returned by the get operation when this sequence of operations is performed on an initially empty FIFO queue.
J N T * U * H Y D * * * E R * * * A B * A * * B [3M]
e) Explain the applications of a Binary Tree. [2M]
f) Give the inorder and postorder traversal for the tree whose preorder traversal is A B C - - D - - E - F - -. The letters correspond to labelled internal nodes; the minus signs to external nodes. [3M]
g) What is overflow handling? [2M]
h) How many comparisons are required to find 73 in the following array using binary search12, 25, 32, 37, 41, 48, 58, 60, 66, 73, 74, 79, 83, 91, 95? [3M]
i) Define a Splay tree and explain its properties. [2M]
j) Draw the binary search tree that results from inserting the integers 57, 85, 35, 9, 47, 20, 26, 99, 93, 10 starting with 57 and ending with 10. [3M]

Part-B **(50 Marks)**

2. An algorithm takes 0.5ms for input size 100. How long will it take for input size 500 if the running time is the following?
a) Linear
b) O (N log N)
c) Quadratic
d) Cubic.

OR

3. Given a singly linked list, write a function swap to swap every two nodes e.g. 1->2->3->4->5->6 should become 2->1->4->3->6->5.

4.a) Write a program that reads in a sequence of characters and prints them in reverse order. *Hint: Use a stack*
b) Show how to implement a stack using two Queues? Analyse the run time of the stack operations?

OR

5.a) Evaluate the following postfix expression 5 9 3 + 4 2 * * 7 + * using stack show the content of the stack at each step.
b) Show how to implement a queue using two stacks? Analyse the running time of the Queue operations.

6.a) Consider the following array representation of a priority queue (i.e., the tree structure used in heap sort): 55, 50, 70, 110, 90, 120. Draw the corresponding priority queue (tree structure). If you see something wrong please explain.

b) Describe two methods of storing edges in a graph. Which requires more space.

OR

7. Given a min-heap, give an algorithm for finding the maximum element and find the complexity of the algorithm.

8. What is the complexity of Quicksort? Show the tree of calls for the quick-sort algorithm using the final element as a pivot on the array 4, 2, 3, 9, 5, 8, 6, 1.

OR

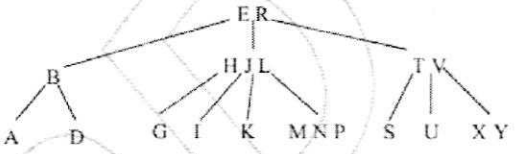
9.a) What is a hash function? Name two desirable properties of a hash function.

b) Show the steps in the in-place heap sort of 4, 7, 2, 1, 3: show the steps in heap construction and show the steps as the sort proceeds.

10. Given a Binary Search Tree write an algorithm to Tree check whether the tree is an AVL tree or not and discuss the run time complexity.

OR

11.a) Consider the following B-tree with a minimum branching factor of t = 2:



Show the B-tree that results from inserting Q into the above B-tree.

b) Explain the concept of Tries with suitable examples.

---oo0oo---

Code No: 113BP

R13

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B.Tech II Year I Semester Examinations, November - 2015
DATA STRUCTURES
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART –A **(25 Marks)**

1.a) Distinguish between Linear and Non Linear data structures. [2]
b) Write a recursive algorithm that finds all occurrences of a substring in a string. [3]
c) What is Stack? Give the declaration of all the functions used in the implementation of a stack. [2]
d) Suppose a queue is represented by a circular array of size N, F and R are used to denote front and rear positions. If F points a location before front element of queue and R points to last element of queue, how many elements are there in the queue? [3]
e) What are the ways in which a tree is represented in computer memory? [2]
f) What is the time complexity of DFS traversal as an n-vertex simple graph that is represented with adjacent matrix structure? [3]
g) Distinguish between tree and graph with an example. [2]
h) Consider an array of 100 sorted numbers. Almost how many searches are needed to search an element using Binary search. Justify your answer. [3]
i) Define Binary search tree. What are the properties of binary search tree? [2]
j) Explain the compressed trie with an example. [3]

PART-B **(50 Marks)**

2.a) Write a C function for insertion operation in a circular linked list.
b) What is algorithm? What are the properties of an algorithm? Explain the performance analysis of an algorithm. [5+5]

OR

3.a) Write an algorithm for deleting duplicate numbers from a linear array.
b) What is Sparse matrix? How Sparse matrices can be represented efficiently in memory? [5+5]

4.a) Write a function to convert a given singly linked list to doubly linked list.
b) Explain about the operations of Queue with an example. [5+5]

OR

5.a) Write a function to reverse elements.
b) Explain the operations of circular linked list. [5+5]

6.a) Create a Heap and sort the following list of elements
{12, 8, 10, 6, 24, 40, 6, 11, 9, 18, 14}
b) Explain the tree traversals with an example. [5+5]

OR

7.a) Explain how BFS can be used to identify the connected components in a graph with an example.

b) Write an algorithm that counts the number of nodes in a binary tree. [5+5]

8.a) Write a function double hash to resolve collisions using double hashing.

b) Explain the Radix sort with an example. [5+5]

OR

9.a) Write an algorithm of Binary search.

b) Insert the following list of elements in to the hash table by using linear probing (size of hash table is 10)

{16, 23, 43, 18, 34, 59, 30, 22} [5+5]

10.a) How a node can be deleted from the binary search tree? Explain the methods.

b) Construct the B-tree of order 4 for the following list of elements

{K, L, T, A, G, H, P, W, R, U, Z, C, Y, B, J, M, E} [5+5]

OR

11.a) Construct the AVL tree with the following keys

{35, 36, 80, 85, 67, 89, 25, 16, 10, 14, 50}

b) Write an algorithm of KMP. [5+5]

---ooOoo---

47

R15

Code No: 123BP

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B.Tech II Year I Semester Examinations, March - 2017
DATA STRUCTURES
(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit.
Each question carries 10 marks and may have a, b, c as sub questions.

Part – A

(25 Marks)

1.a)	What are the disadvantages of an array?	[2]
b)	Explain how to find the performance of an algorithm.	[3]
c)	What are the disadvantages of queue which is implemented using array and how to overcome it?	[2]
d)	Differentiate between doubly and circular linked lists.	[3]
e)	Explain how binary tree is represented using an array and linked list	[2]
f)	Explain the threaded binary tree with suitable example	[3]
g)	Define Hash Clashing.	[2]
h)	Compare Selection sort and Quick sort with an example.	[3]
i)	Write an algorithm to insert an element into the binary search tree.	[2]
j)	Explain the properties of Red-Black tree.	[3]

PART-B

(50 Marks)

2.a)	Write a program to concatenate singly linked lists.	
b)	How two dimensional arrays are represented in memory? Also obtain the formula for calculating the address of any element stored in the array, in case of column major order.	[5+5]

OR

3.a)	Write a program to implement a sparse matrix.	
b)	How can we represent a polynomial in a linked list?	[5+5]
4.a)	Explain the Towers of Hanoi problem with an example.	
b)	Write a program to implement the operations of Queue.	[5+5]

OR

5.a)	Write a recursive procedure to compute the n^{th} Fibonacci number.	
b)	What are the applications of queue?	[5+5]
6.a)	Write an algorithm to find the components of a graph.	
b)	Define Priority Queue? Explain with an example.	[5+5]

OR

7.a)	Differentiate between BFS and DFS.	
b)	Define Binary tree. Explain the Binary tree representations with an example.	[5+5]

8.a) Write an algorithm of Linear Search.

b) Sort the following list of elements by using Insertion Sort
15, 28,46, 10, 35, 54, 5, 17 [5+5]

OR

9.a) Insert the following list of elements into the hash table by using Linear Probing (size of the hash table is 10)
36, 48, 66, 27, 23, 87, 10, 12

b) Explain the Radix sort with an example. [5+5]

10.a) Construct the AVL tree of the following data
20, 40, 25, 18, 15, 5, 10, 46, 60

b) Draw the flow chart of splaying operations of splay tree. [5+5]

OR

11.a) Consider the string = "GCATCGCAGAGAGTATACAGTACG" and search string is "AGTATACA" by using the KMP algorithm.

b) What is trie? Explain the compressed trie with an example. [5+5]

---oo0oo---

Code No: 113BP

R13

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B.Tech II Year I Semester Examinations, March - 2017

DATA STRUCTURES

(Common to CSE, IT)

Time: 3 Hours **Max. Marks: 75**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 25 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit.
Each question carries 10 marks and may have a, b, c as sub questions.

PART- A

(25 Marks)

1.a) Compare singly and doubly linked linear lists. [2]

b) Write a recursive function in C to compute x^n where x and n are integers. [3]

c) Give the ADT specification of a stack. [2]

d) Write a C function for popping an integer item from a stack. Assume that stack is implemented as an array. [3]

e) Give the ADT specification of a max priority queue. [2]

f) Write a recursive function in C for the inorder traversal of a binary tree. Assume that binary tree is already created. Assume linked representation for binary tree. [3]

g) Sort the following list of integers in ascending order using insertion sort:
11, 41, 35, 10, -11
Show the contents of the list at the end of each pass. [2]

h) What is meant by a collision in hashing? List any two methods used for resolving collisions in hashing? [3]

i) Define a Red-Black tree. [2]

j) Write a function in C that returns the location of the smallest integer in a binary search tree of integers. Assume that binary search tree of integers is already created. Assume linked representation for the binary search tree. [3]

PART- B

(50 Marks)

2.a) Define the space complexity of a program.

b) Write a C function for deleting an integer element from doubly linked list of integer elements. Assume that the doubly linked list of integers is already created. [5+5]

OR

3.a) Explain with an example the linked representation of a sparse matrix.

b) Define the asymptotic notations (Big Oh, Omega and Theta) used in algorithm analysis. [5+5]

4.a) Write a C function for deleting an integer element from a circular queue of integers. Assume array representation for the circular queue.

b) Explain with an example how recursion is implemented using stack. [5+5]

OR

5.a) Show how to represent a deque (double ended queue) in a singly linked list.
b) Write functions in C which insert and delete integer elements at either end of the above deque. [5+5]

6.a) Give an example for a threaded binary tree.
b) Write a non recursive procedure for the preorder traversal of a binary tree. Assume that the binary tree of elements is already created. [5+5]

OR

7.a) Give an example for the adjacency list representation of a graph.
b) Write a procedure for the bfs of a graph. [5+5]

8.a) Write a recursive binary search function in C to search for an integer key in a sorted (ascending order) array of integers.
b) Compare the performance of binary search with linear search. [5+5]

OR

9.a) Write quick sort algorithm for sorting a list of integers in ascending order.
b) What is the time complexity of quick sort algorithm in the worst case? [5+5]

10.a) Define an AVL tree. Give an example for it.
b) Write a non recursive function in C to search for an integer key in a binary search tree of integers. Assume that the binary search tree of integers is already created. [5+5]

OR

11.a) What is a bottom-up splay tree?
b) Write a procedure for inserting an element into a B-tree. [5+5]

---ooOoo---

Internal Question Papers with Key

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanguda, Hyderabad.

II - B.Tech – I – Semester –R15 – I - Mid Internal Examinations. August- 2016

Sub: **DATA STRUCTURES** Date:

Branch / Section: **IT** Duration: 60 Min. Max. Marks: **10**

Answer any TWO from the following Questions

1. Write code for implementation of stack using linked list.
2. Write code to convert given infix expression to its postfix notation.
3. Write code to implement Circular Queue using arrays.
4. Write code to implement DeQueue using linked list

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Academic Year: 2016-2017 **Subject:** Data Structures

Class: II B.Tech **Faculty Name:** Mr.Neil Gogte

Mapping Questions with Taxonomy And Outcomes

Mid-1 (Aug- 2016)

S.No	Question	Taxonomy Level	Course Outcome
1	Implement Single Linked List	2	CO1, CO2
2	Convert infix expression to postfix notation.	3	CO4
3	Implement Circular Queues using Arrays.	2	CO1,CO2
4	Implement Dequeue using Linked Lists	2	CO1,CO2

Academic Year : 2016-2017

MID - 1 Key

(1Q) Implementation of Stack using Linked List

(1)

```
#include<stdio.h>
#include<malloc.h>

typedef struct stack
{
   int data;
   struct stack *next;

}node;

void main()
 {
   node *top=NULL;
   int data, item, choice;
   char ans,ch;
   void push(int, node **);
   void display(node **);
   int pop(node **);
   int Semply(node *);

 { Push(10, &top };
   push(20, &top);
   display(&top);
   Pop(&top);
   display(&top);
 }
```

```
Void push(int Item, node **top)
{
   node **New;
   node * get_node(int);
   New = get_node( Item);
   New -> next = *top;
   *top = New;
}

int pop(node **top)
{
   int item; node *temp;
   item = (*top) -> data;
   temp = *top;
   *top = (*top) -> next;
   free(temp);
   return (item);
}

void Display (node **head)
{
  node *temp;
  temp = *head;
  if (empty(temp))
      Pf (" stack Empty ");
  else
    while (temp != NULL)
      {
        Pf ("%d", temp -> data);
        temp = temp -> next;
      }
}
```

#1 Convert Infix expression to Postfix Notation :—

```
#include<stdio.h>
#include<conio.h>
struct stack
 {
    char  s[30];
    int top;
 } st;

void  main()
 {
  char infix[30];
  void intopost(char infix[30]);
  pf("Enter infix expression");
  scanf("%s", infix);
  intopost(infix);
  getch();
 }

void  intopost(char infix[30])
{
  st.top = -1;
 st.top = st.top+1;
 st.s[st.top] = '$';
 int i, j;
```

```
char ch;
int instack(char ch);
int inComing(char ch);
Void push(Char item);
char pop();
j=0;
for(i=0; infix[i] != '\0' ; i++)
{
 ch = infix[i];
 while (instack(st.s[st.top]) > inComing(ch))
    {
       postfix[j] = pop();
       j++;
    }
if(instack(st.s[st.top]) != inComing(ch))
          Push(ch);
 else
     pop();
   }
while ((ch == pop()) != '$')
   {
     postfix[j] = ch;
     j++;
   }
Postfix[j] = ch;
j++;
Postfix[j] = '\0';
Pf("%s", postfix);
}
```

```
int instack(char ch)
{
  int Priority;
  Switch (ch)
  {
   Case '+':
   Case '-':   Priority = 2;
               break;

   Case '*':
   Case '/':   Priority = 4; break

   Case '^':   Priority = 0;
               break

   Case '$':   Priority = -1;
               break;
   default :   Priority = 8;
  }
  return Priority;
}

int inComing(char ch)
{
  int Priority;
  Switch(ch)
  {
     Case '+';
     Case '-':  Priority = 1;
                break;
```

```
    case '*' :
    case '/' :  priority = 3;
                break;

    case '^' :  Priority = 6
                break;

    case '(' :  Priority = 0;
                break;

    case ')' :  priority = 0;
                break

    default  :  Priority = 7;

    }
  return  Priority;
}

void push (char item)
{
  st.top++;
  st.s[st.top] = item;
}

char pop()
{
  char e;
  e = st.s[st.top];
  st.top--;
  return e;
}
```

(3Q) Implement the Concept of Circular Queue using Arrays

```
# include <stdio.h>
# include <conio.h>
 # define  size  5
 int  queue[size];
 int  front = -1;
 int  rear = 0;

 int qFull()
 {
  if (front == (rear+1) % size;
       return 1;
    else
         retun 0;
  }

 int qEmpty()
  {
 if (front == -1)
    return 1;
 else
     return 0;
  }
```

```
Void add(int Item)
{
 if (qFull())
    Printf (" Queue full ");
 else if front == -1 { front = rear = 0 } else
    rear = (rear +1) % Size;
    queue[rear] = Item;
 }
```

```
Void delet()
  { int Item;
    if ( qEmpty() )
       Printf (" Queue is Empty ");
    else
    {
     Item = queue[front];
     if ( front == rear)
       {
          front = rear = -1;
       }
     else
          front = (front +1) % Size;
     Printf (" Deleted item %d ", Item);
    }
  }
```

```
Void display()
{
  int i;
  if (qEmpty())
    {
      Printf("The Queue is Empty");
      return;
    }
  i = front;
  while (i!=rear)
    {
      Printf("%d", queue[i]);
      i=(i+1) % size;
    }
  Printf("%d", queue[i]);
}
```

(Q4) Implement DeQueue

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Dequeue
{
   int data;
   struct Dequeue * prev;
   struct Dequeue * next;
} node;
```

```
node * Create ( )
{
  node *temp, *New, *head;
  int val, flag;
  char ans = 'y';
  node * get_node();
  temp = NULL;
  flag = TRUE;
  do {
    Printf ("Enter the element ");
    Scanf ("%d", &val);
    New = get_node();
    If (New == NULL)
        Printf ("Memory allocated");
    New -> data = val;
    if (flag == TRUE)
      {
        head = New;
        temp = head;
        flag = FALSE;
      }
    else {
        temp -> next = New;
        New -> Prev = temp;
        temp = New;
      }
```

```
        Printf (" Continue (y/n) ");
        ans = getche();
        }
        while (ans == 'y');
        Pf (" DeQueue Created ");
        getch();
        return head;
    }

Void display()
{
    node *temp;
    temp = head;
    if (temp == NULL)
    {
        Pf (" DQ Empty ");
        getch();
        return;
    }
    while (temp -> next != NULL)
    {
    Pf (" %d -> ", temp -> data);
    temp = temp -> next;
    }
}
```

```
node *insertFront(node *head)
{
node *frontNode, *temp;
frontNode = get_node();
scanf("%d", &frontNode->data);
if(head == NULL)
    head = frontNode;
else
    {
      temp = head;
      frontNode->next = temp;
      temp->prev = frontNode;
      head = frontNode;
    }
}

Void insertRear(node *head)
{
  node *rearNode, *temp;
  rearNode = get_node();
  scanf("%d", &rearNode->data);
  if(head == NULL)
      head = rearNode;
  else
    {
      temp = head;
```

```
while (temp -> next != NULL)
        temp = temp -> next;
temp -> next = rearNode;
rearNode -> prev = temp;
rearNode -> next = NULL;
}
}
```

```
node * deleteFront (node *head)
{
  node *frontNode;
  frontNode = head;
  if (frontNode == NULL)
    {
      Pf ("dequeue Empty");
    }
head = frontNode -> next;
head -> prev = NULL;
frontNode = NULL;
free (frontNode);
Printf ("Node deleted");
return head;
}
```

```
Void deleteRear ( node *head)
{
 node *rearNode;
 rearNode = head;
 If (rearNode == NULL)
 {
  Printf ("dequeue is Empty");
  return;
 }
While ( rearNode -> next != NULL)
     rearNode = rearNode -> next;
(rearNode) -> (prev) -> next = NULL;
rear Node = NULL;
free (rear Node);
Printf ("Node deleted");
}
```

Code No:113BP **Set No. 1**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
II B.Tech. I Sem., I Mid-Term Examinations, August- 2016
DATA STRUCTURES
Objective Exam

Name: _____ **Hall Ticket No.** | | | | | | A | | | | |

Answer All Questions. All Questions Carry Equal Marks. Time: 20 Min. Marks: 10.

I Choose the correct alternative:

1. First filed of a node in singly linked list represents_____ [D]
   A) Float B) Integer C) Character D)Data

2. Which of the following is not required criteria for algorithm [D]
   A) Input B)Output C)Solvable D)Process

3. Which data structure is needed to convert infix to postfix notation. [B]
   A) Tree B) Stack C) Linear list D)Queue

4. The postfix equivalent of the prefix *+AB-CD is _____ [B]
   A)AB+CD-* B)ABCD+-* C)AB+CD*- D)(A+B)*(C-D)

5. If the insertions and deletions happen from both the ends then the queue is called [A]
   A) Dequeue B) Header queue C)queue D)Circular queue

6. Degree of a leaf node in a binary tree is [B]
   A) 3 B)0 c)2 D)2

7. Inorder:A/B+C*D-E*F, Pre-order:-+/AB*CD*EF find Post-order [D]
   A) AB/CD+E-*F*/ B)AB/C+D*E-F* C)/ABC+DE-*F*/ D)AB/CD*+EF*-

8. A full binary tree with n leaf nodes contains _____nodes [C]
   A) n B)$\log_2 n$ C)2n-1 D)2n

9. The in-order and pre-order traversals of binary tree are dbeafcy and abdecfg respectively [A]
   Find post-order of the binary tree
   A) debfgca B)edbgfca C)edbfgca D)defgbca

10. If a top is initialized to -1 and the position of top is at 4, no.of elements in a stack are____[B]
    A) 4 B)5 C)3 D)0

Cont.....2

II Fill in the Blanks

11. Head contains address of first node in a single linked list.

12. In Doubly Linked List, previous address field of the first node and next address field of the last node are NULL

13. Double Single linked list is mostly used in implementation of stack.

14. The level of a root node in a binary tree is 0

15. FIFO stands for First In First Out

16. Algoritem is composed of a finite set of steps each of which may require one or operations.

17. Memory holds all the static data and dynamic data.

18. Circular Linked List is a singly linked list except that the last element points to the first element.

19. Linked list is a collection of nodes.

20. Expand ADT Abstract Data Type

-oOo-

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanguda, Hyderabad.

II - B.Tech – I – Semester –R15 – II - Mid Internal Examinations. NOV- 2016

Sub: **DATA STRUCTURES** Date:

Branch / Section: **IT** Duration: 60 Min. Max. Marks: **10**

Answer any TWO from the following Questions

1. Explain in detail about Hashing.
2. Write code to arrange the elements in ascending order using heap sort.
3. Explain with an example LL, LR, RL and RR rotations of an AVL tree.
4. Write KMP algorithm for pattern matching. Explain with an example.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Academic Year: 2016-2017 **Subject: Data Structures**

Class: II B.Tech **Faculty Name: Mr. Neil Gogte**

Mapping Questions with Taxonomy And Outcomes

Mid-2 (Nov- 2016)

S.No	Question	Taxonomy Level	Course Outcome
1	Explain in detail about Hasing Technique.	3	CO4
2	Write code to Implement Heap Sort Algorithm.	3	CO3
3	Explain about LL, LR, RR, RL rotations of AVL tree.	2	CO2
4	Write code to Implement KMP pattern matching algorithm.	4	CO4,CO5

Academic Year : 2016 – 2017

MID – II Key

(19) Explain indetail about Hashing

Hashing is an effective way to store the elements in some data structure. It allows to reduce the number of comparisons. Using the hashing technique we can obtain the concept of direct access of stored record.

Hash Table

Is the Data structure used for storing and retrieving data very quickly. Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associated with some key. For example for storing an employee record in the hash table the emp id will work as key.

Using the hash key the required piece of data can be searched in the hash table by few or more key comparisons. The searching time is then dependant upon the size of the Hash Table.

The Effective ~~hash~~ key representation of dictionary can be done using hash table. we can place the dictionary entries (key, value) in hes table using hash function.

Hash Function :—

Takes a key value as input and Produces the address at which the given the record with the given key should be placed in the Hash Table

Hashing Methods

1. Division method :—

The Hash function depends upon the remainder of division.

divisor will be table length.

$h(key) = record \% table\_Size.$

Ex : Size = 10, record = 54

$\Rightarrow$: $54 \% 10$

: 4

In 4th Cell we have place record 54.

2. Mid square Method :—

The key is squared and the middle or mid Part of the result is used as the index.

Ex : record = 3111 $\Rightarrow (3111)^2 = 9678321$.

Let Size = 1000.

$H(3111) = 783$

Multiplicative Hash function

The given record is multiplied by some constant value.

H(key) = floor(P * (fractional part of key * A))

P → integer constant
A → real number.

Prefered A = 0.6180

If key = 107 & P = 50 then

H(key) = floor(50 * (107 * 0.6180))
= floor(3306.4818)
= 3306.

⇒ Record 107 should be placed at address '3306'

Digit Folding

Key is divided into separate parts using some simple operation these parts are combined to produce the hash key.

Ex :- let record = 12365412

divide as 123 654 12.

H(key) = 123 + 654 + 12
= 0789.

⇒ Location = 789.

(Q2) Implement Heap Sort algorithm.

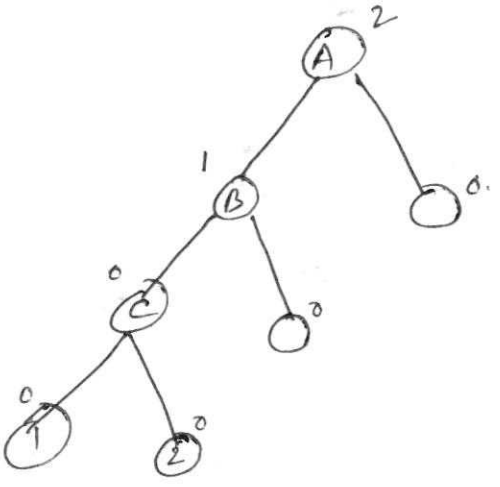
```
#include <stdio.h>
#include <stlib.h>.

#define MAX 10.

void main()
{
   int i, n;
   int arr[MAX];
   void makeheap (int arr[MAX], int n);
   void heapSort(int arr[MAX], int n);
   void display(int arr[MAX], int n);

   for(i=0; i<MAX; i++)
       arr[i] = 0;

   scanf("%d", &n);

   for(i=0; i<n; i++)
       scanf("%d", &arr[i]);

   display(arr,n);
   makeheap(arr, n);
   heapSort(arr, n);
   display(arr,n);
   getch();
}
```

```
void makeheap(int arr[MAX], int n)
{
  int i, val, j, father;
  for(i=1; i<n; i++)
  {
     val = arr[i];
     j=i;
     father = (j-1)/2;
     while( j>0 && arr[father] < val)
        {
           arr[j]= arr[father];
           j = father;
           father = (j-1)/2;
        }
     arr[j] =val;
  }
}

void heapsort(int arr[MAX], int n)
{
  int i,k, temp, j;
  for(i=n-1; i>0; i--)
  {
    temp = arr[i];
    arr[i]= arr[0];
    k=0;
    if(i==1)  j= -1;
    else
       j=1;
```

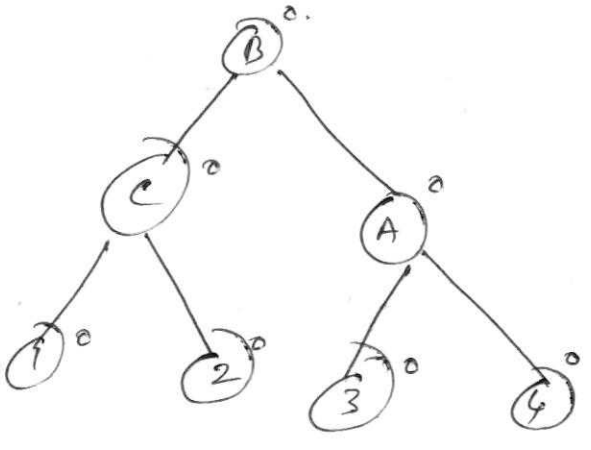
```
if ( i>2 && arr[2] > arr[1] )
        j = 2 ;
while ( j>=0 && temp< arr[j])
  {
     arr[k]= arr[ j];
     k = j ;
     j = 2 * k+1;
     if ( j+1 <= i-1 && arr[j] < arr[j+1])
            j++;

     if ( j > i-1)
          j = -1;
  }
  arr[ k] = temp;
 }

void display (int arr[MAX], int n)
{
  int i;
 for(i=0; i<n ; i++)
    Printf ("%d", arr[i]);
}.
```

(3Q) AVL Tree Rotations

(i) LL Rotation :—

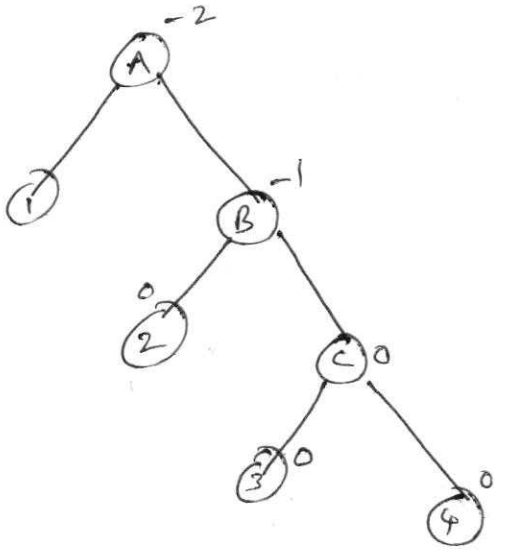


After LL Rotation at the root node, we get



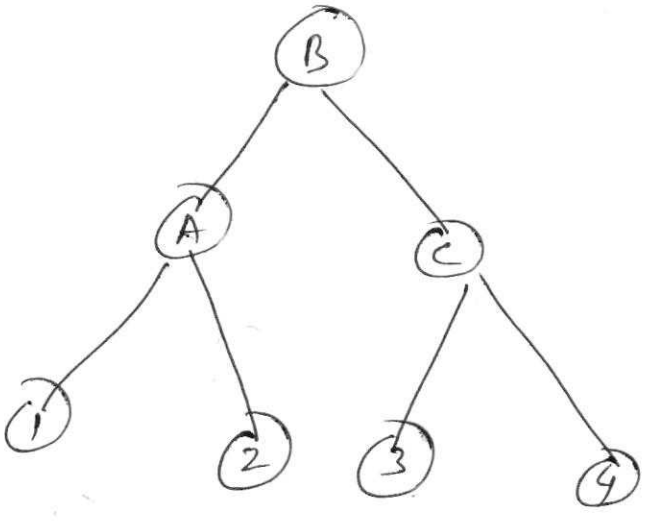
The Resultant tree is balanced.

(ii) RR Rotation :—

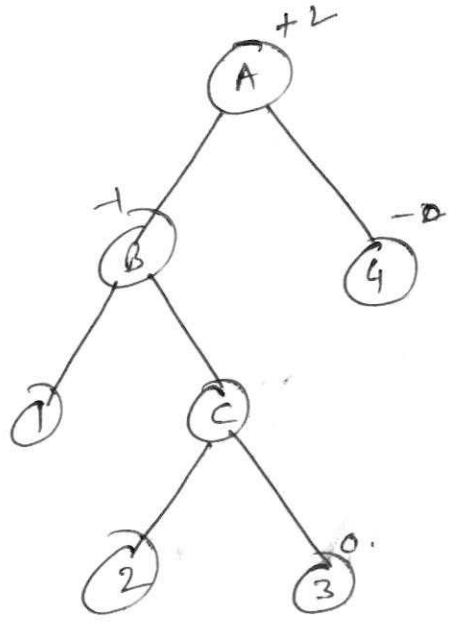


Apply RR Rotation at root.

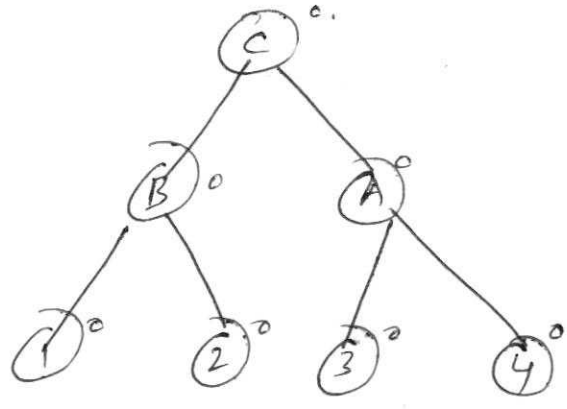
where BF = -2.



(iii) LR Rotation :—

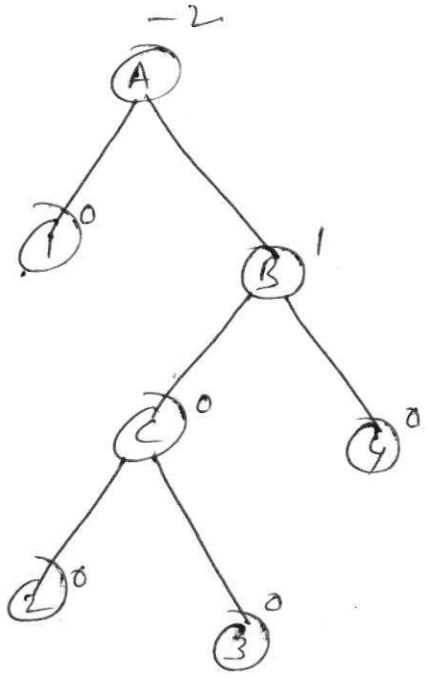


Applying LR Rotation, we get

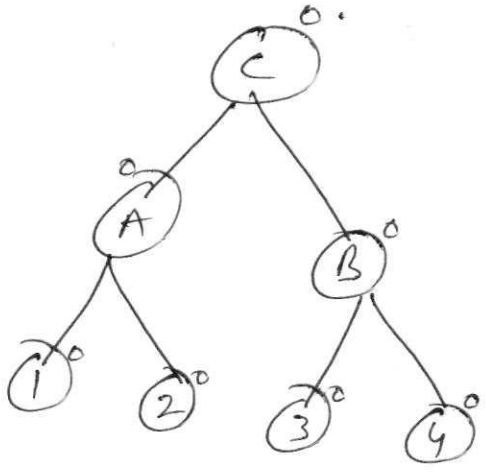


The Resultant tree after the LR rotation is a balanced tree.

RL Rotation :—



Applying R-L Rotation at root node, we get



(48) KMP Pattern Matching Algorithm :—

```
kmp_match ( char t[50], char p[10])
{
  j ← 0 ;
  n = strlen( t);
  m = strlen (p);
  Prefix_table = Create_Prefix_table (p);
 for (i ← 0 to i<n) do
 {
  while ( j >0 And p[j] != t[i]) do
      j ← prefix_table [ j-1]
  if ( p[j] == t[i]) then
     j++;
  if ( j == m) then
  {
   write (" pattern is Present ");
   write ( i-m+1);
  }
  j ← Prefix_table [ j-1];
 }
}
```

Code No:113BP **Set No. 1**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
II B.Tech. I Sem., II Mid-Term Examinations, November-2016
DATA STRUCTURES
Objective Exam

Name: _____ **Hall Ticket No.** | | | | | | A | | | | |

Answer All Questions. All Questions Carry Equal Marks. Time: 20 Min. Marks: 10.

I Choose the correct alternative:

1. Which of the following uses the divide-and-conquer design strategy [a]
   a) Binary Search b) Linear Search c) Insertion Sort d) Selection Sort

2. Which of the following an external sorting technique [b]
   a) Quick Sort b) Merge Sort c) Selection Sort d) Insertion Sort

3. Which of the following is the best sorting technique to sort large data sets [d]
   a) Insertion Sort b) Selection Sort c) Bubble Sort d) Quick Sort

4. Which of the following is not an application of Stacks? [d]
   a) Checking the Balancing of parenthesis in an expression
   b) Converting an expression from infix to postfix
   c) Evaluating the Postfix Expression
   d) Keeping track of the jobs submitted to a printer

5. Which of the following is an equivalent postfix form of the infix expression (Based on C language Precedence and Associative Rules) : A + B * C – D / E [b]
   a) AB+C*D-E/ b) ABC*+DE/- c) ABC*+D-E/ d) AB+CD-*E/

6. The average depth of binary search tree is_____ [c]
   a) o(n/2) b) o(n) c) o(log n) d)o(n^2)

7. External merge sort is based on the _____-on the external merging principle [c]
   a) External sorting b) Poly phase merge c) multi phase merge d) internal sorting

8. The worst case running time for search in AVL tree is_____ [c]
   a) o(n/2) b) o(n) c) o(log n) d) o(n^2)

9. Comparisons in Brute-Force is _____ [c]
   a) from middle b) start to middle c) left to right d)right to left

10. In a graph set of nodes are called as _____ [c]
    a) notes b) arrows c) vertices d) edges

Cont.....2

II Fill in the Blanks

11. Binary searching technique requires the elements to be in sorted order
12. Time Complexity (efficiency) of the linear search is O(n)
13. Time Complexity of the Selection Sort is $O(n^2)$ -
14. Collection of elements and their relationships are called as Data Structure
15. The level of root node in binary tree is 0
16. AVL tree is based on balancing
17. In a B-tree the data items are stored at Leaf
18. DFS stands for Depth First Search
19. DAG stands for Directed Acyclic Graph.
20. Heaps are used for Priority queues.

-oOo-

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanguda, Hyderabad.

II- B.Tech – I – Semester –R13 – I - Mid Internal Examinations. AUG- 2015

Sub: **DS**

Branch / Section: IT Duration : 60 Min. Max. Marks: **10**

Answer any TWO from the following Questions

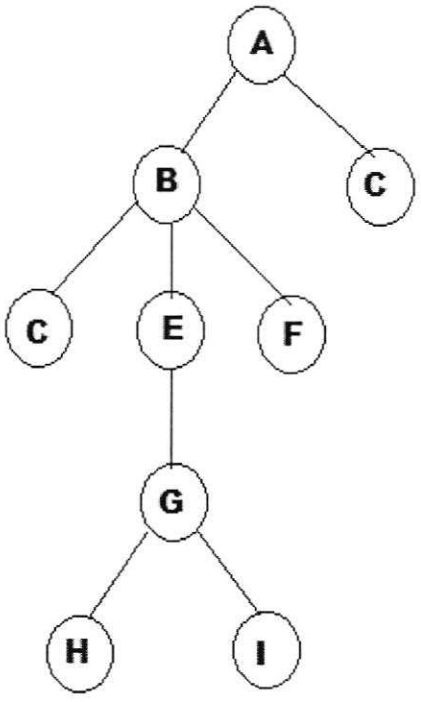
1. (a) What is Algorithm and criteria for writing algorithm?
   (b) What is recursive algorithm, Explain with example of factorial? And find its time complexity.

2. (a) What is Asymptotic notation explain theta notation for f (n) =3n+2.
   (b) Explain circular link list. Write the code for insertion & display functions.

3 . (a) Write an algorithm for infix to postfix and trace the opstack (operator stack) for the following expression A*(B+C/(D+E))-F*(G/H+I)

   (b) Write a program for implementation of Queue using Arrays.

4. (a) Give the values of properties of a tree as given below.



Property	Values
No. of Nodes	
Height	
Root Node	
Leaves	
Interior Nodes	
Ancestors of H	
Descendant of B	
Sibling of E	
Right subtree of A	
Degree of tree	

(b) Write C programs to implement a double ended queue using array.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Narayanguda, Hyderabad.

II- B.Tech – I – Semester –R13 – II - Mid Internal Examinations. OCT- 2015

Sub: **DS**

Branch / Section: IT Duration : 60 Min. Max. Marks: **10**

Answer any TWO from the following Questions

1. (a) Explain the Graph traversal algorithms in detail with example.

   (b) Define the following

   Hash Table, Hash Functions and Collision resolution techniques.

2. (a) Arrange the following elements in Max heap. Showing the Right / Left shift how will you re-heap when root element is deleted. 25,22,17,19,22,14,15,18,14,21,3,9,11

   (c) Differentiate between Depth-First search and Breadth-First search traversal of a graph.

3. (a) Write the program for insertion sort. Give the complete sequence of in which the array containing following elements will be sorted 29,10,14,37,13 in insertion sort order.

   (b) Arrange the nodes of Red-Black tree when each node value comes in following order: 47,17,26,30,41,50,38. What is height and Black Height of Red-Black Tree. Specify height and Black height of each node. Also prove that the sub-tree rooted at any node x contains **at least** $2^{bh(x)}$ - 1 internal nodes. State the rules for creating Red-Black Tree.

4. (a) Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45.

   (b) Give the sequence in which Quick sort can be carried out for following elements. 3,7,8,5,2,1,9,5,4 and Write c program to perform quick sort.

Code No:113BP Set No. 1

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

II B.Tech. I Sem., I Mid-Term Examinations, August- 2015

DATA STRUCTURES

Objective Exam

Name: _____ **Hall Ticket No.** | | | | | | A | | | | |

Answer All Questions. All Questions Carry Equal Marks. Time: 20 Min. Marks: 10.

I Choose the correct alternative:

1. A technique for direct search is [ ]
   (a) Binary Search (b) Linear Search
   (c) Tree Search (d) Hashing

2. Which of the following sorting algorithms does not have a worst case running time of O(n2) ? [ ]
   (a) Insertion sort (b) Merge sort
   (c) Quick sort (d) Bubble sort

3. The data structure required for Breadth First Traversal on a graph is [ ]
   (a) queue (b) stack
   (c) array (d) tree

4. A BST is traversed in the following order recursively: Right, root, left [ ]
   The output sequence will be in
   (a) Ascending order (b) Descending order
   (c) Bitomic sequence (d) No specific order

5. In order to get the contents of a Binary search tree in ascending order, one has to traverse it in [ ]
   (a) pre-order. (b) in-order.
   (c) post order. (d) not possible.

6. The number of interchanges required to sort 5, 1, 6, 2 4 in ascending order using Bubble Sort is
   (a) 6 (b) 5 [ ]
   (c) 7 (d) 8

7. AVL tree is a _ _ _ _ _ _ _ _ _ binary tree [ ]
   a) Complete b) Full c) Height balanced d) Skewed

8. A priority queue can be implemented by [ ]
   a) Heap b) BST c) DFS method d) AVL Tree

9. In a red black tree, a root node is _____ leaf node is _____ [ ]
   a) Black, Red b) Red, Black c) Black, Black c) Red, Red

10. Graph can be represented by [ ]
    a) Adjacency matrix b) adjacency list c) queue d) both a & b

Cont...2

II FIIL IN THE BLANKS

11. AVL tree was developed by ----------

12. In a max heap the child element should be _____ than parent element

13. The permissible balance factors of an AVL trees are ----------

14. Graph is a collection of _____ and _____

15. Merge sort uses_____ strategy

16. The difference between the height of left sub tree & right sub tree is called _____

17. A binary search tree is constructed with the following keys 20,22,26,21,13,19,18,15,26,28 .
The above keys are inserted in that order. Then the total keys in the left sub tree and the right sub tree of the tree are _____ , _____

18. _____ is a technique of finding the substring in text which is equal to pattern.

19. In AVL , when an element inserted at left sub tree of left child then _____ rotation will be performed.

20. In a BST the left child should be _____ than parent element

-oOo-

Code No: 113BP **Set No. 1**

DATA STRUCTURES

MID-1 KEYS

I Choose the correct alternative

1. D

2. D

3. B

4. A

5. A

6. B

7. D

8. C

9. A

10. B

II Fill in the blanks

11. Second node

12. NULL

13. Single

14. 0

15. First In First Out

16.Algorithm

17. Stack

18. Circular linked list

19. Linked list

20. Abstract Data Type

Code No: 113BP **Set No. 1**

DATA STRUCTURES

MID2 KEYS

I Choose the correct alternative

1. D

2. B

3. A

4. B

5. A

6. B

7. C

8. A

9. A

10. D

II Fill in the blanks

11. Adel'son-Velskii and Landis

12. Smaller

13. -1,0,+1

14. Vertices and Edges

15. Divide and Conquer

16 Balance factor

17.4,5

18. Pattern matching

19. LL Rotation

20. <=

Assignment Question Papers

(15)
Assignments

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Academic Year: 2016-2017 **Subject: Data Structures**

Class: II B.Tech **Faculty Name : Neil Gogte**

Mapping Questions with Taxonomy And Outcomes

ASSIGNMENT-1

S.No	Question	Taxonomy Level	Course Outcome
1	Implement Towers of Hanoi problem using Recursion.	3	CO4
2	Implement In-order Traversal of Tree using an Interative (non-recursive approach).	3	CO2,CO3
3	Implement Dictionary operations using Hashing Techniqus.	3	CO4

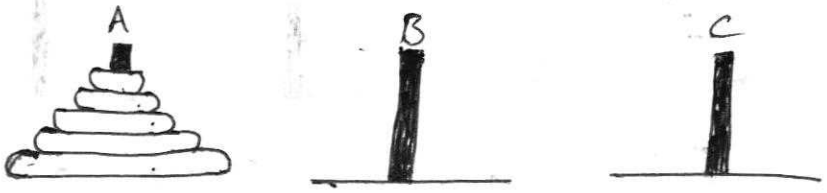
AY : 2016 - 17

TOWERS OF HANOI

Assignment - 1

Roll No: 15BDIA1250

(1Q) The problem is the "Towers of Hanoi". The initial setup is as shown in fig.



There are three pegs named as A, B and C. The five disks of different diameters are placed on peg A. The arrangent of the disks is such that every smaller disk is placed on the larger disk.

The problem of "Towers of Hanoi" states that move the five disks from peg A to peg C using peg B as a auxiliary.

The conditions are:

i) Only the top disk on any peg may be moved to any other peg.

ii) A larger disk should never rest on the smaller one.

iii) The above problem is the classic example of recursion. The solution to this problem is very simple.

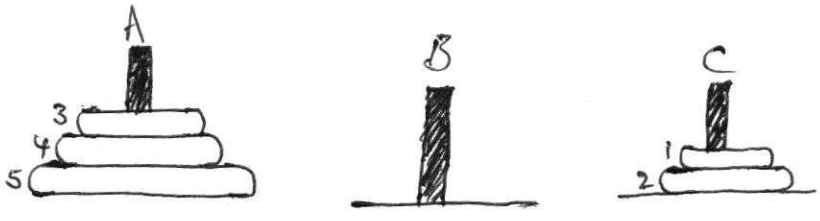
First of all let us number out the disks for our comfort.

The solution can be stated as

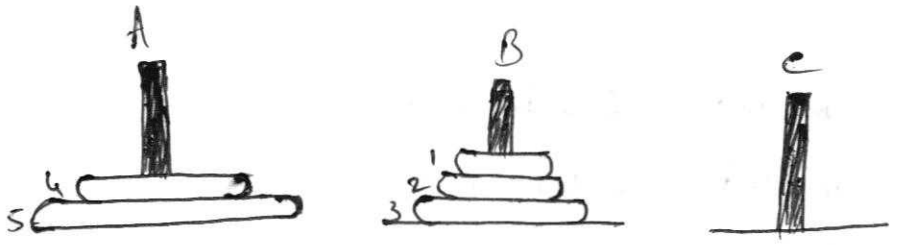
1. Move top n-1 disks from A to B using C as auxiliary.
2. Move the remaining disk from A to C.
3. Move the n-1 disks from B to C using A as auxiliary.

We can convert it to

move disk 1 from A to B.
move disk 2 from A to C.
move disk 1 from B to C.



move disk 3 from A to B
move disk 1 from C to A
move disk 2 from C to B
move disk 1 from A to B



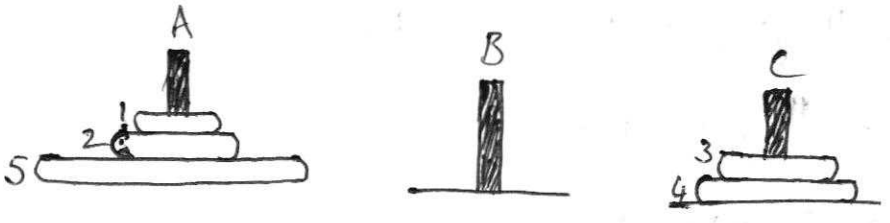
move disk 4 from A to C

move disk 1 from B to C

move disk 2 from B to A

move disk 1 from C to A

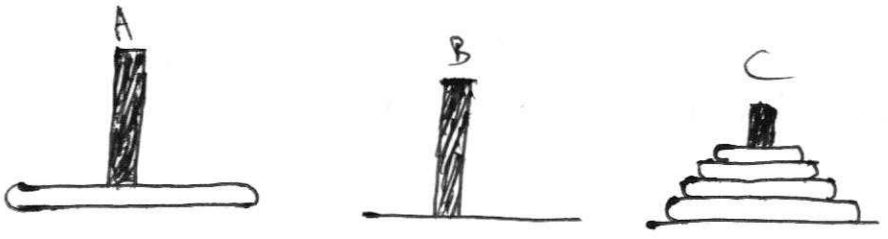
move disk 3 from B to C



move disk 1 from A to B

move disk 2 from A to C

move disk 1 from B to C



This actually we have moved n-1 disks from peg A to C. In the same way we can move the remaining disk from A to C.

```
# include <stdio.h>
# include <conio.h>
# include <studlib.h>

void main (void)
{
int n;
void towers (int n, char from, char to, char aux);
clrscr ();
printf ("\n\t\t Program For Towers Of Hanoi");
print f ("\n\n Enter the total number of disks");
scanf ("%d" &n);
towers (n, 'A', 'C', 'B');
getch ();
}
void towers (int n, char from , char to , char aux)
{
/*if only one disk has to be moved*/
if (n==1)
{
    print f ("\n Move disk 1 from %c peg to %c peg",
                                    from, to);
}
/*move top n-1 disks from A to B using C*/
towers (n-1, from , aux , to);
```

```
printf("\n Move disk %d from %c peg to %c peg",
                                    n, from, to);
/* move remaining disk from B to C using A */
towers (n-1, aux, to, from);
}
```

(20)

NON RECURSIVE IN-ORDER TRAVERSAL

```
Inorder display of tree
*/
Void inorder (node *root)
{
node *current, *S[10];
int top = -1;
if (root == NULL)
{
    printf ("\n Tree is empty \n");
    return;
}
current = root;
for (;;)
{
```

```
        while (current != NULL)
        {
            push (current, &top, s);
            current = current -> left;
        }
        if (! stempty (top))
        {
            pop (&top, s, &current);
            printf (" %d", current -> data);
            current = current -> right;
        }
        else
            return.
    }
}
```

(39) Implement Dictionary operations using. (4)

hashing.

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
# include <math.h>
# define MAX 10
struct DCT
{
        int k;
        int val;
} a[MAX];

int HashDivMethod(int);
int HashMulMethod(int);
void init();
void insert(int, int, int);
void display();
void size();
void search(int);

Void init()
{
for (int i=0; i<MAX; i++)
{
```

```
            a[i].k = -1;
            a[i].val = -1;
        }
    }
    int HashDivMethod(int num)
    {
    int Hkey;
    Hkey = num % 10;
    return Hkey;
    }
    int HashMulMethod(int num)
    {
    int Hkey;
    double A = 0.6180;
    int p=1;
    Hkey = (int)floor(num*p*A);
    return Hkey;
}
void insert(int index, int key, int value)
{
int flag, i, count=0;
flag = 0;
if(a[index].k == -1)/* if the location indicated by hash
                       key is empty*/
```

```
{
        a[index].k  = key;
        a[index].val = Value;
}
else
{
        i = 0;
        while(i<MAX)
        {
        if (a[i].k != -1)
                count++;
        i++;
    }
    if (count == MAX)     /*checking for the hash full*/
    {
    printf("\n Hash Table Is full");
    }
    for (i= index+1; i<MAX; i++) /* moving linearly down*/
    if (a[i].K == -1)  /* searching for empty location*/
{
        a[i].k = key;
        a[i].val = value;
                /*placing the number at empty location*/
```

```
flag =1;
break;
        }
/* From key position to the end of array
we have searched empty location and now
we want to check empty location in the
upper part of the array */
        for (i= 0; i< index && flag == 0; i++)/*
        array from 0th to keyth location
        will be scanned */
      if (a[i].k == -1)
      {
          a[i].k = key;
          a[i].val = value;.
          flag = 1;
          break ;
      }
} /* outer else*/
} /* end */
```

```
void display()
{
int i;
printf("\n The Hash Table is ....\n");
printf("\n ----------");
for (i=0 ; i < MAX ; i++)
{
    printf("\n %d \t %d \t %d", i, a[i].k, a[i].val);
}
  printf("\n ----------");
}

void size()
{
 int len =0,i;
for (i=0,; i < MAX , i++)
{
     if (a[i].k! = -1)
     len ++;
}
void search (int search_key )
{
int i,j;
```

```
i = Hash Div Method (search_key);
if (a[i].k == search_key)
 {
      printf("\n The Record is present at %d
                    location ", i);
      return;
 }
if (a[i].k != search_key)
 {
 // searching from hash key to end of
    hash table
for (j = i ; j < MAX ; j++)
{
    if (a[j].k == search_key)
     {
      printf ("\n The Record is present at %d
                    location ", j);
      return;
     }
}
    // searching from beginning of hash table
    upto hash key
 for (j=0; j<i; j++).
{
```

```
            if (a[j].k == search_key)
            {
                printf ("\n The Record is present at %d
                        location", i);
                return;
            }
        }
    }
    else
        printf ("\n The Record is not present in
                the hash table");
}
```

ASSIGNMENT - 1

AY : 2016-17

SUB : Data structures

ROLL NO : 15BDIA1217

(1a)

Implementation of Towers of Hanoi :-

```
#include<stdio.h>
Void towers( int, char, char, char);
Int main()
{
  int num;
  Printf (" Enter the no. of disks ");
  Scanf (" %d ", &num);
  Printf ( " The sequence of moves are :\n ");
  towers (num, 'A', 'C', 'B');
  return 0;
}
Void towers (int num, char frompeg, char topeg, char auxpeg)
{
  If ( num == 1) {
     Printf ( " Move disk 1 from Peg %C to Peg %C", frompeg, topeg);
     return ;
  towers( num-1, frompeg, auxpeg, topeg);
  Printf ( "\n move disk %d from Peg %C to Peg %C ",
                num, frompeg, topeg);
  towers (num-1, auxpeg, topeg, frompeg);
}
```

(28) Non Recursive Inorder Traversal implementation in 'c' lang :-

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>

typedef struct bint
{
  int data;
  struct bint *left, *right;
}node;

node *create(node *r, int d)
{
   if(r== NULL)
     {
       r= (node*) malloc(sizeof(node));
       r->data=d;
       r->left = r->right = NULL;
     }
  else
      if(r->data <=d)
          r->right = create(r->right,d);
      else
          r->left = create(r->left,d);
     }
  return r;
}
```

```
Void non_in (node *r)
{
  int top=0;
  node *s[20], *ptr= r;
  s[0] = NULL;
  while ( pt != NULL)
  {
     s[++top] = pt;
     pt = pt->left;
  }
 pt = s[top--];
 while ( pt != NULL)
 {
   Pf ("%d\t", pt-> data);
   if ( pt-> right != NULL)
     {
       pt = pt-> right;
       while ( pt != NULL)
          {
             s[++top] = pt;
             pt= pt-> left;
          }
     }
   pt= s[top--];
 }
}
```

```
Void  main()
{
  int d;
  Char ch = 'y';
  node *head = NULL;
  clrscr();

  while( toupper(ch) == 'y')
  {
   Printf ("\n Enter the item to insert");
   Scanf ("%d", &d);
   head = Create(head, d);
   Printf ("\n Do you wish to Continue (y/n) ");
   fflush( Stdin);
   ch = getchar();
  }

 non_in(head);
 getch();
}
```

(3Q) IMPLEMENT DICTIONARIES USING HASHING

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 10

struct DCT
{
   int K;
   int val;
} a[MAX];

int HashDivMethod(int);
int HashMulMethod(int);
Void init();
Void insert(int, int, int);
Void display();
Void size();
Void search(int);

Void init()
{
  for(int i=0; i<MAX; i++)
     {
       a[i].k = -1;
       a[i].val = -1;
     }
}
```

```
int HashDivMethod(int num)
{
  int Hkey;
  Hkey = num % 10;
  return Hkey;
}

int HashMulMethod(int num)
{
  int Hkey;
  double A=0.6180;
  Int P=1;
  Hkey =(int) floor(num *P*A);
  return Hkey;
}

void insert(int index, int key, int value)
{
 Int flag, i, count=0;
 flag=0;

 if(a[index].k == -1)
    {
      a[index].k= key;
      a[index].val=value;
    }
```

```
else
{
  i=0;
  while ( i < MAX)
  {
   if (a[i].k != -1)
        Count++;
   i++;
  }
  if (Count == MAX)
  {
    Printf( " Hash Table is Full ");
  }
 for(i=index+1 ; i<MAX ; i++)
 if (a[i].k == -1)
 {
   a[i].k = key;
   a[i]. val = value;
   flag=1;
   break;
 }
for(i=0; i<index && flag==0 ; i++)
if (a[i].k == -1)
  {
    a[i].k = key;  a[i].val = value;
    flag=1; break;
  }
 }
}
```

```
void display()
{
 int i;
 for(i=0; i<MAX ; i++)
 {
  printf("%d %d %d ", i, a[i].k, a[i].val);
 }
}

void size()
{
 int len=0, i;
 for(i=0; i<MAX ; i++)
 {
   if (a[i].k != -1)
         len++;
 }
 Pf ("The size of Dictionary = %d ", len);
}

void search(int search_key)
{
 int i, j;
 i = HashDivMethod (search_key);
 if (a[i].k == search_key)
   {
     Pf ("The Record is present at %d loc ", i);
     return ;
   }
}
```

```
if (a[i].k != search_key)
{
 for(j=i ; j<MAX ; j++)
  {
   if (a[j].k == search_key)
     {
       Pf(" Record present at loc = %d ",j);
       return;
     }
  }

 If
for(j=0 ; j<i ; j++)
{
  if (a[j].k == search_key)
   {
     Printf (" Record present at loc = %d ", j);
     return;
   }
  }
 }
else
{
 Printf(" Record is not present in the Hash
          Table ");
  }

}
```

```
void main()
{
  int key, value, HKey, search_key, choice;
  char ans;
  init();
  do
  {
    scanf("%d", &choice);
    switch(choice)
    {
    case 1 : Printf("\n Enter the key");
             scanf("%d", &key);
             Printf "Enter value");
             scanf("%d", &value);
             HKey = HashDiv Method(key);
             insert(HKey, key, value);
             break;
    case 2 : Printf("Enter key");
             scanf("%d", &key);
             scanf("%d", &value);
             HKey = Hash Mul Method(key)
             insert(HKey, key, value));
             break;
```

```
Case 3 : display();
         size();
         break;

Case 4 : Printf ("Enter Key to Search");
         scanf ("%d", &search_key);
         Search(search_key);
         break;

 }

Printf ("\n Do you wish to Continue (y/n)");
ans = getche();

}
while (ans == 'y');

getch();

}.
```

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Academic Year: 2016-2017 **Subject: Data Structures**

Class: II B.Tech **Faculty Name: Neil Gogte**

Mapping Questions with Taxonomy And Outcomes

ASSIGNMENT-2

S.No	Question	Taxonomy Level	Course Outcome
1	Analyze and compare the Algorithms that help us to arrange the elements in the sorted order.	3	CO5

ASSIGNTMET-2

Bhargavi Naini
15BD1A1207

DS Assignment AY: 2016-17

Sorting: Sorting means arranging a set of data in some order. There are different methods that are used to sort the data in ascending or descending order.

Selection sort:

This is the simplest method of sorting. To sort the data in ascending order, the 0th element is compared with all the elements. If the 0th element is found to be greater than the compared element then they are interchanged. So after the first iteration the smallest element is placed at 0th position. The same procedure is repeated for next elements and so on.

Insertion Sort:

Insertion sort is implemented by insertion of particular element at the appropriate position. In the method 1st iteration starts with comparision of 1st element with the 0th element.

In 2nd iteration, 2nd element is compared with 0th and 1st element.

During comparision it is found that the element in question can be inserted at a suitable position then space is created for it by shifting the element one position to right and inserting element at the suitable position.

Radix Sort :

Radix Sort is a non-comparitive integer sorting algorithm that sorts data with integer keys by grasping keys by the individual digits which share the same significant position and value.

Quick Sort :

Quick sort is also known as Partition exchange sort. In general quick sort can sort a list of data elements significantly faster than any of the common sorting algorithms. The basic strategy of quick sort is to divide and conquer. To split the element is

selected which is known as Pivot element.

Heap sort:

Heap is a type of binary tree. when root of any subtree is less than or equal to its children it is less/than or/e called min-heap, max heap is vice-versa.

Heap sort is bascially an improvement over the binary tree sort. It builds the heap by adjusting the position of elements within the array itself.

AY : 2016-17

ASSIGNMENT-2

Hirdesh Arora
15BDIA1221
hirdesh

Sorting

Sorting is ordering a list of objects. We can distinguish two types of sorting. If the number of objects is small enough to fits into the main memory, sorting is called internal sorting. If the number of objects is so large that some of them reside on external storage during the sort, it is called external sorting. In this chapter we consider the following internal sorting algorithms

Insertion sort

Selection sort

Heapsort

Quicksort

Radixsort

Selection sort

Class Sorting algorithm

Data structure Array

Worst case performance O(n2)

Best case performance O(n2)

Average case performance O(n2)

Worst case space complexity O(n) total, O(1) auxiliary

In computer science, selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

Insertion sort

Example of insertion sort sorting a list of random numbers

Graphical illustration of insertion sort

Class Sorting algorithm

Data structure Array

Worst case performance O (n2) comparisons, swaps

Best case performance O(n) comparisons, O(1) swaps

Average case performance O (n2) comparisons, swaps

Worst case space complexity O (n) total, O(1) auxiliary

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages:

Simple implementation: Bentley shows a three-line C version, and a five-line optimized version

Efficient for (quite) small data sets, much like other quadratic sorting algorithms

More efficient in practice than most other simple quadratic (i.e., O(n2)) algorithms such as selection sort or bubble sort

Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is O(nk) when each element in the input is no more than k places away from its sorted position

Stable; i.e., does not change the relative order of elements with equal keys

In-place; i.e., only requires a constant amount O(1) of additional memory space

Online; i.e., can sort a list as it receives it

Radix sort

Class Sorting algorithm

Data structure Array

Worst case performance O(wn)

Worst case space complexity O(w + N)

In computer science, radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines.

Most digital computers internally represent all of their data as electronic representations of binary numbers, so processing the digits of integer representations by groups of binary digit representations is most convenient. Two classifications of radix sorts are least significant digit (LSD) radix sorts and most significant digit (MSD) radix sorts. LSD radix sorts process the integer representations starting from the least digit and move towards the most significant digit. MSD radix sorts work the other way around.

LSD radix sorts typically use the following sorting order: short keys come before longer keys, and keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as "b, c, d, e, f, g, h, i, j, ba" would be lexicographically sorted as "b, ba, c, d, e, f, g, h, i, j". If lexicographic ordering is used to sort variable-length integer representations, then the representations of the numbers from 1 to 10 would be output as 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, as if the shorter keys were left-justified and padded on the right with blank characters to make the shorter keys as long as the longest key for the purpose of determining sorted order.

Quick sort

Class Sorting algorithm

Worst case performance O(n2)

Best case performance O(n log n) (simple partition)

or O(n) (three-way partition and equal keys)

Average case performance O(n log n)

Worst case space complexity O(n) auxiliary (naive)

O(log n) auxiliary (Sedgewick 1978)

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Developed by Tony Hoare in 1959, with his work published in 1961, it is still a commonly used algorithm for sorting. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort.

Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

Mathematical analysis of quicksort shows that, on average, the algorithm takes O(n log n) comparisons to sort n items. In the worst case, it makes O(n2) comparisons, though this behavior is rare.

Heap sort

Class Sorting algorithm

Data structure Array

Worst case performance {\displaystyle O(n\log n)} O(n\log n)

Best case performance {\displaystyle \Omega (n),O(n\log n)} \Omega(n), O(n\log n)[1]

Average case performance {\displaystyle O(n\log n)} O(n\log n)

Worst case space complexity {\displaystyle O(1)} O(1) auxiliary

A run of the heapsort algorithm sorting an array of randomly permuted values. In the first stage of the algorithm the array elements are reordered to satisfy the heap property. Before the actual sorting takes place, the heap tree structure is shown briefly for illustration.

In computer science, heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum.

Although somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favorable worst-case O(n log n) runtime. Heapsort is an in-place algorithm, but it is not a stable sort.

Heapsort was invented by J. W. J. Williams in 1964. This was also the birth of the heap, presented already by Williams as a useful data structure in its own right. In the same year, R.

W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the treesort algorithm.

		Time Complexity					
		Best	Worst	Avg.	Space	Stable	Comments
Comparison Sort	Bubble Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*For each pair of indices, swap the elements if they are out of order*
	Modified Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*At each Pass check if the Array is already sorted.Best Case-Array Already sorted*
	Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*Swap happens only when once in a Single pass*
	Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*Very small constant factor even if the complexity is O(n^2).* *Best Case: Array already sorted* *Worst Case: sorted in reverse order*
	Quick Sort	O(n.lg(n))	O(n^2)	O(n.lg(n))	O(1)	Yes	*Best Case: whenpivote divide in 2 equal halves* *Worst Case: Array already sorted - 1/n-1 partition*
	Randomized Quick Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	Yes	*Pivot choosen randomly*
	Merge Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(n)	Yes	*Best to sort linked list (constant extra space).* *Best for very large number of elements which cannot fit in memory (External sorting)*
	Heap Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	No	
Non-Comparison Sort	Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(n+2^k)	Yes	*k = Range of Numbers in the list*
	Radix Sort	O(n.k/s)	O(2^s.n.k/s)	O(n.k/s)	O(n)	No	
	Bucket Sort	O(n.k)	O(n^2.k)	O(n.k)	O(n.k)	Yes	

EXPT. No. _____

Date : _____

* Selection Sort :

Scan all the items and find the smallest. Swap it into position as the first item. Repeat the seclection sort on remaining N-1 items. I found this the most intuitive and easiest to implement - you always iterate forwoard, and swap with the smallest element.

* Insertion Sort :

Start with a sorted list of elements on the left, and N-1 unsorted items on the right. Take the first unsorted item and insert it into the sorted list, moving elements as necessary. We now have a sorted list of size 2, and N-2 unsorted elements. Repeat for all elements.

Like Bubble sort, I found this counter-intuitive because you step backwoards.

This is a little like bubble sort for moving items, except when you encounter an item smaller than you, you stop. If the data is reverse sorted, each item must travel to the head of the list. and this becomes bubble sort.

There are various ways to move the item leftwards - you can do a swap on each iteration, or copy each item over its neighbour.

* HeapSort :

Add all items into a heap. Pop the largest item from the heap and insert it at the end. Repeat for all the items.

Heapsort is just like selection sort, but with a better way to get the largest element. Instead of scanning all the items to find the max, it pulls it from a heap. Heaps have properties that allow heapsort to work in place, without additional memory.

Creating the heap is $O(N \lg N)$. Popping items is $O(1)$, and fixing the heap after Pop is $\lg N$. There are N pops, so there is another

EXPT. No. _____

Date : _____

$O(N \lg N)$ factor, which is $O(N \lg N)$ overall.
Heap sort has $O(N \lg N)$ behaviour, even in the worst case, making it good for real time applications.

* Radix Sort:

Get a series of numbers, and sort them one digit at a time.
Repeat the sorting on each set of digits.
Radix sort uses counting sort for efficient $O(N)$ sorting of digits
Actually, radix sort goes from least significant digit to most significant.
Radix and counting sort are fast, but require structured data external memory and donot have the caching benefits of quicksort.

* Quick Sort:

There are many versions of quick sort, which is one of the most popular sorting method due to its speed.

- Using external memory:

Pick a pivot item.
Partition the other items by adding them to a less than pivot sublist or greater than pivot sublist.
The pivot goes between the two lists.
Repeat the quicksort on the sublists, until you get to a sublist of size 1.
Combine the lists - the entire list will be sorted.

- Using - in place memory w/two pointers:

Pick a pivot and swap it out of the way.
Going left to right, find an odd ball item that is greater than pivot
Going right to left, find an odd ball item that is less than pivot
Swap the items if found, and keep going until the pointer cross

EXPT. No. _____

Date : _____

reinsert the pivot.
Quicksort the left and right partitions.

Keshav memorial institute of technology

DEPARTMENT OF INFORMATION TECHNOLOGY

Assignment 1

II B Tech - I Semester

Branch: IT | Acad. Year : 2015-2016

Subject: DATA STRUCTURES | Max Marks: **5M**

---

1) Explain about Time complexity and Space complexity ? Explain about different Asymptotic notations(Big O,Omega,Theta).Give Examples to each.

2) Explain the procedure and Write the C Program for Concatenating two single linked lists?

3) Write the stack ADT? Write the steps for converting expression from infix to postfix with example?

Data Structures ASSIGNMENT-1

1. Explain about Time Complexity and Space complexity. Explain about different Asymptotic notations. Give examples to each.

Ans: Time and Space Complexity :

Analysing an algorithm means determining the amount of resources needed to execute it. Algorithms are generally designed to work with an arbitrary number of inputs, so the efficiency or complexity of an algorithm is stated in terms of time and space complexity.

The time complexity of an algorithm is basically the running time of a program as a function of the input size. Similarly, the space complexity of an algorithm is the amount of computer memory that is required during the program execution as a function of input size.

In other words, the number of machine instructions which a program executes is called its time complexity. This number is primarily dependent on the size of the program's input and the algorithm used.

* Generally, the space needed by a program depends on the following two parts.

- Fixed part : It varies from problem to problem. It includes the space needed for storing instructions, constants, variables and structured variables (like arrays and structures).

- Variable part: It varies from program to program. It includes the space needed for recursion stack, and for structured variables that are allocated space dynamically during the runtime of a program.

Worst-case running time: This denotes the behaviour of an algorithm with respect to the worst possible case of the input instance. The worst-case running time of an algorithm is an upper bound on the running time for any input. Therefore, having the knowledge of worst-case running time gives us an assurance that the algorithm will never go beyond this time limit.

Average-case running time: The average-case running time of an algorithm is an estimate of the running time for an 'average' input. It specifies the expected behaviour of the algorithm when the input is randomly drawn from a given distribution. Average-case running time assumes that all inputs of a given size are equally likely.

Best-Case running time: The term 'best-case performance' is used to analyse the algorithm under optimal conditions. For example, the best case for a simple linear search on an array occurs when the desired element is the first in the list. However, while developing and choosing an algorithm to solve a problem, we hardly base our decision on the best-case performance. It is always recommended to improve the average performance and the worst-case performance of an algorithm.

Asymptotic Notations:

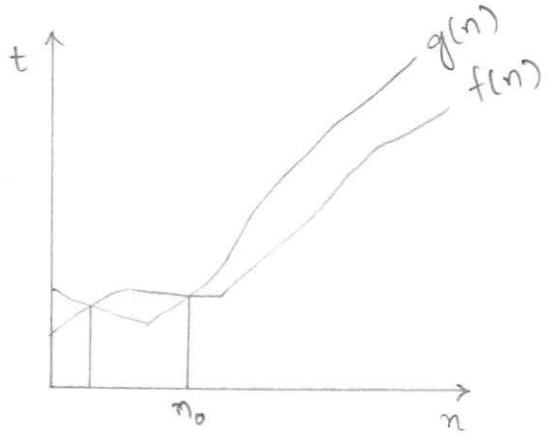
These notations are used to determine lower (best), upper (worst) bounds of a function $f(n)$.

There are 3 major notations.

1. Big O Notation (O) $\Rightarrow$ bounds $f(n)$ with its upper bound.
2. Omega Notation (Ω) $\Rightarrow$ bounds $f(n)$ with its lower bound.
3. Theta Notation (Θ) $\Rightarrow$ bounds $f(n)$ with its lower and upper bounds.

Big O (O):

Def: We say $f(n) = O(g(n))$ if and only if for $f(n) \leq c \cdot g(n)$ where c is a constant and $c > 0$ and $n \geq n_0$ where $n_0 \geq 1$.



$n \rightarrow$ input size

$t \rightarrow$ time

Eg. $f(n) = 3n + 5$

$3n + 5 = O(n)$

$f(n) \quad g(n)$

$f(n) \leq c \cdot g(n)$

$3n+5 \leq c \cdot n$ $\quad$ $\boxed{c=4}$

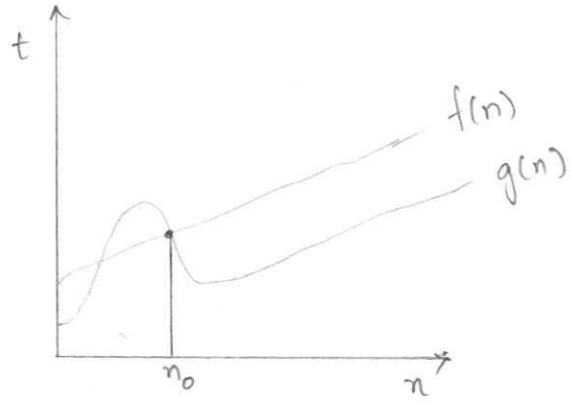
$3n+5 \leq 4n$ $\quad$ $\boxed{n_0 \geq 2}$

$f(n) = O(n)$

$\therefore$ $3n+5 = O(n)$ is true for $3n+5 \leq c \cdot n$ where $c=4$ and $n_0 \geq 2$.

Omega (Ω) :

Def: We say $f(n) = \Omega(g(n))$ if and only if for $f(n) \geq c \cdot g(n)$ where $c > 0$ and $n \geq n_0$ and $n_0 \geq 1$.



$n \rightarrow$ input size

$t \rightarrow$ time

Eg. $f(n) = 3n+5$

$3n+5 = \Omega(n)$

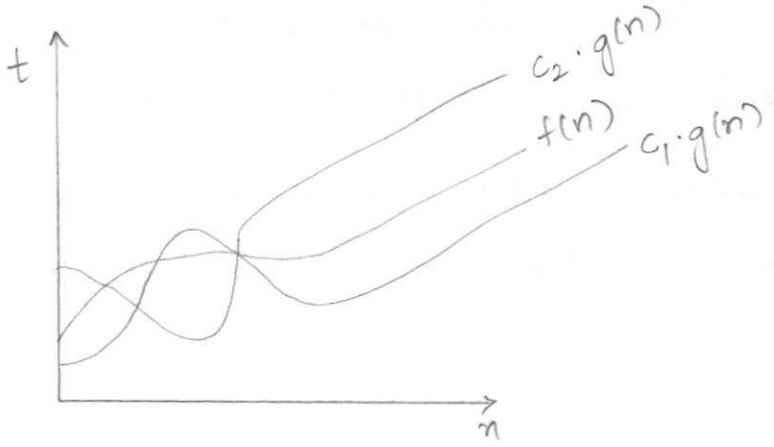
$3n \leq 3n+5$, $n \geq 0$

$f(n) = \Omega(n)$, $c=3$

$\therefore$ $3n+5 = \Omega(n)$ is true for $3n+5 \geq c \cdot n$ where $c=3$ & $n \geq 0$.

Theta (Θ):

Def: We say $f(n) = \Theta(g(n))$ is true for $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ where $c_1, c_2 > 0$ and $n > n_0$, $n_0 \geq 1$.



$n \rightarrow$ input size

$t \rightarrow$ time

Eg. $f(n) = 3n+5$

Big O $3n+5 \leq 4n, \quad n \geq 5$

Omega $3n \leq 3n+5, \quad n \geq 0$

$3n \leq 3n+5 \leq 4n, \quad n \geq 5$

$c_1 = 3$

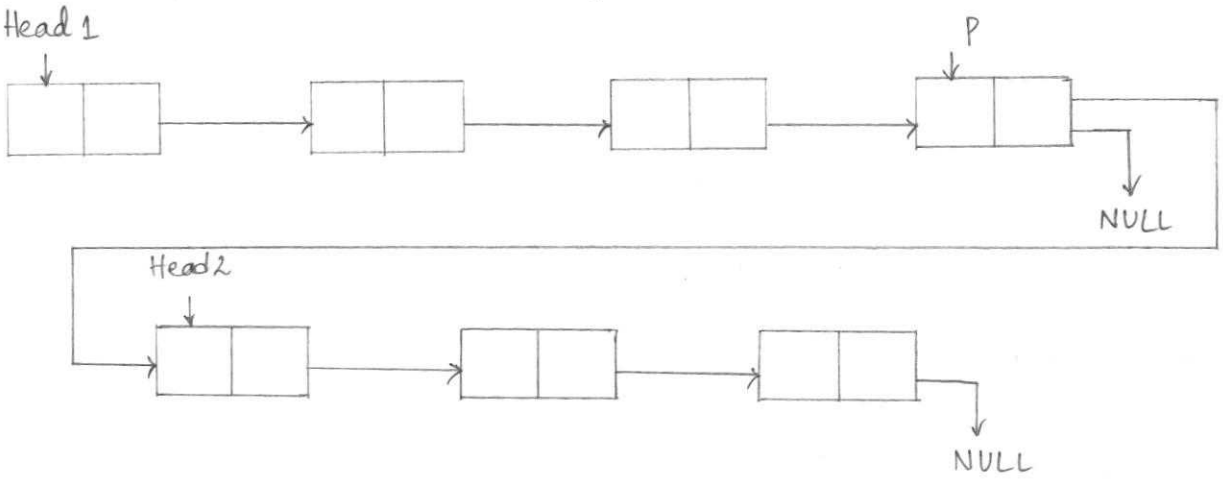
$c_2 = 4.$

2. Explain the procedure and write the C program for concatenating two single linked lists.

Ans: Algorithm for concatenation:

Let us assume that the two linked lists are referenced by head1 and head2 respectively.

1. If the first linked list is empty then return head2.
2. If the second linked list is empty then return head1.
3. Store the address of the starting node of the first linked list in a pointer variable, say p.
4. Move the p to the last node of the linked list through simple linked list traversal technique.
5. Store the address of the first node of the second list in the next field of the node pointed by p. Return head1.



C program for concatenating two single linked lists:

```
# include <stdio.h>
# include <stdlib.h>
struct node * create (struct node * head);
struct node * concat (struct node * list1, struct node * list2);
void display (struct node * list1);
struct node
{
int data;
struct node * next;
};
```

```
struct node *head, *list1, *list2, *new, *temp;
void main( )
{
head = NULL;
list1 = NULL;
list2 = NULL;
printf("create list1 : ");
list1 = create(list2);
printf("list1 is : ");
display(list1);
printf("list2 is : ");
display(list2);
printf("concatenation of list1, list2 is ");
list1 = concat(list1, list2);
display(list1);
}
struct node *create(struct node *head)
{
int ele, ch;
while(ch)
{
printf("enter the element");
scanf("%d", &ele);
new=(struct node *) malloc(sizeof(struct node));
new->data = ele;
new->next = NULL;
```

```
if (head == NULL)
{
head = new;
}
else
{
temp = head;
temp -> next = new;
temp = temp -> next ;
printf (" Do you want to create another node enter 1 else 0");
scanf (" %d", &ch);
}
return head;
}
}
struct node * concat (struct node * list1, struct node * list2) {
{
if (list1 == NULL)
{
return list2;
}
else if (list2 == NULL)
{
return list1;
}
else
{
```

```
temp = list1 ;
while (temp -> next! = NULL)
{
temp = temp -> next;
}
temp -> next = list2;
}
return list1;
}
void display (struct node * head)
{
temp = = head;
if (head == = NULL)
{
printf("list is empty");
}
else
{
while (temp ! = NULL)
{
printf ("%d -> ", temp -> data);
temp = temp -> next;
}
}
}
```

3. Write the stack ADT? Write the steps for converting expression from infix to postfix with example?

Ans: Stack ADT:

A stack is an abstract data type that serves as a collection of elements, with two principal operations: push, adds an element to the collection, and pop, which removes the most recently added element that was not yet removed. The order in which elements come off a stack gives rise to its alternative name, LIFO (last in, first out). Additionally, a peek operation may give access to the top without modifying the stack.

The ~~na~~ name "stack" for this type of structure comes from the analogy to a set of physical items stacked on top of each other, which makes it easy to take an item off the top of the stack, while getting to an item deeper in the stack may require taking off multiple other items first.

A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack.

Steps for converting an expression from infix to postfix:

1. If the incoming token is an operand directly write the operand to the postfix expression.

2. If the incoming token is an operator.

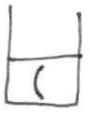
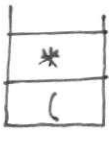
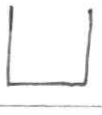
* If stack is empty push the operator into the stack else compare the priority of operator with the already available operators in the stack.

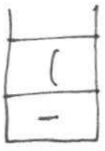
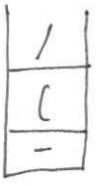
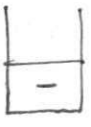
If the incoming token has highest priority then we directly push the operator into the stack. If the incoming taken have equal or less priority then operators in the stack must be popped out and return to the postfix expression.

3. If the incoming taken is (, push directly into the stack. If the incoming token is), we pop out all the operators in the stack till we reach the immediate opening into the stack.

4. After reaching the end of expression all the operators in the stack must be popped out and return back to postfix expression.

Eg. (a * b) - (c/D)

Token	Stack	Postfix
(		
a		a
*		
b		ab *
)		

−		ab *
(		
c		ab * c
/		
D		ab * cD/
)		ab * cD/−

Keshav memorial institute of technology

DEPARTMENT OF INFORMATION TECHNOLOGY

Assignment 2

II B Tech - I Semester

Branch: IT | Acad. Year : 2015-2016

Subject: DATA STRUCTURES | Max Marks: **5M**

---

1)Define the following

Hash Table, Hash Functions and Collision resolution techniques.

2 Write c program for heap sorting method

3 Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45. Insert into B-Tree of order 3

Assignment II :

(1) Define the following : Hash table, Hash functions and collision resolution techniques.

Ans: Hash table: It is a data structure that is used to store dictionary pairs. It is of fixed size. The 'search' operation is applied to a part of the dictionary pair called the Key. The size of the hash table is represented by a variable called the table size which ranges from 0 to (table size-1)

Hash Function: Hash function takes this key as an input and produces an Hash key which will be used as an index in to the hash table to place the corresponding record of the given key at that location.

There are few commonly used Hash functions:

(1) Division Method

(2) Mid Square Method

(3) Multiplication Method

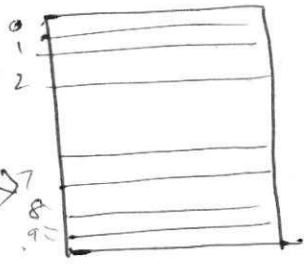
(4) Folding Method

(1) Division Method: In this method Key is divided by the Hash table size and that will be Hash Key which will pointer in Hash table.

eg: Let HT size = 10

Key = 47 $\quad 47 \% 10 = 7$

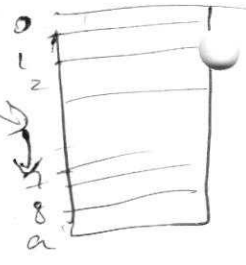
Key = 32 $\quad 32 \% 10 = 2$.



(2) Mid square Method: In this method given Key is first squared and then mid number is used as Hash Key.

eg Key = 3036

$HF(key) = (3036)^2 = 9217296$



if Table Hash table size is 10.

(3) Multiplicative Method:

Hash Key = HF (Key) = floor(A * (Key * γ))

where,

r is a real number

prefered value for r = 0.61803

'A' can be any +ve integer eg = 5

(r & A are same for all).

eg: Key = 25

Hash Key = floor (5 * (25 * 0.6183))

= floor (5 * 15.4575)

= floor (77.255)

= 77

(4) Folding Method: simply fold & sum up.

eg: Key = 324713

Let HT size is 1000

Hash Key = 324 | 713 | 8.

=> 324
713
8
1045

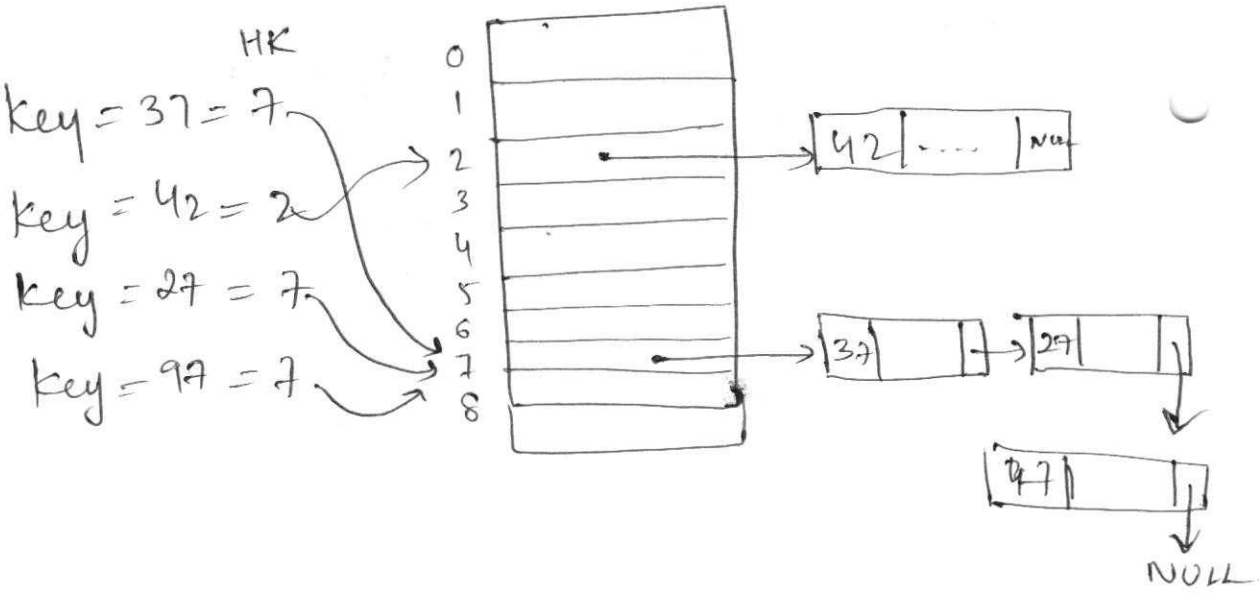
=> 1045
+5
1050 → Index

Collision resolution techniques:-

If the hash function produces same hash key for 2 different key values then it is called collision. To solve this problem we have collision resolution techniques.

1. chaining:

In this type of hashing, the hash table stores a set of pointers, where each pointer points to a separate linked list. For each element k, hash function HF(k) is applied, which gives the hash code (index) of a particular bucket (cell) in hash table.



2. Linear probing

method = Division method

Key - 37 = 7 (HK)

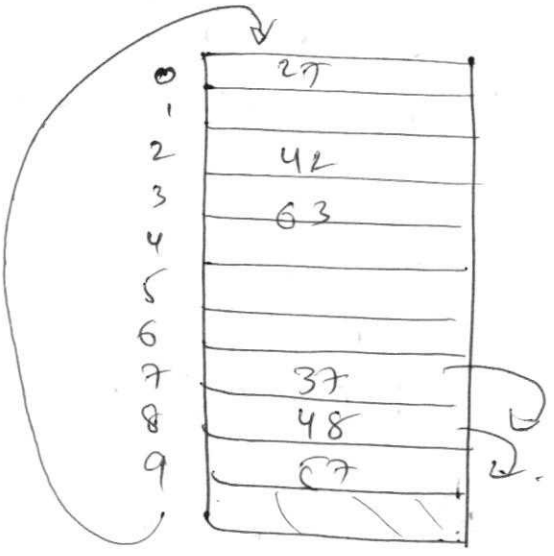
42 → 2

48 → 8

57 - 7

27 - 7.

63 → 3



If collision occurs, place the key in empty cell after the occupied Hash key in circular manner.

3. Quadratic probing:

If collision occurs for given key then

Hashkey = $(Key + x^2)$ % HT SIZE

where, $x = 1, 2, 3 - - -$

4. Double Hashing:

HF1 (key) = hashkey (use Division method)

HF_2 (key) = M - hashkey = P

Choose M as the biggest prime no less

than size of the table.

Now jump forward by p locations from collision index & place the record there.

Ex: Key = 34

$HF_1(34) = 34 \% 10 = 4$ (SIZE = 10)

Choose M as 7

$HF_2(34) = M - 4$

$= 7 - 4$

$= 3$

Collision index = 4

p = 3

jump to 7th index & place the record.

(2) Write c program for heap sorting method.

Ans

```
#include<stdio.h>
#include<stdlib.h>
void makeheap (int A[10], int n)
{
```

```
    int i, val, j, parent;
    for(i=1; i<n; i++)
    {
        val = A[i];
        j=i;
        parent = (j-1)/2;
        while(j>0 && A[parent]<val)
        {
            A[j]= A[parent];
            j=parent;
            parent = (j-1)/2;
        }
        A[j]=val;
    }
}
void heapsort(int A[10], int n)
{
    int i, j, k, temp;
    for(i=n-1; i>0; i--)
    {
        temp = A[i];
        A[i] = A[0];
```

```
k = 0;
if (i == 1)
        j = -1;
else
        j = 1;
if (i > 2  && A[2] > A[1])
        j = 2;
while (j >= 0 && temp < A[j])
{
    A[k] = A[j];
    k = j;
    j = (2 * k) + 1;
    if (j+1 <= i-1 && A[j] < A[j+1])
        j++;
    if (j > i-1)  j = -1;
}
A[k] = temp;
}
```

```
Void main ( )
{
    int i, A[10], n=10;
    for (i=0; i<n; i++)
    {
      Scanf ("%d", &A[i]);
    }
    makeheap (A, n);
    heapsort (A, n);
    printf ("\n sorted array: ");
    for (i=0; i<n; i++)
    {
      printf ("    %d", A[i]);
    }
}
```

(3) Suppose we start with an empty Btree and keys arrive in the following order.
1, 12, 8, 2, 25, 5, 14, 28, 17, 7, 52, 16, 48, 68
3, 26, 29, 53, 55, 45. Insert into Btree of order 3

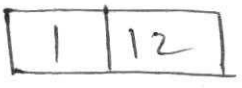
Ans

order = 3

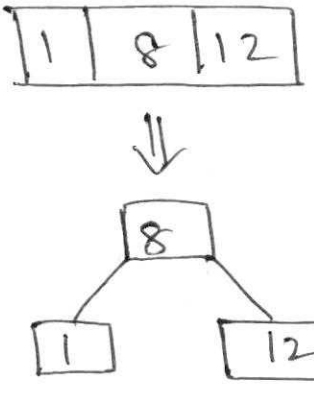
no: of possible keys = 2

Insert 1:

Insert 12:

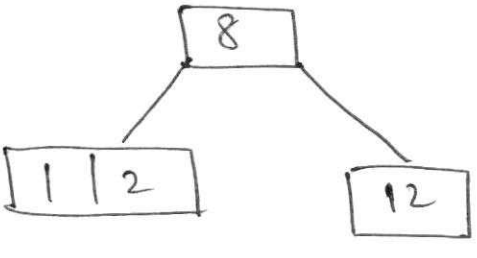


Insert 8:

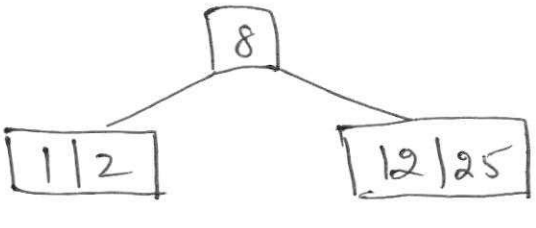


It is not possible
As no: of keys > 2.

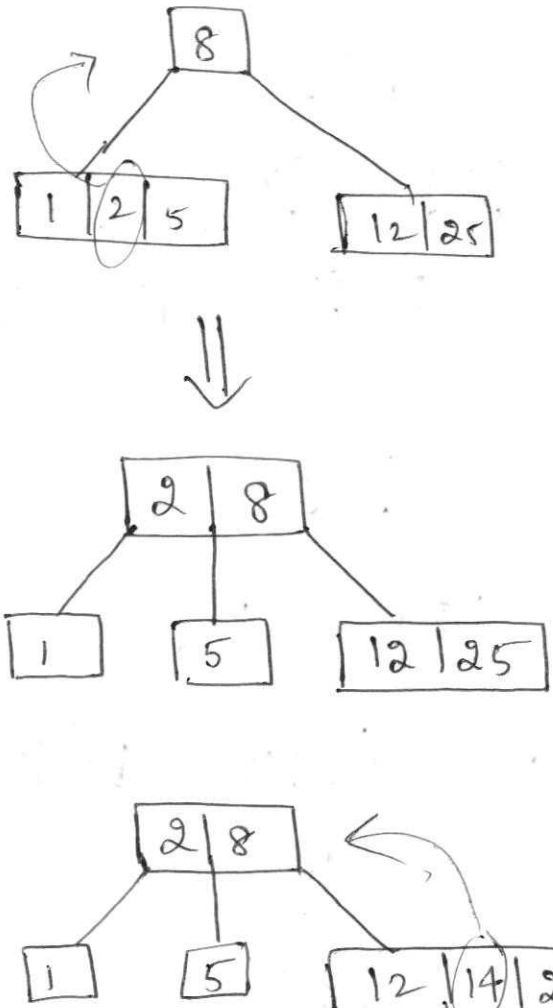
Insert 2:



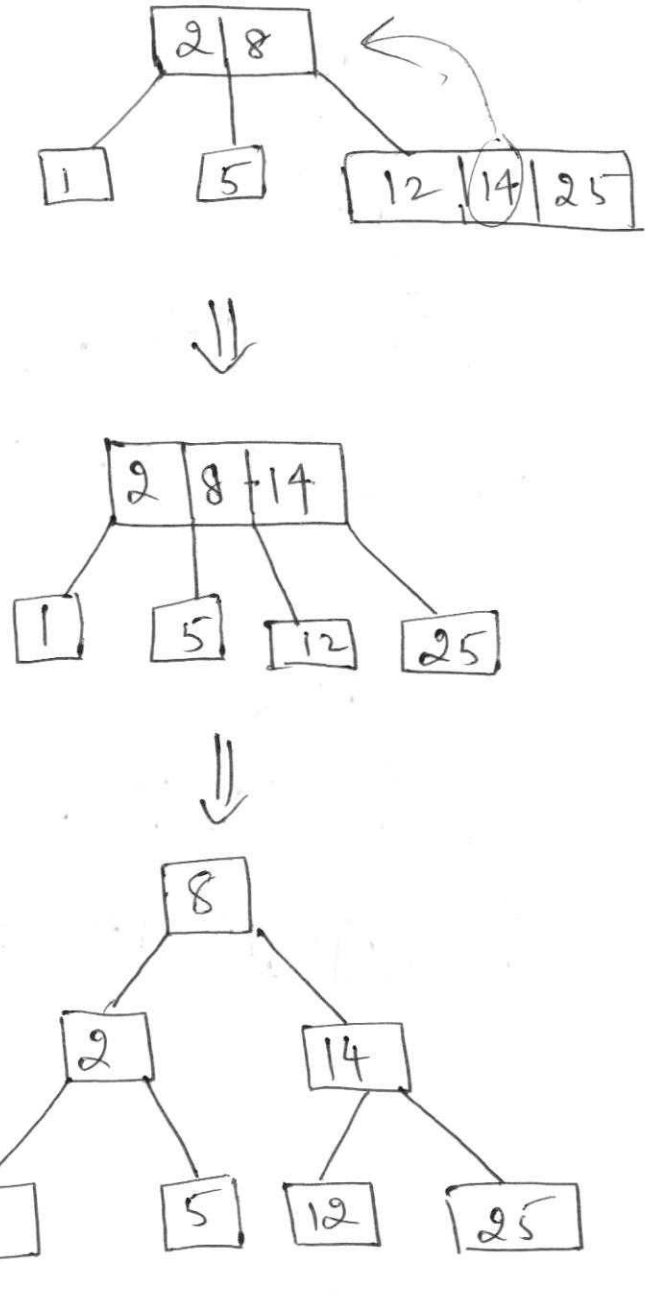
Insert 25:



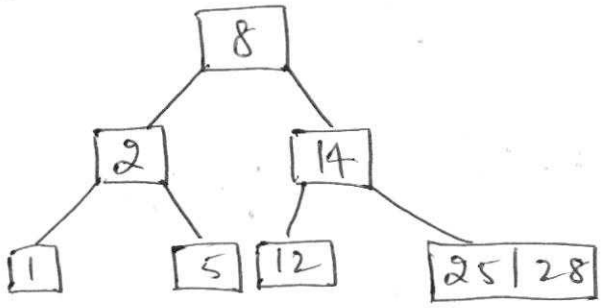
Insert 5 :-



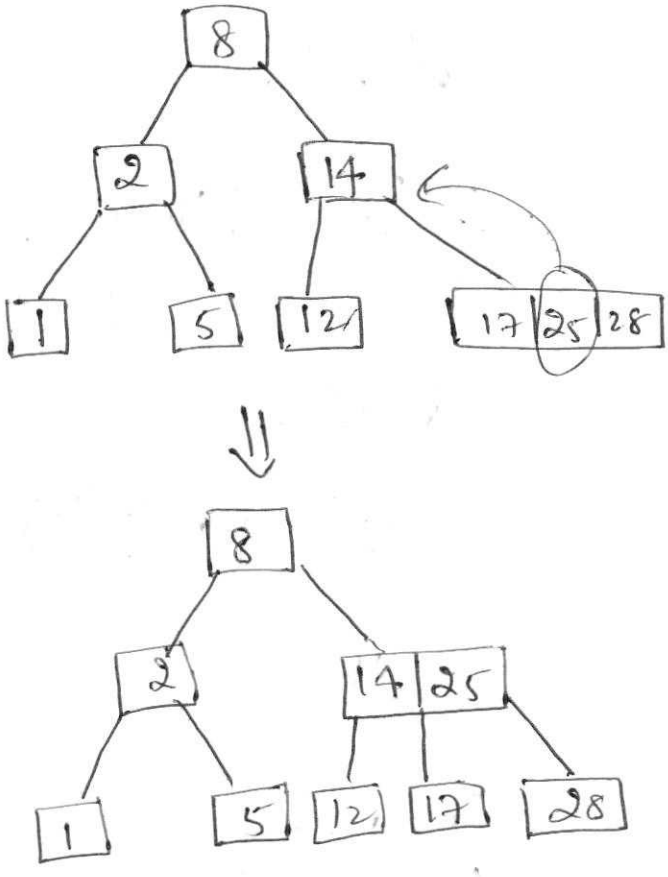
Insert 14 :-



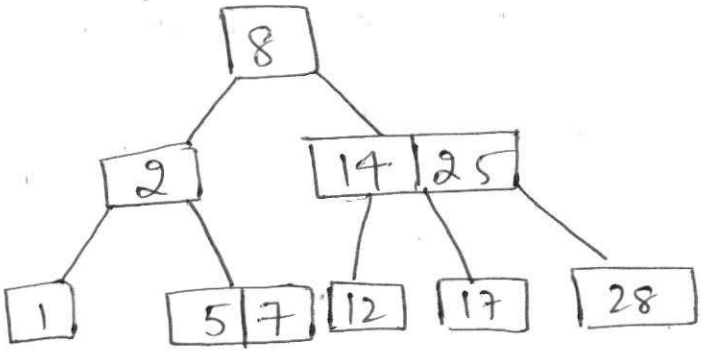
insert 28:-



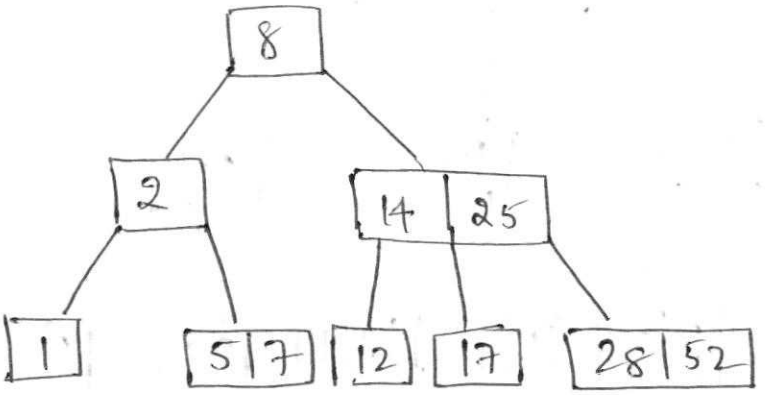
insert 17:-



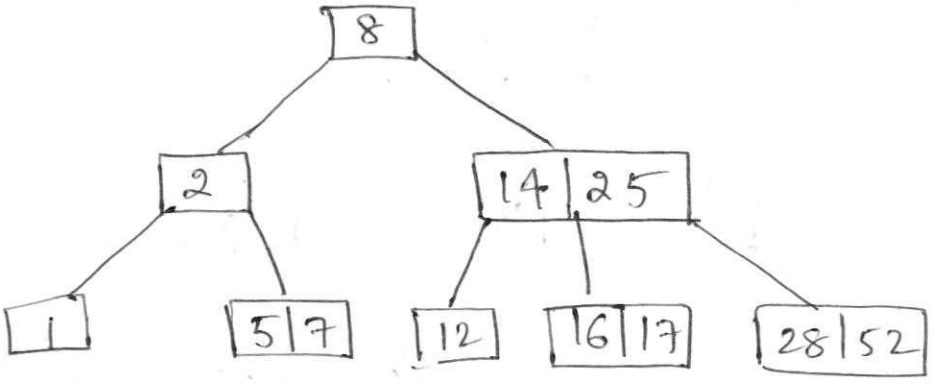
insert 7:



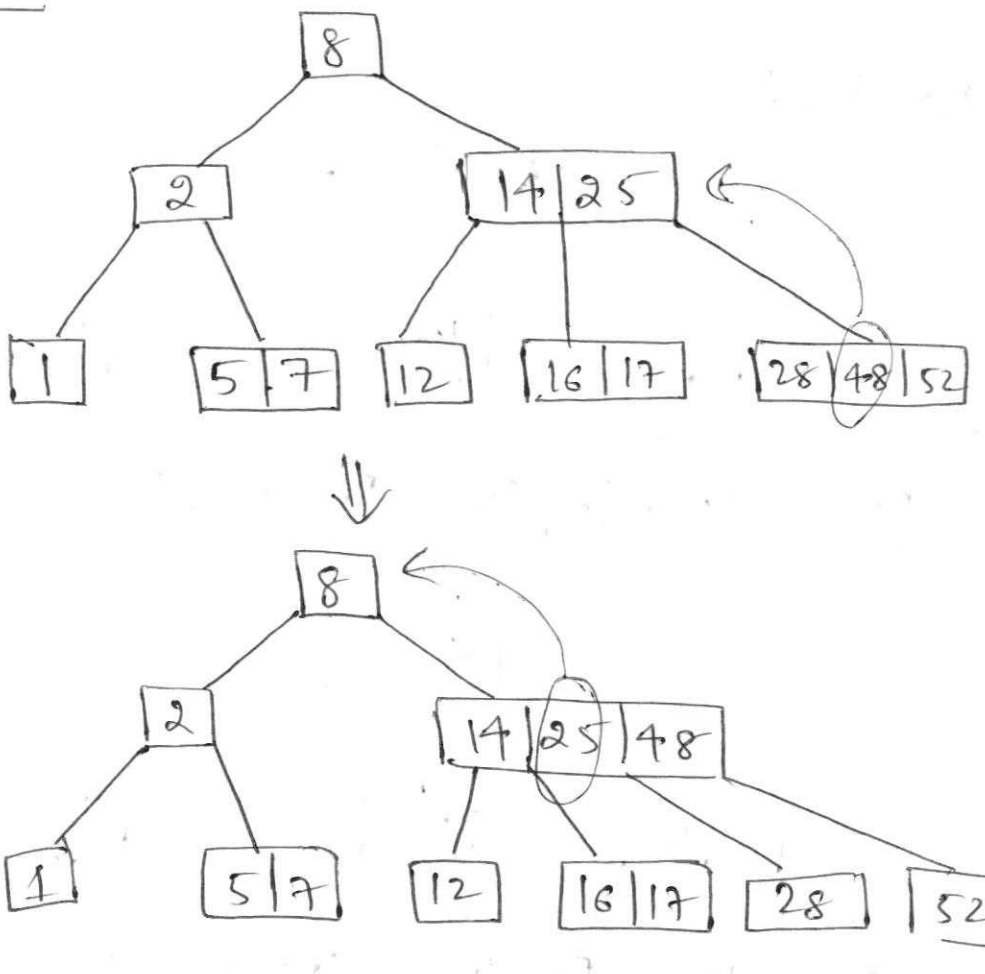
insert 52:-

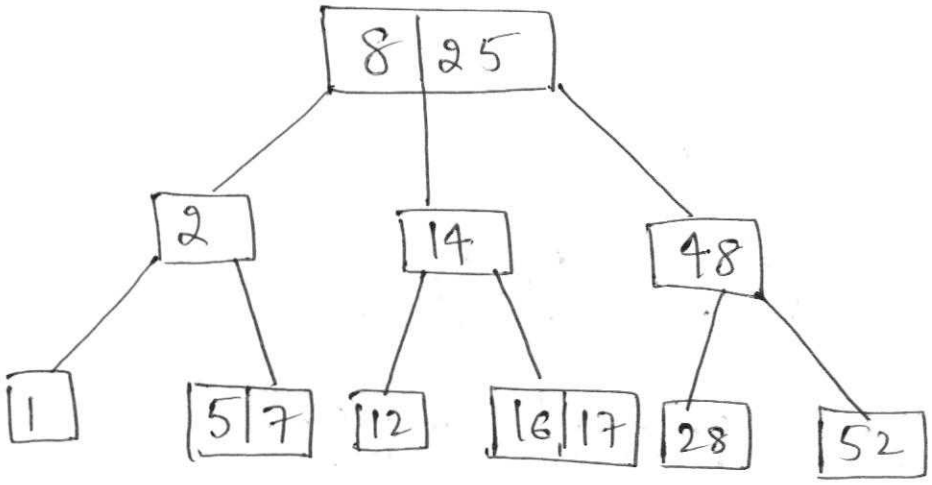


insert 16:-

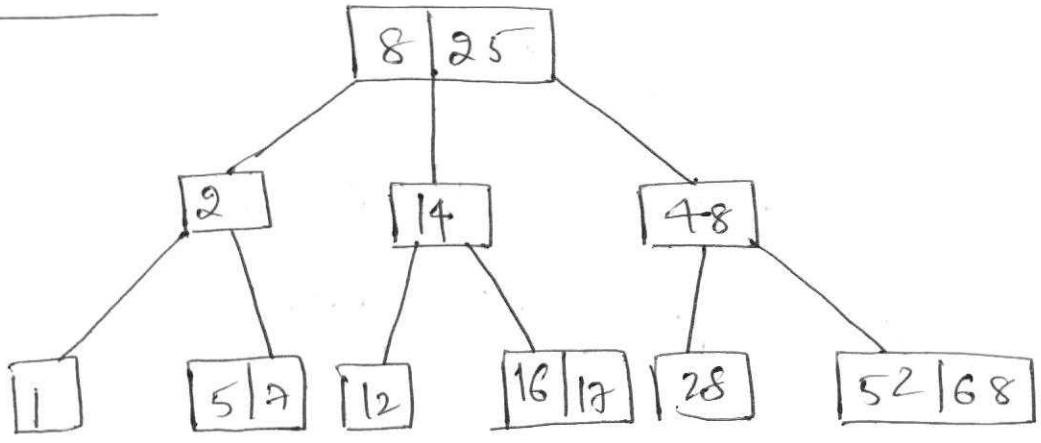


insert 48:-

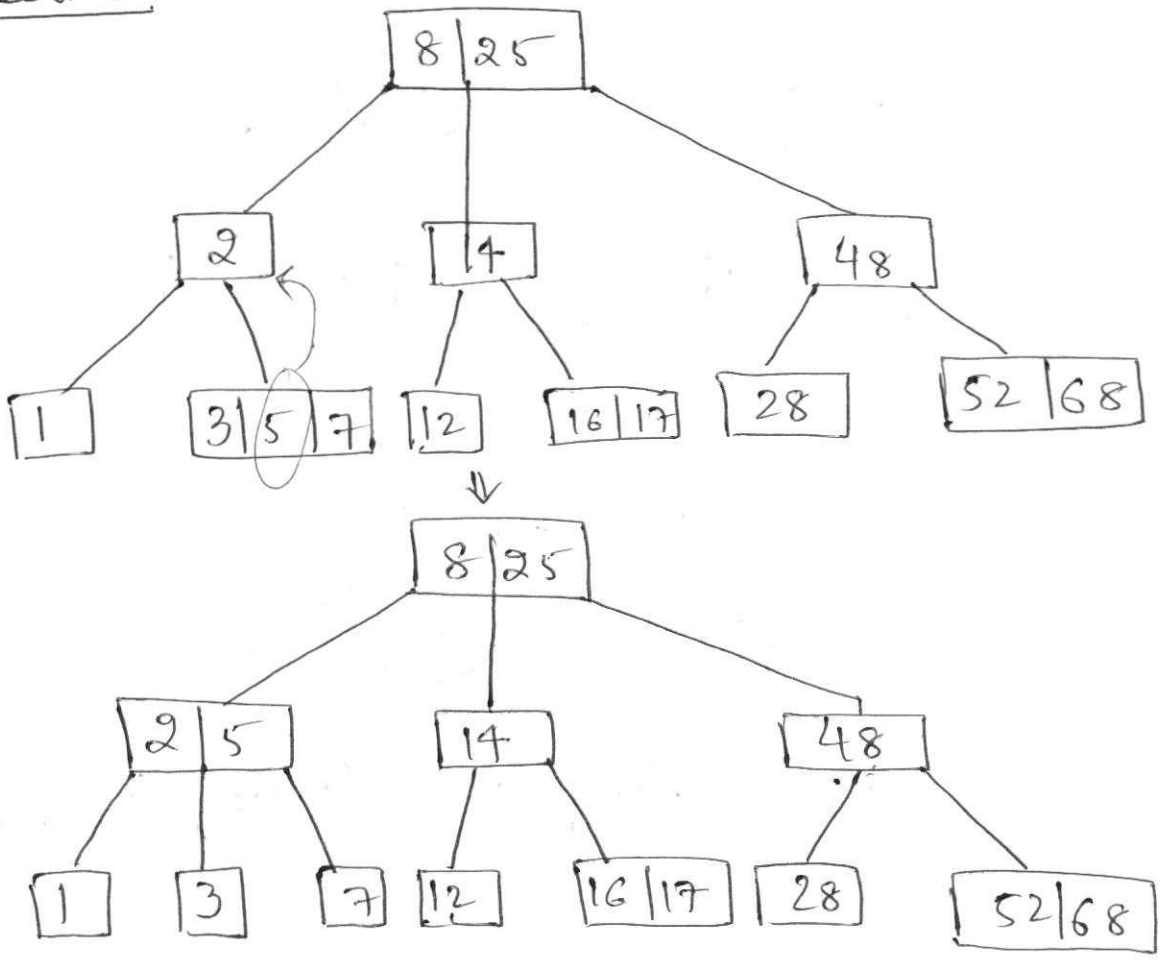




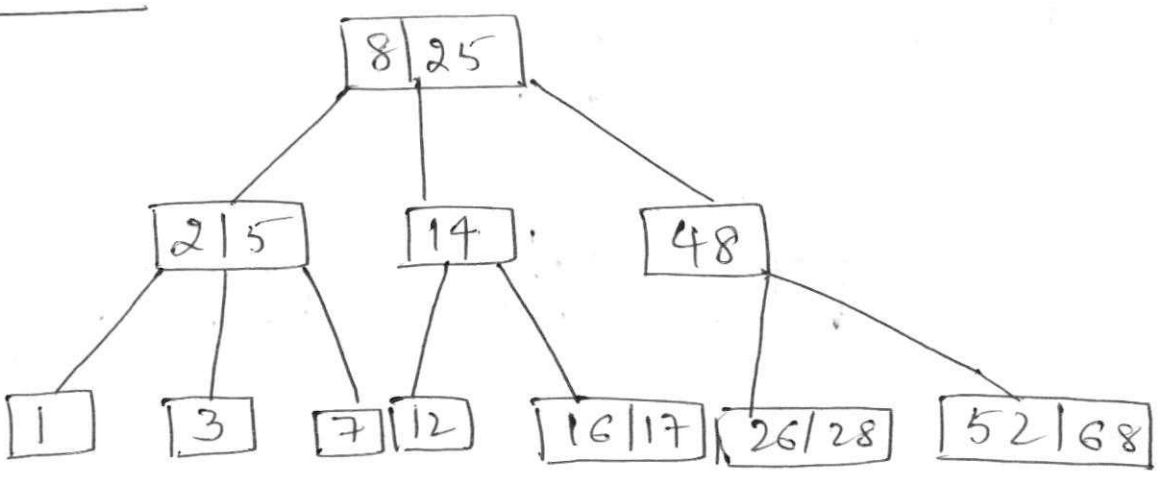
Insert 68 :-



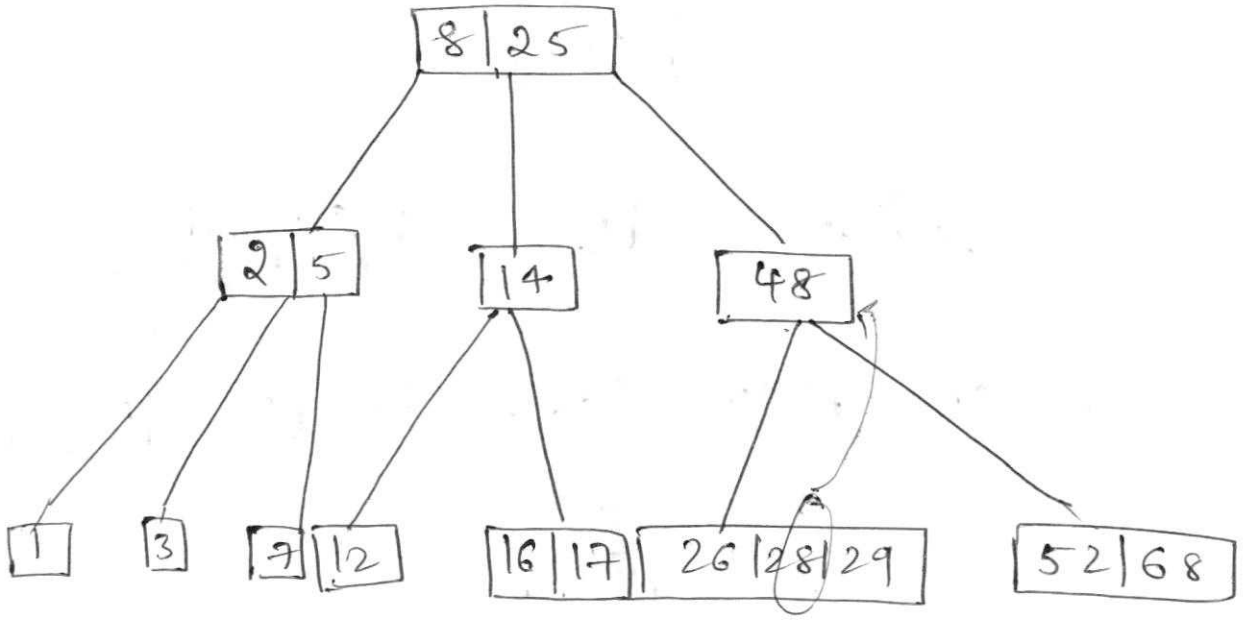
Insert 3 :-



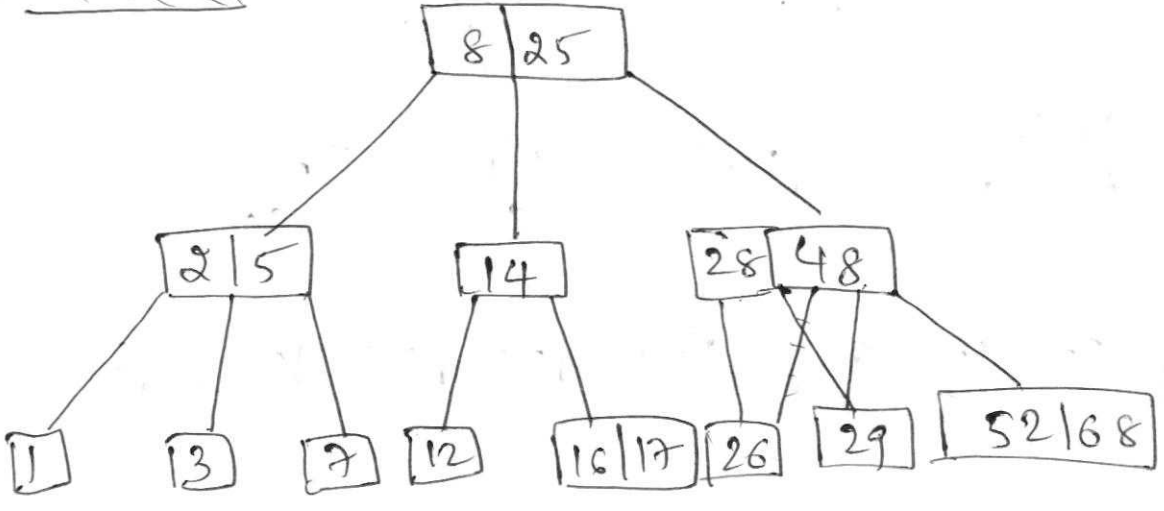
Insert 26:-



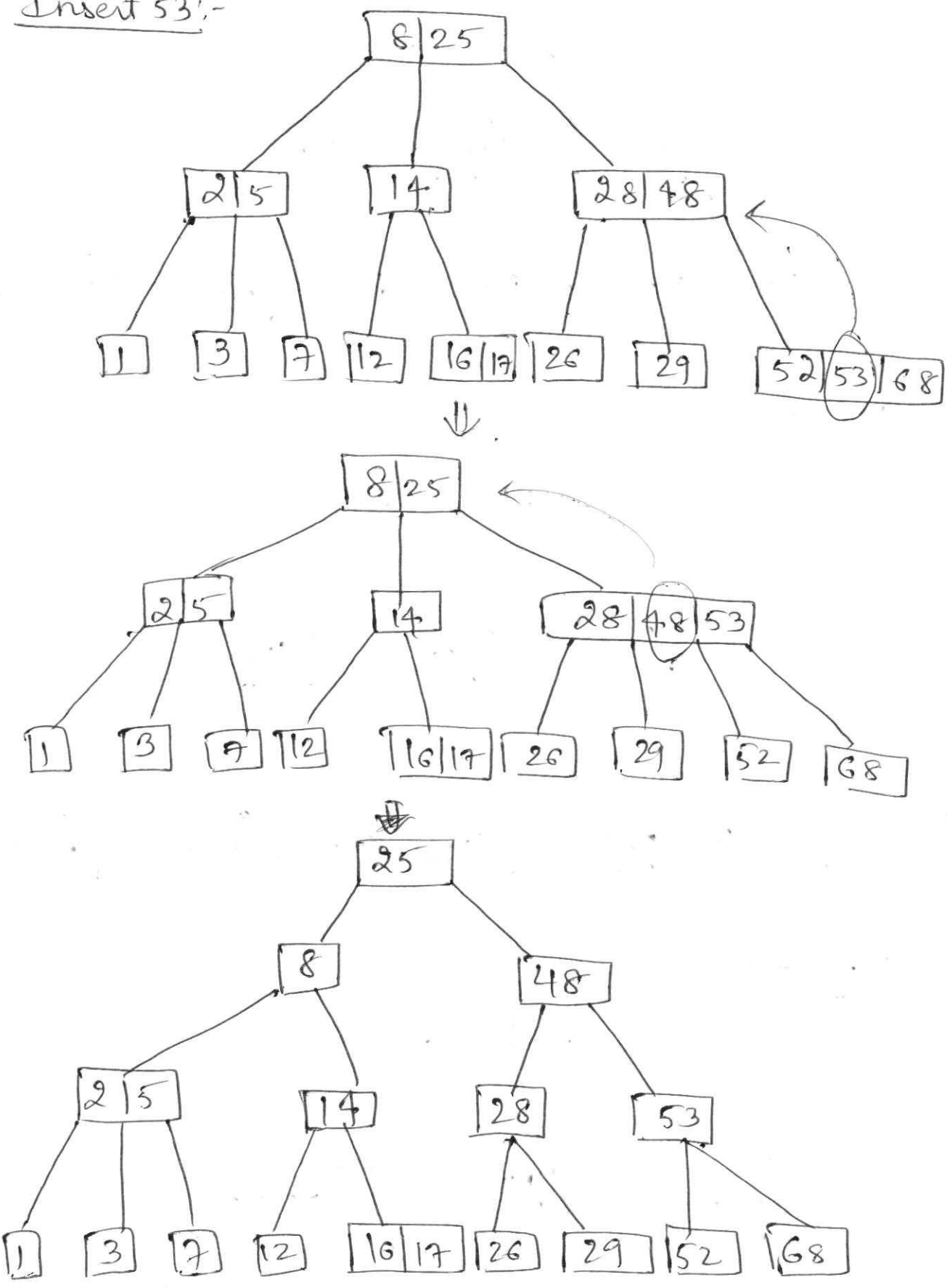
Insert 29:-



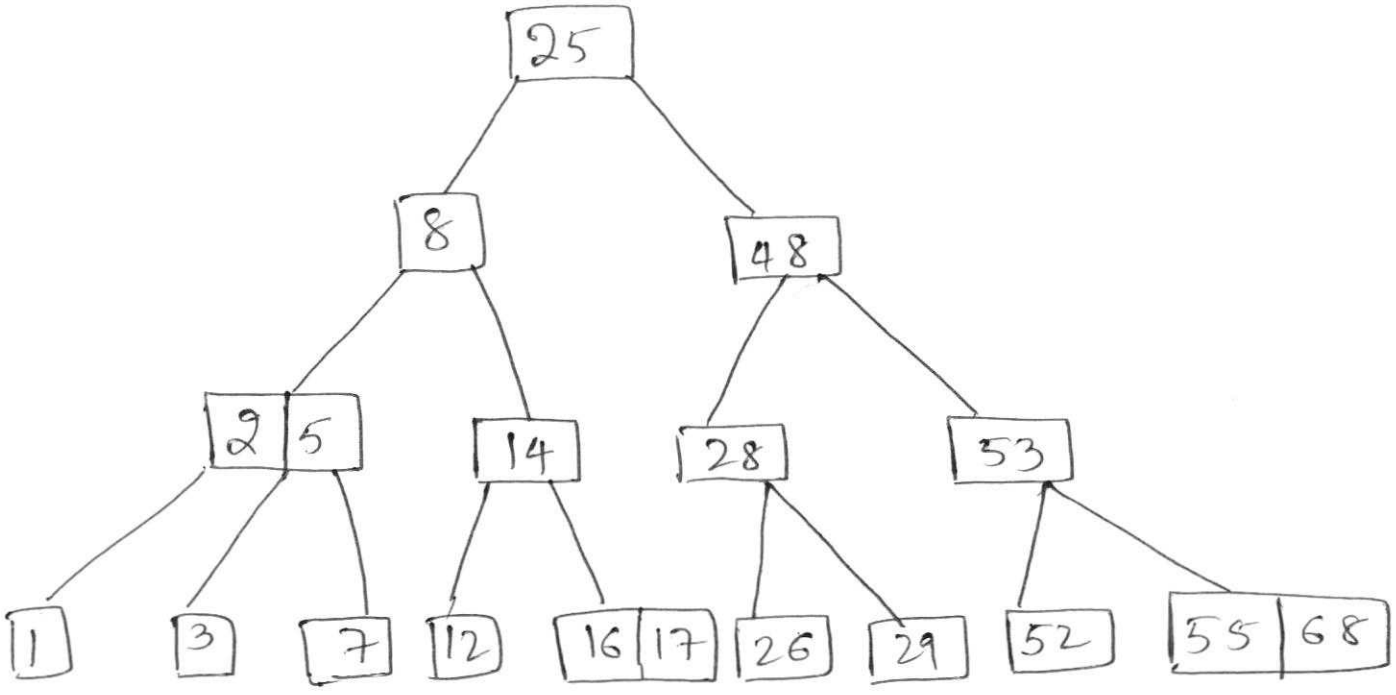
Insert 53:-



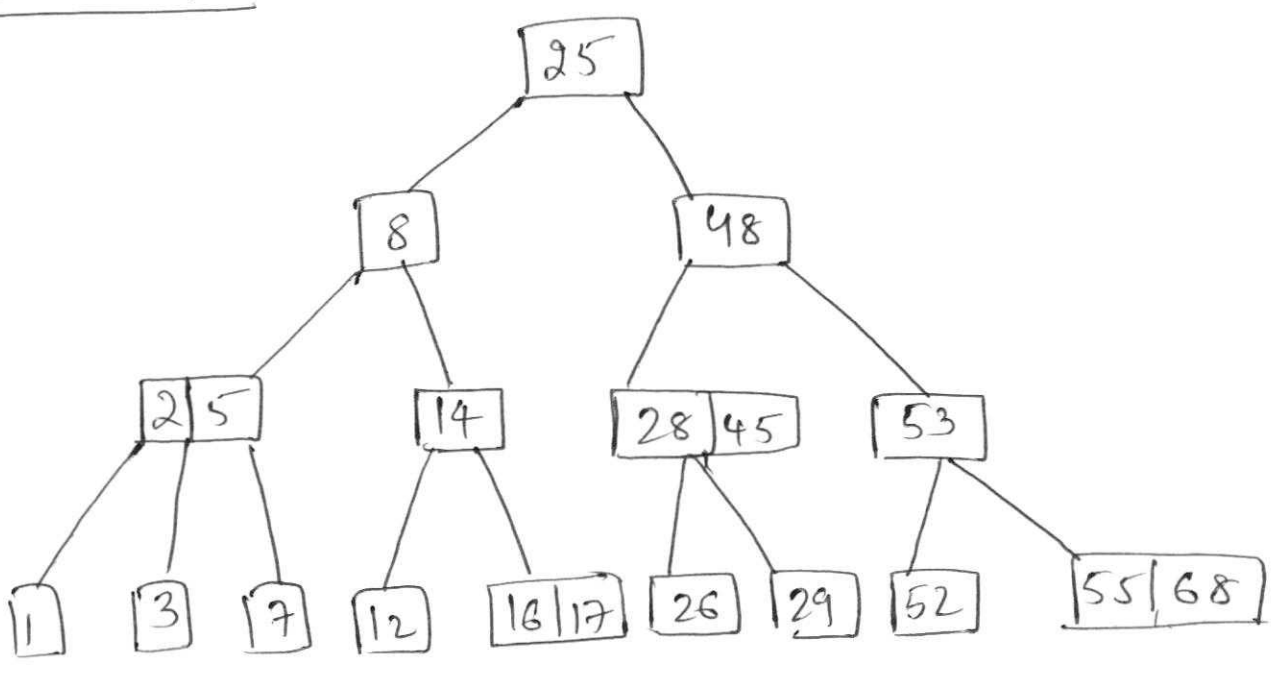
Insert 53:-


8 25
2 5
14
28 48
1
3
7
12
16 17
26
29
52 53 68
8 25
2 5
14
28 48 53
1
3
7
12
16 17
26
29
52
68
25
8
48
2 5
14
28
53
1
3
7
12
16 17
26
29
52
68

Insert 55:-



Insert 45:-



DATA STRUCTURES ASSIGNMENT QUESTIONS

ASSESMENT OF LEARNING OBJECTIVES AND OUT COMES: DIRECT

Blooms Taxonomy:

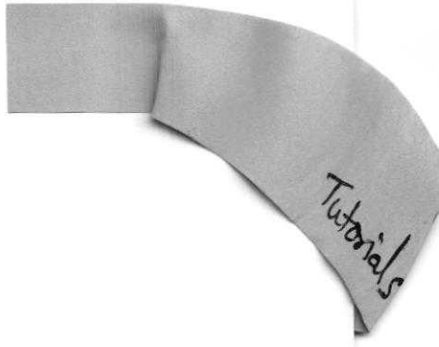
LEVEL 1	**REMEMBERING**	Exhibit memory of previously learned material by recalling facts, terms, basic concepts, and answers
LEVEL 2	**UNDERSTANDING**	Demonstrate understanding of facts and ideas by organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas.
LEVEL 3	**APPLYING**	Solve problems to new situations by applying acquired knowledge, facts, techniques and rules in a different way
LEVEL 4	**ANALYZING**	Examine and break information into parts by identifying motives or causes. Make inferences and find evidence to support generalizations.
LEVEL 5	**EVALUATING**	Present and defend opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.
LEVEL 6	**CREATING**	Compile information together in a different way by combining elements in a new pattern or proposing alternative solutions.

Unit Wise Assignment Questions (With different Levels of thinking (Blooms Taxonomy))

Unit – 1	
1.	Explain about Time complexity and Space complexity **Level 1 of Blooms Taxonomy, Course Outcome-5**
2.	Explain about different Asymptotic notations(Big O,Omega,Theta).Give Examples to each **Level 1 of Blooms Taxonomy, Course Outcome- 5**
3.	Explain the procedure and Write the C Program for Concatenating two single linked lists **Level 2 of Blooms Taxonomy, Course Outcome- 1**
Unit – 2	
1.	Write the stack ADT **Level 2 Of Blooms Taxonomy,Course Outcome- 2**
2	Write the steps for converting expression from infix to postfix with example **Level-2,Course Outcome- 2**
Unit – 3	
1.	Write c program for heap sorting method **Level-2, Course Outcome- 3**
Unit – 4	
1.	Define the following Hash Table, Hash Functions and Collision resolution techniques. **Level-3,Course Outcome -4**

Unit – 5	
1.	Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45. Insert into B-Tree of order 3 **Level-4,Course Outcome- 4**

Tutorial Evidence



KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Academic Year : 2016-2017 **Subject: Data Structures**

Class: II B.Tech **Faculty Name: Neil Gogte** **Duration :** 50 min.

For the benefit and improvement of the students, IT Department is conducting tutorial classes for the students of II B.Tech – I Sem. Of IT. The summary of the tutorials are given below.

S. No.	DATE	TOPIC	DESCRIPTION
1	18/06/16	Asymptotic notations	Representing time complexity of the algorithm using Big-O, Omega and Theta Notations.
2	02/07/16	Linked Lists	Creating the linked list of values,. Traversing , inserting new values, deleting and searching.
3	16/07/16	STACKS & QUEUES	Implementing the operations on linear data structures like stacks and queues.
4	30/07/16	CIRCULAR QUEUES	Implementing the concept of circular queue to effectively utilize the freed memory of queue.
5	20/08/16	BINARY TREES	Creating biray trees, traversing using inorder, preorder and postorder techniques. Also inseting, deleting and searching opeations.
6	03/09/16	GRAPH TRAVERSALS	Using BFS and DFS algorithms to traverse the Graph.
7	17/09/16	HASHING	Implenting the new hash fuction based method to store records in memory and retrieve with O(1) complexity
8	01/10/16	SEARCH TREES	Implement AVL TREE, B TREES which enable us to have a quick search of elements in the tree.

Subject In-charge **HOD, IT**

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

TUTORIAL TOPICS

SUB: Data Structures **CLASS:IT** **YEAR/SEM: II/1**

TUTORIAL NUMBER	DATE	TOPICS
1	4/7/2015	Recursion and Arrays.
2	11/7/2015	Reverse a linked list, Find Length of a Linked List (Recursive)
3	16/7/2015	Merge two sorted linked lists
4	25/7/2015	program to implement sparse matrix using linked list.
5	1/8/2015	**Towers of Hanoi problem**
6	8/8/2015	Implement a stack using two queues, Implement a queue using two stacks.
7	22/8/2015	program to find the Total Nodes and Total Leaf Nodes of Binary Tree
8	12/9/2015	Graph traversals- BFS
9	26/9/2015	Merge Sort, Shell sort
10	3/10/2015	program to find the height of a Binary search Tree
11	10/10/2015	Brute Force algorithm

Result Analysis to identify weak and advanced learners

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(Narayanaguda, Hyderabad)

Class: II B.Tech(IT) **Code:123BP**
Semester: I **A.Y: 2016-2017**

Result Analysis

Sl No	Roll No	Name of the student	Student performance upto previous Semester	Student performance on prerequisite course if any(C language)	Student performance based on Internal exam marks	Overall Grade
1	15BD1A1201	A AVINASH GOUD	C	C	B	C
2	15BD1A1202	A SAHITYA	C	C	B	C
3	15BD1A1203	AKULA AKHILA	A	B	B	B
4	15BD1A1204	AMRUTHAKAVI DIVYASREE	A	A	B	A
5	15BD1A1205	ANNAPARTHI VYSHNAVI	C	A	C	B
6	15BD1A1206	APOORVA RAMARAPU	A	A	A	A
7	15BD1A1207	BHARGAVI NAINI	A	A	A	A
8	15BD1A1208	BYROSU KISHAN	A	A	A	A
9	15BD1A1209	CH D STAN ROBINS	C	C	C	C
10	15BD1A1210	CHAPALAMADUGU JANARDHAN NITIN	C	C	C	C
11	15BD1A1211	CHEGI REDDY ABHAY SIMHA REDDY	C	C	C	C
12	15BD1A1212	CHENNOJU NAGARJUNA	C	C	A	B

13	15BD1A1213	CHEVVA KUSHAL	A	A	A	A
14	15BD1A1214	CHIKILE YUVA MAITHRI	C	C	A	B
15	15BD1A1215	DIVYA GATTANI	A	A	A	A
16	15BD1A1216	DUDDALA KARTHIK	A	A	B	A
17	15BD1A1217	GODAVARTHI SAI SAHITHI	A	C	A	B
18	15BD1A1218	GUDUR NIKHIL SARMA	C	C	A	B
19	15BD1A1219	HARSH DESAI	C	C	A	B
20	15BD1A1220	HEENA	B	A	C	B
21	15BD1A1221	HIRDESH ARORA	A	A	A	A
22	15BD1A1222	ISHA KHAKHAR	C	A	A	A
23	15BD1A1223	JAIDI SRAVANI	A	A	B	A
24	15BD1A1224	JANAMANCHI GANESH	A	A	A	A
25	15BD1A1225	K R SPANDANA	A	A	A	A
26	15BD1A1226	K SAI KALYAN	C	C	B	C
27	15BD1A1227	K.SWAPNIL	C	C	C	C
28	15BD1A1228	KALLURI ANVITHA REDDY	A	B	A	A
29	15BD1A1229	KANCHARLA NISHAL CHAKRAVARTHY	B	C	B	B
30	15BD1A1230	KANDIKONDA BALA HARSHITH	A	B	A	A
31	15BD1A1231	KANDUKURI REENA	C	B	B	B
32	15BD1A1232	KASIREDDY SANJANA REDDY	C	A	A	A

33	15BD1A1233	KATIPALLY RAJESH	B	A	A	A
34	15BD1A1234	KOMATI EFRAIM RAJ	C	C	B	B
35	15BD1A1235	KOMURAVELLI SAI NIKHIL	A	A	A	A
36	15BD1A1237	KUNCHAM MADHU KUMAR	C	C	B	C
37	15BD1A1238	MAHANKALI GAYATRI	C	B	B	B
38	15BD1A1239	MARUKUKULA PAVAN SAI	C	C	A	B
39	15BD1A1240	MARYADA SUCHET REDDY	C	C	B	C
40	15BD1A1241	MUKUNDA VANI SRIVAISHNAVI	C	B	B	B
41	15BD1A1242	N SAI SAMYUKTHA	A	A	A	A
42	15BD1A1243	NALLAGONDA VENKATA HARI KRISHNA	A	A	A	A
43	15BD1A1244	NAVYA TUBATI	A	A	A	A
44	15BD1A1245	NENAVATH MAHESH	B	A	B	B
45	15BD1A1246	NIMMAGUDEM SRI VARSHA	C	C	B	C
46	15BD1A1247	P SURYA TEEJA	C	C	C	C
47	15BD1A1248	PAMULA SAI AVINASH	A	A	A	A
48	15BD1A1249	PASUPULETI SHIVA SAI DEEPTHI	C	B	B	B
49	15BD1A1250	PONUGOTI PREETHAM	B	A	A	A
50	15BD1A1252	SHASAM RANGA	A	A	A	A

		PAVAN				
51	15BD1A1253	T V PRANAV VAMSHA TILAK	C	A	C	B
52	15BD1A1254	TALLURI ABHIJIT	A	A	A	A
53	15BD1A1255	THATIKONDA NIHAR	C	B	A	B
54	15BD1A1257	V AKSHAY REDDY	C	B	C	C
55	15BD1A1258	VARUN SAXENA	C	B	B	B
56	15BD1A1259	VEMULA SRINITH REDDY	C	C	C	C
57	15BD1A1260	ZOHEBUDDIN KHAN	C	C	C	C

Department of Information Technology

Result Analysis based on External Marks

Name of the faculty : Mr. Neil Gogte
Branch & Section: IT
Subject: **Data Structures**

A.Y : 2016-17
Exam: University
Year: II/I

SL.No	REG. NO	NAME OF THE STUDENT	TOTAL	Grade
		Max Marks	75.00	
1	15BD1A1201	A AVINASH GOUD	16	C
2	15BD1A1202	A SAHITYA	27	B
3	15BD1A1203	AKULA AKHILA	26	B
4	15BD1A1204	AMRUTHAKAVI DIVYASREE	28	B
5	15BD1A1205	ANNAPARTHI VYSHNAVI	26	B
6	15BD1A1206	APOORVA RAMARAPU	17	C
7	15BD1A1207	BHARGAVI NAINI	26	B
8	15BD1A1208	BYROSU KISHAN	30	B
9	15BD1A1209	CH D STAN ROBINS	1	C
10	15BD1A1210	CHAPALAMADUGU JANARDHAN NITIN	7	C
11	15BD1A1211	CHEGI REDDY ABHAY SIMHA REDDY	0	C
12	15BD1A1212	CHENNOJU NAGARJUNA	11	C
13	15BD1A1213	CHEVVA KUSHAL	30	B
14	15BD1A1214	CHIKILE YUVA MAITHRI	6	C
15	15BD1A1215	DIVYA GATTANI	26	B
16	15BD1A1216	DUDDALA KARTHIK	26	B
17	15BD1A1217	GODAVARTHI SAI SAHITHI	26	B
18	15BD1A1218	GUDUR NIKHIL SARMA	15	C
19	15BD1A1219	HARSH DESAI	8	C
20	15BD1A1220	HEENA	27	B
21	15BD1A1221	HIRDESH ARORA	31	B
22	15BD1A1222	ISHA KHAKHAR	26	B
23	15BD1A1223	JAIDI SRAVANI	27	B
24	15BD1A1224	JANAMANCHI GANESH	26	B
25	15BD1A1225	K R SPANDANA	27	B
26	15BD1A1226	K SAI KALYAN	26	B
27	15BD1A1227	K.SWAPNIL	8	C
28	15BD1A1228	KALLURI ANVITHA REDDY	30	B
29	15BD1A1229	KANCHARLA NISHAL CHAKRAVARTHY	13	C
30	15BD1A1230	KANDIKONDA BALA HARSHITH	26	B
31	15BD1A1231	KANDUKURI REENA	26	B
32	15BD1A1232	KASIREDDY SANJANA REDDY	16	C
33	15BD1A1233	KATIPALLY RAJESH	26	B
34	15BD1A1234	KOMATI EFRAIM RAJ	22	B
35	15BD1A1235	KOMURAVELLI SAI NIKHIL	26	B
36	15BD1A1237	KUNCHAM MADHU KUMAR	12	C
37	15BD1A1238	MAHANKALI GAYATRI	27	B

38	15BD1A1239	MARUKUKULA PAVAN SAI	7	C
39	15BD1A1240	MARYADA SUCHET REDDY	16	C
40	15BD1A1241	MUKUNDA VANI SRIVAISHNAVI	17	C
41	15BD1A1242	N SAI SAMYUKTHA	31	B
42	15BD1A1243	NALLAGONDA VENKATA HARI KRISHNA	27	B
43	15BD1A1244	NAVYA TUBATI	27	B
44	15BD1A1245	NENAVATH MAHESH	13	C
45	15BD1A1246	NIMMAGUDEM SRI VARSHA	5	C
46	15BD1A1247	P SURYA TEEJA	4	C
47	15BD1A1248	PAMULA SAI AVINASH	26	B
48	15BD1A1249	PASUPULETI SHIVA SAI DEEPTHI	26	B
49	15BD1A1250	PONUGOTI PREETHAM	26	B
50	15BD1A1252	SHASAM RANGA PAVAN	26	B
51	15BD1A1253	T V PRANAV VAMSHA TILAK	4	C
52	15BD1A1254	TALLURI ABHIJIT	17	C
53	15BD1A1255	THATIKONDA NIHAR	0	C
54	15BD1A1257	V AKSHAY REDDY	15	C
55	15BD1A1258	VARUN SAXENA	6	C
56	15BD1A1259	VEMULA SRINITH REDDY	0	C
57	15BD1A1260	ZOHEBUDDIN KHAN	4	C

Sr No	H.T. No	Name	Student Performance upto prev Sem	Student Performance on prerequisite course if any	Student Performance based on Internal exam	Student Performance based on Internal exam marks	Overall Rank	Overall Rank	Int	Ext
1	13BD1A1221	KAMALLA LAVANYA	57.7	NA	B	15	D	31	15	34
2	14BD1A1201	A C SHREYA DEVI	57.3	NA	C	13	C	40	13	36
3	14BD1A1202	AHMED NAIMANUDDIN	68.9	NA	A	20	C	46	20	28
4	14BD1A1203	AITHA ABHISARIKA	75	NA	A	22	C	50	22	50
5	14BD1A1204	ALUGOJU APARNA	75.6	NA	C	14	C	40	14	42
6	14BD1A1205	AMBATI HARSHITHA	71	NA	A	21	D	38	21	41
7	14BD1A1206	AMRUTHA VALLI SALIKE	77.2	NA	A	22	C	48	22	44
8	14BD1A1207	ANISH STEPHEN MATHEW	78.8	NA	A	23	B	53	23	40
9	14BD1A1208	APEKSHA SREEDHAR	42.5	NA	D	6	D	7	6	35
10	14BD1A1209	B H ANIRUDH	47.3	NA	D	9	D	16	9	32
11	14BD1A1210	B MANISHA	41.9	NA	D	6	D	6	6	42
12	14BD1A1211	B SESHA PAVANI	52.1	NA	C	12	D	23	12	36
13	14BD1A1212	BOJJI VADIRAJA VAMSHI KRISHNA	75.4	NA	A	22	B	52	22	37
14	14BD1A1213	DEVULAPALLY SRIMUKHI	57	NA	C	13	D	19	13	35
15	14BD1A1214	DONTHULA HARSHINI	71.8	NA	A	23	C	49	23	39
16	14BD1A1215	ALAPATI SRIKANTH	63	NA	B	19	C	45	19	28
17	14BD1A1216	G DHIKSHITA	65.7	NA	B	19	C	45	19	40
18	14BD1A1217	GARIMELLA RAM SIMRAN	53	NA	C	11	D	26	11	31
19	14BD1A1218	GUNDU GOUTHAMILATHA	47.6	NA	C	11	D	19	11	44
20	14BD1A1219	GUNTHA PRIYANKA	58.2	NA	B	19	C	46	19	41
21	14BD1A1220	GYARA MANASA	78.6	NA	A	24	B	55	24	27
22	14BD1A1221	JAKKULA SHRAVYA	74.7	NA	B	18	C	44	18	28
23	14BD1A1222	K NIKHIL REDDI	72.4	NA	B	18	C	45	18	17
24	14BD1A1223	KANCHERLA SHREYA REDDY	57.9	NA	A	21	C	47	21	41
25	14BD1A1224	KANCHUMARTHY PRASANNA KUMAR	68.9	NA	A	22	C	49	22	23
26	14BD1A1225	KARNATI VISHNU PRIYA	58.3	NA	B	17	C	43	17	45
27	14BD1A1226	KOMPELLY SAI PALLAVI	49.1	NA	D	9	D	17	9	27
28	14BD1A1227	KOPPURAPU PUJITHA REDDY	73.1	NA	B	19	C	49	19	35
29	14BD1A1228	KOTTUR DINESH REDDY	55.3	NA	B	16	D	29	16	9

30	14BD1A1229	M VINEETH	61	NA	A	20	C	46	20	12
31	14BD1A1230	MANCHUKONDA VAISHNAVI GUPTHA	67.2	NA	C	14	C	40	14	47
32	14BD1A1231	NALLA ARAVISH	71.4	NA	A	20	D	36	20	27
33	14BD1A1232	NAMPALLY SOWMYA	74.7	NA	A	21	C	47	21	27
34	14BD1A1233	PERAKALAPUDI BHANUSREE	51.3	NA	D	8	D	30	8	45
35	14BD1A1234	P KEERTHANA	68.7	NA	A	22	C	48	22	47
36	14BD1A1235	P NIKITHA REDDY	57.3	NA	C	14	D	26	14	33
37	14BD1A1236	PARIMI SRI DEEPTHI DRAKSHAYANI	60.6	NA	B	15	C	42	15	8
38	14BD1A1237	PENCHALA SACHITH SANTOSH	56.4	NA	C	10	D	17	10	19
39	14BD1A1238	PESARAMELLI SUPRIYA	57.4	NA	C	15	D	31	15	44
40	14BD1A1239	PINNINTI VEENA	61	NA	B	16	D	33	16	28
41	14BD1A1240	R DEEPTI	64.5	NA	A	22	B	53	22	28
42	14BD1A1241	R RHICHA SINGH	62.5	NA	A	22	C	49	22	41
43	14BD1A1242	RACHURI SANDEEP	77.1	NA	A	24	B	51	24	35
44	14BD1A1243	RESHMI VINOD	55.6	NA	B	18	D	31	18	35
45	14BD1A1244	RESHAM JAISWAL	52.8	NA	C	11	D	16	11	31
46	14BD1A1245	S SAI KUMAR	42.5	NA	D	6	D	10	6	32
47	14BD1A1246	S SINDHURA	63.6	NA	A	21	C	47	21	37
48	14BD1A1247	SAADIA TABASSUM	58.5	NA	B	17	C	43	17	43
49	14BD1A1248	SEEMAKURTHY SAI SRUTHI	70.9	NA	A	22	C	48	22	44
50	14BD1A1249	SHASHANK NETHA GUTTHI	67.7	NA	A	21	C	47	21	18
51	14BD1A1250	SHESHIT KARTHIKEYA AAKULA	51.1	NA	C	12	D	16	12	28
52	14BD1A1251	SNEHA BIDARKAR	69.8	NA	A	23	C	40	23	37
53	14BD1A1252	SOUMYA GUPTA	48.9	NA	B	15	D	15	15	26
54	14BD1A1253	SUSHMITHA A	47	NA	C	14	D	29	14	44
55	14BD1A1254	SWAMY SHIVANAND	54.5	NA	B	15	D	21	15	34
56	14BD1A1255	TADAKAMALLA SUMANTH	49.5	NA	D	8	D	8	8	15
57	14BD1A1256	V NIRMAL SRINITHYA	45.4	NA	D	8	D	12	8	41
58	14BD1A1257	VADUGULA PAVAN KUMAR	34.7	NA	B	12	B	21	8	44
59	14BD1A1259	VONGUR AKSHITHA	45.8	NA	B	15	B	20	8	43

Course Assesment

~~Result~~ ~~Analysis~~ at the end of the course

Keshav Memorial Institute Of Technology

Department of Information Technology

Course Outcome Attainment

Name of the faculty : Mr. Neil Gogte Academic Year: 2016-17

Branch & Section: IT Exam: I Internal

Subject: **Data Structures** Year: II Semester: I

S.No	HT No.	Question No.												Obj1	A1
		1A	1B	1C	2A	2B	2C	3A	3B	3C	4A	4B	4C		
Max. Marks ==>		5			5			5			5			10	5
1	15BD1A1201	0			4									5	5
2	15BD1A1202	3			2									5	0
3	15BD1A1203	4			5									6	5
4	15BD1A1204	5			5									7	5
5	15BD1A1205													7	0
6	15BD1A1206	4			5									7	5
7	15BD1A1207							4			5			6	5
8	15BD1A1208	5						5						9	5
9	15BD1A1209													2	0
10	15BD1A1210							1			1			2	4
11	15BD1A1211													4	0
12	15BD1A1212	1			2									5	0
13	15BD1A1213	5			5									5	5
14	15BD1A1214	5												3	4
15	15BD1A1215	4			5									8	5
16	15BD1A1216	5									3			6	5
17	15BD1A1217							5			4			6	5
18	15BD1A1218													**0**	0
19	15BD1A1219	3												5	2
20	15BD1A1220	3						3						7	5
22	15BD1A1221	5			5									9	5
23	15BD1A1222				4									3	5
24	15BD1A1223	5			5									6	0
25	15BD1A1224	4									5			6	5
26	15BD1A1225	5			5									7	5
27	15BD1A1226	5												8	5
28	15BD1A1227	1												5	0
29	15BD1A1228	4									4			6	5
30	15BD1A1229	5									2			6	5
31	15BD1A1230	4						5						6	5
32	15BD1A1231	1												8	0
34	15BD1A1232	3			3									8	5
35	15BD1A1233	4			3									9	5
36	15BD1A1234	1												4	0
37	15BD1A1235	5									4			8	5
38	15BD1A1237	1												7	5
39	15BD1A1238	3												4	5
40	15BD1A1239				0									3	5
41	15BD1A1240				2									6	5
42	15BD1A1241	3			1									4	5
43	15BD1A1242	4			5									8	5
44	15BD1A1243	5									5			8	5
45	15BD1A1244	4			5									10	5

46	15BD1A1245	2						4						6	5
47	15BD1A1246				0									1	5
48	15BD1A1247				0									1	0
49	15BD1A1248	5			5									6	5
50	15BD1A1249	3			2									6	5
51	15BD1A1250	3						4						8	5
52	15BD1A1252	5			5									6	5
53	15BD1A1253	1												3	0
54	15BD1A1254	5			4									9	5
55	15BD1A1255	1												4	4
56	15BD1A1257	0												2	0
57	15BD1A1258	2												4	5
58	15BD1A1259	1												3	0
59	15BD1A1260				1									2	0
60															
	SUM	147	0	0	88	0	0	31	0	0	33	0	0	315	199
	COUNT	44	0	0	26	0	0	8	0	0	9	0	0	57	56
	AVERAGE	3.34			3.38			3.88			3.67			5.5263	3.55

CO Mapping with Exam Questions:

CO - 1	**y**						**y**			**y**			y	
CO - 2	**y**						**y**			**y**			y	y
CO - 3														y
CO - 4				**y**									y	y
CO - 5													y	

Students Scored >Target %	32	58	58	17	58	58	7	58	58	7	58	58	33	41
% Students Scored >Target %	73%			65%			88%			78%			58%	73%

CO Attainment based on Exam Questions:

CO - 1	73%						88%			78%			58%	
CO - 2	73%						88%			78%			58%	73%
CO - 3														73%
CO - 4				65%									58%	73%
CO - 5													58%	

CO	Subj	obj	Asgn	Overall	Level
CO-1	79%	58%		69%	3
CO-2	79%	58%	73%	70%	3
CO-3			73%	73%	3
CO-4	65%	58%	73%	65%	3
CO-5		58%		58%	2

Attainment Leve	
1	<40%
2	40-60%
3	>60%

Overall Course Attainment = 2.8

Department of Information Technology

Course Outcome Attainment

Name of the faculty : Mr. Neil Gogte Academic Year: 2016-17

Branch & Section: IT Exam: II Internal

Subject: **Data Structures** Year: II Semester: I

S.No	HT No.	Question No.												Obj1	A1
		1A	1B	1C	2A	2B	2C	3A	3B	3C	4A	4B	4C		
Max. Marks ==>		5			5			5			5			10	5
1	15BD1A1201	5												5	5
2	15BD1A1202	2						3						5	5
3	15BD1A1203	2			5									8	5
4	15BD1A1204				5			5						7	5
5	15BD1A1205				5			3						7	4
6	15BD1A1206	4			5									6	5
7	15BD1A1207	5			5									8	5
8	15BD1A1208	5						5						6	5
9	15BD1A1209				0									5	5
10	15BD1A1210	1												4	5
11	15BD1A1211				0									3	4
12	15BD1A1212				4									7	5
13	15BD1A1213							5			5			8	5
14	15BD1A1214	1												7	5
15	15BD1A1215				5			5						9	5
16	15BD1A1216				4			4						6	5
17	15BD1A1217				5									7	5
18	15BD1A1218													0	5
19	15BD1A1219				1									6	5
20	15BD1A1220				5			3						6	5
21	15BD1A1221				5			5						9	5
22	15BD1A1222	5									5			8	5
23	15BD1A1223				4			5						6	5
24	15BD1A1224	5			5									7	5
25	15BD1A1225				5			5						7	5
26	15BD1A1226							4						7	5
27	15BD1A1227							2						5	5
28	15BD1A1228				5			3						6	5
29	15BD1A1229				2									6	5
30	15BD1A1230	2			4									7	5
31	15BD1A1231	2			5									6	5
32	15BD1A1232	5						5						6	5
33	15BD1A1233	4						5						6	5
34	15BD1A1234				0									6	5
35	15BD1A1235	5						5						7	5
36	15BD1A1237	4												5	5
37	15BD1A1238				4			3						6	5
38	15BD1A1239							1						6	5
39	15BD1A1240	1			4									7	5
40	15BD1A1241	2						5						7	5
41	15BD1A1242	5			5									6	5
42	15BD1A1243				5			4						7	5
43	15BD1A1244	5						5						8	5

44	15BD1A1245				3			5						6	5
45	15BD1A1246				2			2						6	5
46	15BD1A1247	2												3	5
47	15BD1A1248				5			5						6	5
48	15BD1A1249	2			5									6	5
49	15BD1A1250	5									5			8	5
50	15BD1A1252	4						5						7	5
51	15BD1A1253				5			3						7	2
52	15BD1A1254				5						4			8	5
53	15BD1A1255	2			5									8	5
54	15BD1A1257	2			5									8	5
55	15BD1A1258				5			2						7	5
56	15BD1A1259										0			7	5
57	15BD1A1260				0									7	5
58															
	SUM	87	0	0	142	0	0	112	0	0	19	0	0	365	280
	COUNT	26	0	0	36	0	0	28	0	0	5	0	0	57	57
	AVERAGE	3.35			3.94			4			3.8			6.4035	4.91

CO Mapping with Exam Questions:

CO - 1														
CO - 2							**y**						y	
CO - 3				**y**									y	
CO - 4	**y**									**y**			y	
CO - 5										**y**			y	y

Students Scored >Target %	14	58	58	29	58	58	24	58	58	4	58	58	53	56
% Students Scored >Target %	54%			81%			86%			80%			93%	98%

CO Attainment based on Exam Questions:

CO - 1														
CO - 2							86%						93%	
CO - 3				81%									93%	
CO - 4	54%									80%			93%	
CO - 5										80%			93%	98%

CO	Subj	obj	Asgn	Overall	Level
CO-1					
CO-2	86%	93%		89%	3
CO-3	81%	93%		87%	3
CO-4	67%	93%		80%	3
CO-5	80%	93%	98%	90%	3

Attainment Leve	
1	<40%
2	40-60%
3	>60%

Overall Course Attainment = 3

Department of Information Technology

Course Outcome Attainment

Name of the faculty : Mr. Neil Gogte 2016-17

Branch & Section: IT Exam: University

Subject: **Data Structures** Year: II Semester: I

SL.No	REG. NO	NAME OF THE STUDENT	TOTAL
		Max Marks	75.00
1	15BD1A1201	A AVINASH GOUD	16
2	15BD1A1202	A SAHITYA	27
3	15BD1A1203	AKULA AKHILA	26
4	15BD1A1204	AMRUTHAKAVI DIVYASREE	28
5	15BD1A1205	ANNAPARTHI VYSHNAVI	26
6	15BD1A1206	APOORVA RAMARAPU	17
7	15BD1A1207	BHARGAVI NAINI	26
8	15BD1A1208	BYROSU KISHAN	30
9	15BD1A1209	CH D STAN ROBINS	1
10	15BD1A1210	CHAPALAMADUGU JANARDHAN NITI	7
11	15BD1A1211	CHEGI REDDY ABHAY SIMHA REDDY	0
12	15BD1A1212	CHENNOJU NAGARJUNA	11
13	15BD1A1213	CHEVVA KUSHAL	30
14	15BD1A1214	CHIKILE YUVA MAITHRI	6
15	15BD1A1215	DIVYA GATTANI	26
16	15BD1A1216	DUDDALA KARTHIK	26
17	15BD1A1217	GODAVARTHI SAI SAHITHI	26
18	15BD1A1218	GUDUR NIKHIL SARMA	15
19	15BD1A1219	HARSH DESAI	8
20	15BD1A1220	HEENA	27
21	15BD1A1221	HIRDESH ARORA	31
22	15BD1A1222	ISHA KHAKHAR	26
23	15BD1A1223	JAIDI SRAVANI	27
24	15BD1A1224	JANAMANCHI GANESH	26
25	15BD1A1225	K R SPANDANA	27
26	15BD1A1226	K SAI KALYAN	26
27	15BD1A1227	K.SWAPNIL	8
28	15BD1A1228	KALLURI ANVITHA REDDY	30

29	15BD1A1229	KANCHARLA NISHAL CHAKRAVARTH	13
30	15BD1A1230	KANDIKONDA BALA HARSHITH	26
31	15BD1A1231	KANDUKURI REENA	26
32	15BD1A1232	KASIREDDY SANJANA REDDY	16
33	15BD1A1233	KATIPALLY RAJESH	26
34	15BD1A1234	KOMATI EFRAIM RAJ	22
35	15BD1A1235	KOMURAVELLI SAI NIKHIL	26
36	15BD1A1237	KUNCHAM MADHU KUMAR	12
37	15BD1A1238	MAHANKALI GAYATRI	27
38	15BD1A1239	MARUKUKULA PAVAN SAI	7
39	15BD1A1240	MARYADA SUCHET REDDY	16
40	15BD1A1241	MUKUNDA VANI SRIVAISHNAVI	17
41	15BD1A1242	N SAI SAMYUKTHA	31
42	15BD1A1243	NALLAGONDA VENKATA HARI KRISH	27
43	15BD1A1244	NAVYA TUBATI	27
44	15BD1A1245	NENAVATH MAHESH	13
45	15BD1A1246	NIMMAGUDEM SRI VARSHA	5
46	15BD1A1247	P SURYA TEEJA	4
47	15BD1A1248	PAMULA SAI AVINASH	26
48	15BD1A1249	PASUPULETI SHIVA SAI DEEPTHI	26
49	15BD1A1250	PONUGOTI PREETHAM	26
50	15BD1A1252	SHASAM RANGA PAVAN	26
51	15BD1A1253	T V PRANAV VAMSHA TILAK	4
52	15BD1A1254	TALLURI ABHIJIT	17
53	15BD1A1255	THATIKONDA NIHAR	0
54	15BD1A1257	V AKSHAY REDDY	15
55	15BD1A1258	VARUN SAXENA	6
56	15BD1A1259	VEMULA SRINITH REDDY	0
57	15BD1A1260	ZOHEBUDDIN KHAN	4
58			
59			
60			
		sum	1071
		avg	19.472727

no. of students scored more than target %	0
no. of students present	57
Percentage of students scored more than target %	0%
Attainment level	**1**

Attainment Level	Percentage
1	<40%
2	40-60%
3	>60%

Department of Information Technology

Course Outcome Attainment

Name of the faculty : Mr. Neil Gogte Academic Year 2016-17

Branch & Section: IT Exam Overall

Subject: **Data structures** Year: II Semester: I

Course Outc	1st Internal Exam	2nd Internal Exam	University Exam
CO1	3.00		1
CO2	3.00	3	1
CO3	3.00	3	1
CO4	3.00	3	1
CO5	2.00	3	1

Attainment level of Course Outcomes

	Course Outcomes	Attainment
CO1	Describe the concept of Data Structure such as array, linked list, stacks and queues.	1.5
CO2	Explain the abstract properties and operations of various data structures such	1.5
CO3	Apply algorithms for solving problems like soring, searching, insertion and deletion of data	1.5
CO4	Use data structure concepts for realistic problems.	1.5
CO5	Analyze the performance of algorithms.	1.375

Average 1.5

Overall course attainment l 1

Faculty Signature

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Program Specific Outcome Attainment

Name of Faculty:Neil Gogte | Academic Year: 2016-17

Branch & Section: IT | Year: II

Subject: Data Structures

Course outcome attainment

CO	IstMid	IIndMid	Univ	DIRECT	INDIRECT	OVERALL
CO1	3		1	1.5		1.2
CO2	3	3	1	1.5		1.2
CO3	3	3	1	1.5		1.2
CO4	3	3	1	1.5		1.2
CO5	2	3	1	1.375		1.1
average				1.475	#DIV/0!	1.18

CO-PSO mapping

	PSO1	PSO2
CO1	2	1
CO2	2	3
CO3	2	2
CO4	3	2
CO5	2	2
average	2	2
attainment		

Faculty

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Program Outcome Attainment

Name of Faculty: Neil Gogte | Academic Year: 2016-17

Branch & Section: IT | Year: II | Semester: I

Subject: Data Structures

Course outcome attainment

CO	Mid-1	Mid-2	AVG	Univ	DIRECT	INDIREC	OVERALL
CO1	3			1	1.5		1.2
CO2	3	3		1	1.5		1.2
CO3	3	3		1	1.5		1.2
CO4	3	3		1	1.5		1.2
CO5	2	3		1	1.375		1.1
ATTAINMENT			#DIV/0!	1	1.475	#DIV/0!	1.18

CO-PO mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2		2									
CO2	2	2	2	2								
CO3	2		2									
CO4		1	3									
CO5	2	3	1	2	2							
AVERAGE	2	2	2	2								
ATTAINMENT												

Faculty

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Information Technology

Course Outcome Attainment

Name of the faculty : **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: **IT** Exam: **I Internal**

Subject: **Data Structures** Year: **II** Semester: **I**

S.No	HT No.	Question No.												Obj1	A1
		1A	1B	1C	2A	2B	2C	3A	3B	3C	4A	4B	4C		
Max. Marks ==>		2	3		2.5	2.5		2	3		2.5	2.5		10	5
1	13BD1A1221					1		1						3	5
2	14BD1A1201				0	0					0	2		3	5
3	14BD1A1202	0	0		1	1								3	5
4	14BD1A1203				1.5	1					0.5			3	5
5	14BD1A1204	2	3					2	2.5					10	5
6	14BD1A1205				0	0		1.5	5					3	5
7	14BD1A1206				2	2.5		2	3					8	5
8	14BD1A1207	1	1.5		1	0								6	5
9	14BD1A1208							1.5	1		1	0		6	5
10	14BD1A1209	0	0		0	0								AB	5
11	14BD1A1210				1.5	0.5					0	0		3	5
12	14BD1A1211	0	0		0.5	1.5								3	5
13	14BD1A1212				0	0		1	1					3	5
14	14BD1A1213	1	1		0	0								3	5
15	14BD1A1214	1	0		0	1.5								3	5
16	14BD1A1215	0	1		1	0								3	5
17	14BD1A1216				0	0.5		1.5	0					3	5
18	14BD1A1217							2	0					3	5
19	14BD1A1218				0	1.5		2	0					6	5
20	14BD1A1219	2	0								0	0		3	5
21	14BD1A1220	0	0					1.5	0.5					3	5
22	14BD1A1221					1.5		1.5						3	5
23	14BD1A1222	0	0					1	1					3	5
24	14BD1A1223		1.5					1.5	1					3	5
25	14BD1A1224	0	0		0	0								AB	5
26	14BD1A1225							1	0.5		0	0		3	5
27	14BD1A1226					0.5		1						3	5
28	14BD1A1227				0.5	1								3	5
29	14BD1A1228				0.5			1						3	5
30	14BD1A1229	1			0.5									AB	5
31	14BD1A1230				1.5						0	0.5		3	5
32	14BD1A1231	1	1											3	5
33	14BD1A1232				0.5	1.5		0	0					3	5
34	14BD1A1233				1.5	0		0	0					3	5
35	14BD1A1234		2					0	0					3	5
36	14BD1A1235				0.5	1.5					0	0		3	5
37	14BD1A1236	1	1									1.5		6	5
38	14BD1A1237				1	0		1	0					3	5
39	14BD1A1238				1.5	0					0.5	0		3	5
40	14BD1A1239				2									3	5

41	14BD1A1240							2	0		0	2		6	5
42	14BD1A1241	1.5	0.5											3	5
43	14BD1A1242					1						1		3	5
44	14BD1A1243					0.5		0.5	1					3	5
45	14BD1A1244					0		1.5	0.5					3	5
46	14BD1A1245	1	0								1			3	5
47	14BD1A1246	0.5	1					0	0					3	5
48	14BD1A1247					0		2						3	5
49	14BD1A1248	0.5	1					1.5	1					6	5
50	14BD1A1249				1	0					1			3	5
51	14BD1A1250		0.5									1.5		3	5
52	14BD1A1251	0	1			1								3	5
53	14BD1A1252	0.5			2	1								6	5
54	14BD1A1253	2	0		1.5	0								6	5
55	14BD1A1254							2				0		3	5
56	14BD1A1255		1			1								3	5
57	14BD1A1256					1		0.5						3	5
58	14BD1A1257	0	1					1	0					3	5
59	14BD1A1259							0.5	0		1.5	0		3	5
	SUM	16	18	0	23	22.5	0	35.5	18	0	5.5	8.5	0	204	295
	COUNT	23	25	0	29	35	0	30	23	0	13	14	0	56	59
	AVERAGE	0.6957	0.72		0.79	0.6429		1.18	0.78		0.42	0.61		3.6429	5

CO Mapping with Exam Questions:

CO - 1	**Y**	**Y**		**Y**									**Y**	**Y**
CO - 2					**Y**					**Y**	**Y**		**Y**	**Y**
CO - 3							**Y**	**Y**					**Y**	**Y**
CO - 4														
CO - 5														

Students Scored >Target %	4	2	59	9	7	59	14	3	59	1	4	59	13	59
% Students Scored >Target %	17%	8%		31%	20%		47%	13%		8%	29%		23%	100%

CO Attainment based on Exam Questions:

CO - 1	17%	8%		31%									23%	100%
CO - 2					20%					8%	29%		23%	100%
CO - 3							47%	13%					23%	100%
CO - 4														
CO - 5														

CO	**Subj**	obj	Asgn	Overall	Level
CO-1	19%	23%	100%	47%	2
CO-2	19%	23%	100%	47%	2
CO-3	30%	23%	100%	51%	2
CO-4					
CO-5					

Attainment Leve	
1	<40%
2	40-60%
3	>60%

Overall Course Attainment = 2

Keshav Memorial Institute of Technology

Department of Information Technology

Course Outcome Attainment

Name of the faculty : **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: **IT** Exam: **II Internal**

Subject: **Data Structures** Year: **II** Semester: I

S.No	HT No.	Question No.												Obj1	A1
		1A	1B	1C	2A	2B	2C	3A	3B	3C	4A	4B	4C		
Max. Marks ==>		2	3		2.5	2.5		2	3		2.5	2.5		10	5
1	13BD1A1221					1		1						6	5
2	14BD1A1201				2	2.5					1	2		8	5
3	14BD1A1202	0	0		0	0								8	5
4	14BD1A1203	2	2		1.5	1								7	5
5	14BD1A1204							2	2		1	2		8	5
6	14BD1A1205				0	0		2	2.5					8	5
7	14BD1A1206				2	2.5					1	2		9	5
8	14BD1A1207	0	0		0	0								7	5
9	14BD1A1208							1.5	1		1	2		7	5
10	14BD1A1209	0	0		0	0								6	5
11	14BD1A1210				1.5	0.5					0	2.5		8	5
12	14BD1A1211	2	1		0.5	1.5								7	5
13	14BD1A1212				0.5	1		1	1					8	5
14	14BD1A1213	1	1		2.5	0.5								8	5
15	14BD1A1214	1	0		0	1.5								7	5
16	14BD1A1215				1	0								8	5
17	14BD1A1216				0	1.5		1.5	0					6	5
18	14BD1A1217							2	2.5					8	5
19	14BD1A1218				0	1.5		2	0					8	5
20	14BD1A1219	2	1		1.5	2								8	5
21	14BD1A1220				2.5	0		1.5	0.5					6	5
22	14BD1A1221					1.5		1.5						5	5
23	14BD1A1222	0	0		0	0								0	5
24	14BD1A1223		1.5					1.5						7	5
25	14BD1A1224	0	0		0	0								4	5
26	14BD1A1225							1	1.5		0	0		8	5
27	14BD1A1226					0.5		1						8	5
28	14BD1A1227				2.5	1								8	5
29	14BD1A1228				0.5									8	5
30	14BD1A1229				0.5									8	5
31	14BD1A1230				1.5						2	1.5		8	5
32	14BD1A1231	1	1											8	5
33	14BD1A1232				0.5	1.5			0.5					8	5
34	14BD1A1233				1.5	2		2	1					8	5
35	14BD1A1234		2.5					1.5	1.5					8	5
36	14BD1A1235				0.5	1.5					0	0.5		8	5
37	14BD1A1236		1									1.5		6	5
38	14BD1A1237	0	0		0	0								0	5
39	14BD1A1238				1.5	1.5					0.5	2		8	5
40	14BD1A1239				2									7	5
41	14BD1A1240							2	1		0	2		7	5
42	14BD1A1241	1.5	2.5											8	5

43	14BD1A1242					1								9	5
44	14BD1A1243					0.5		0.5						8	5
45	14BD1A1244					2		1.5	0.5					7	5
46	14BD1A1245	0	0		0	0								7	5
47	14BD1A1246	2	1					2	1.5					7	5
48	14BD1A1247					2.5		2						8	5
49	14BD1A1248	0.5	1					1.5						8	5
50	14BD1A1249	0	0		0	0								0	5
51	14BD1A1250		0.5											7	5
52	14BD1A1251	2	1			1								8	5
53	14BD1A1252	0.5				1								8	5
54	14BD1A1253	2	2		2.5	1								8	5
55	14BD1A1254							2				2		7	5
56	14BD1A1255					1								7	5
57	14BD1A1256					1		1.5						8	5
58	14BD1A1257	1.5	1					1	0					7	5
59	14BD1A1259							1.5	1		1.5	1		8	5
	SUM	19	20	0	29	38	0	38.5	18	0	8	21	0	418	295
	COUNT	21	24	0	32	39	0	25	17	0	11	13	0	59	59
	AVERAGE	0.905	0.833		0.90625	1		1.54	1.06		0.73	1.62		7.0847	5

CO Mapping with Exam Questions:

CO - 1														
CO - 2														
CO - 3	**Y**	**Y**											**Y**	**Y**
CO - 4				**Y**	**Y**						**Y**		**Y**	**Y**
CO - 5							**Y**	**Y**		**Y**			**Y**	**Y**

Students Scored >Target %	11	5	59	13	14	59	24	6	59	2	10	59	55	59
% Students Scored >Target %	52%	21%		41%	36%		96%	35%		18%	77%		93%	100%

CO Attainment based on Exam Questions:

CO - 1														
CO - 2														
CO - 3	52%	21%											93%	100%
CO - 4				41%	36%						77%		93%	100%
CO - 5							96%	35%		18%			93%	100%

CO	Subj	obj	Asgn	Overall	Level
CO-1					
CO-2					
CO-3	37%	93%	100%	77%	3
CO-4	51%	93%	100%	81%	3
CO-5	50%	93%	100%	81%	3

Attainment Leve	
1	<40%
2	40-60%
3	>60%

Overall Course Attainment = 3

Keshav Memorial Institute of Technology

Department of Information Technology

Course Outcome Attainment

Name of the faculty : **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: IT Exam: **University**

Subject: **Data Structures** Year: **II** Semester **I**

SL.No	REG. NO	NAME OF THE STUDENT	TOTAL
		Max Marks	75.00
1	13BD1A1221	KAMALLA LAVANYA	34
2	14BD1A1201	A C SHREYA DEVI	36
3	14BD1A1202	AHMED NAIMANUDDIN	28
4	14BD1A1203	AITHA ABHISARIKA	50
5	14BD1A1204	ALUGOJU APARNA	42
6	14BD1A1205	AMBATI HARSHITHA	41
7	14BD1A1206	AMRUTHA VALLI SALIKE	44
8	14BD1A1207	ANISH STEPHEN MATHEW	40
9	14BD1A1208	APEKSHA SREEDHAR	35
10	14BD1A1209	B H ANIRUDH	32
11	14BD1A1210	B MANISHA	42
12	14BD1A1211	B SESHA PAVANI	36
13	14BD1A1212	BOJJI VADIRAJA VAMSHI KRISHNA	37
14	14BD1A1213	DEVULAPALLY SRIMUKHI	35
15	14BD1A1214	DONTHULA HARSHINI	39
16	14BD1A1215	ALAPATI SRIKANTH	28
17	14BD1A1216	G DHIKSHITA	40
18	14BD1A1217	GARIMELLA RAM SIMRAN	31
19	14BD1A1218	GUNDU GOUTHAMILATHA	44
20	14BD1A1219	GUNTHA PRIYANKA	41
21	14BD1A1220	GYARA MANASA	27
22	14BD1A1221	JAKKULA SHRAVYA	28
23	14BD1A1222	K NIKHIL REDDI	17
24	14BD1A1223	KANCHERLA SHREYA REDDY	41
25	14BD1A1224	KANCHUMARTHY PRASANNA KUMAR	23
26	14BD1A1225	KARNATI VISHNU PRIYA	45
27	14BD1A1226	KOMPELLY SAI PALLAVI	27
28	14BD1A1227	KOPPURAPU PUJITHA REDDY	35
29	14BD1A1228	KOTTUR DINESH REDDY	9
30	14BD1A1229	M VINEETH	12
31	14BD1A1230	MANCHUKONDA VAISHNAVI GUPTHA	47
32	14BD1A1231	NALLA ARAVISH	27
33	14BD1A1232	NAMPALLY SOWMYA	27
34	14BD1A1233	PERAKALAPUDI BHANUSREE	45
35	14BD1A1234	P KEERTHANA	47
36	14BD1A1235	P NIKITHA REDDY	33

37	14BD1A1236	PARIMI SRI DEEPTHI DRAKSHAYANI	8
38	14BD1A1237	PENCHALA SACHITH SANTOSH	19
39	14BD1A1238	PESARAMELLI SUPRIYA	44
40	14BD1A1239	PINNINTI VEENA	28
41	14BD1A1240	R DEEPTI	28
42	14BD1A1241	R RHICHA SINGH	41
43	14BD1A1242	RACHURI SANDEEP	35
44	14BD1A1243	RESHMI VINOD	35
45	14BD1A1244	RESHAM JAISWAL	31
46	14BD1A1245	S SAI KUMAR	32
47	14BD1A1246	S SINDHURA	37
48	14BD1A1247	SAADIA TABASSUM	43
49	14BD1A1248	SEEMAKURTHY SAI SRUTHI	44
50	14BD1A1249	SHASHANK NETHA GUTTHI	18
51	14BD1A1250	SHESHIT KARTHIKEYA AAKULA	28
52	14BD1A1251	SNEHA BIDARKAR	37
53	14BD1A1252	SOUMYA GUPTA	26
54	14BD1A1253	SUSHMITHA A	44
55	14BD1A1254	SWAMY SHIVANAND	34
56	14BD1A1255	TADAKAMALLA SUMANTH	15
57	14BD1A1256	V NIRMAL SRINITHYA	41
58	14BD1A1257	VADUGULA PAVAN KUMAR	44
59	14BD1A1259	VONGUR AKSHITHA	43
		sum	2000
		avg	36.363636

no. of students scored more than target %	22
no. of students present	58
Percentage of students scored more than target %	38%
Attainment level	**1**

Attainment Leve	Percentage
1	<40%
2	40-60%
3	>60%

Keshav Memorial Institute of Technology

Department of Information Technology

Course Outcome Attainment

Name of the faculty : **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: **IT** Exam **Overall**

Subject: **Data Structures** Year:**II** Semester:**I**

Course Outco	1st Internal Exam	2nd Internal Exam	University Exam	OVERALL
CO1	2.00		1	1.25
CO2	2.00		1	1.25
CO3	2.00	3	1	1.375
CO4		3	1	1.5
CO5		3	1	1.5

Attainment level of Course Outcomes

	Course Outcomes	Attainment Level
CO1	Describe the basic data structures such as arrays, linked lists, stacks and queues	1.25
CO2	Explain the abstract properties of various data structures such as stacks, queues, lists, trees and graphs	1.25
CO3	Apply Algorithms for solving problems like sorting, searching, insertion and deletion of data.	1.375
CO4	use data structure concepts for realistic problems	1.5
CO5	Analyze performance of algorithms	1.5

Average	1.4

Overall course attainment level 1

Faculty Signature

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Information Technology

Program Outcome Attainment

Name of Faculty: **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: **IT** Year:**II** Semester:**I**

Subject: **Data Structures**

Course outcome attainment

CO	Mid-1	Mid-2	AVG	Univ	DIRECT	INDIRECT	OVERALL
CO1	2		2	1	1.25	2.76	1.552
CO2	2		2	1	1.25	2.72	1.544
CO3	2	3	2.5	1	1.375	2.7	1.64
CO4		3	3	1	1.5	2.68	1.736
CO5		3	3	1	1.5	2.64	1.728
CO6							
attainment			**2.5**	**1**	**1.38**	**2.70**	**1.64**

CO-PO mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2		2									
CO2	2	2	2	2								
CO3	2		2									
CO4		1	3									
CO5	2	3	1	2	2							
CO6												
average	**2**	**2**	**2**	**2**	**2**							
attainment	**1.10**	**1.10**	**1.10**	**1.10**	**1.10**							

Faculty Head of the Deapartment

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

Department of Information Technology

Program Specific Outcome Attainment

Name of Faculty: **Mr Neil Gogte** Academic Year: **2015-16**

Branch & Section: **IT** Year:**II** Semester:**I**

Subject: **Data Structures**

Course outcome attainment

CO	IstMid	IIndMid	Univ	DIRECT	INDIRECT	OVERALL
CO1	2		1	1.25	2.76	1.552
CO2	2		1	1.25	2.72	1.544
CO3	2	3	1	1.375	2.7	1.64
CO4		3	1	1.5	2.68	1.736
CO5		3	1	1.5	2.64	1.728
attainment				**1.38**	**2.70**	**1.64**

CO-PSO mapping

	PSO1	PSO2
CO1	2	1
CO2	2	3
CO3	2	2
CO4	3	2
CO5	2	2
average	**2.2**	**2**
attainment	**1.10**	**1.10**

Faculty Head of the Deapartment

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
NARAYANGUDA HYDERABAD
LEARNING OUTCOMES ANALYSIS

Course Title : B.Tech Branch : IT Class : II-I AY: 2015-16
Faculty Name : Mr Neil Gogte Subject: Data Structures

For each item please indicate your level of agreement with the following statement by choosing a score between 1 and 3. (1 -Poor 2-Average 3-Excellent)

QUESTIONAIRE

1. Can you describe the basic data structures such as arrays, linked lists, stacks and queues
2. Can you explain the abstract properties of various data structures such as stacks, queues, lists, trees and graphs
3. Can you apply algorithms for solving problems like sorting, searching, insertion and deletion of data?
4. Explain data structures concept for realistic problems
5. Can you analyze performance of algorithms

SNO	QUESTIONAIRE				
	1	2	3	4	5
1	3	3	3	3	3
2	3	3	3	3	2
3	3	2	2	3	2
4	2	1	2	2	1
5	3	3	3	2	2
6	3	3	3	2	2
7	2	3	3	2	3
8	3	3	3	3	3
9	3	3	3	2	3
10	2	3	3	2	3
11	3	3	3	3	3
12	2	2	2	2	2
13	3	2	3	3	2
14	3	3	3	3	3
15	3	3	3	3	3
16	3	3	3	3	3
17	2	2	2	2	2
18	3	3	3	3	3
19	3	3	3	3	3
20	2	3	2	3	2
21	3	3	3	3	3
22	3	3	3	3	3
23	3	3	3	3	3
24	3	3	2	3	2
25	3	3	3	3	3
26	3	3	3	2	2
27	3	3	3	3	3
28	3	3	3	3	3

29	3	3	3	3	3
30	3	3	3	3	3
31	2	2	2	2	2
32	3	3	3	3	3
33	2	2	1	1	3
34	3	3	3	3	3
35	3	3	3	3	3
36	3	2	3	3	2
37	3	3	2	3	3
38	3	3	3	3	3
39	2	3	2	2	2
40	3	3	3	3	3
41	2	3	3	3	3
42	3	3	3	3	3
43	3	3	3	3	3
44	3	2	3	2	2
45	3	2	2	3	3
46	2	2	2	2	2
47	3	3	2	3	3
48	2	2	3	2	2
49	3	2	2	3	3
50	3	3	3	3	3

TOTAL	138	136	135	134	132
AVG	2.76	2.72	2.7	2.68	2.64

Overall **2.70**

FACULTY **DIRECTOR** **PRINCIPAL**

Attendance Register

YEAR	II YEAR	SEM	I SEM		BRANCH		IT		SECTION		A	
SUBJECT NAME			DATA STRUCTURES								DS	
FORTNIGHT ATTENDANCE		FROM	Jun 13, 2016	Jul 1, 2016	Jul 16, 2016	Aug 1, 2016	Aug 16, 2016	Sep 1, 2016	Sep 16, 2016	Oct 1, 2016	TOTAL	% PER
		TO	Jun 30, 2016	Jul 15, 2016	Jul 31, 2016	Aug 15, 2016	Aug 31, 2016	Sep 15, 2016	Sep 30, 2016	Oct 15, 2016		
HT NO	ME OF THE STUDENT	CLASSES HELD	16	14	9	8	4	0	11	4	66	
15BD1A1201	A AVINASH GOUD		14	13	8	7	2	0	8	4	56	85%
15BD1A1202	A SAHITYA		14	10	6	7	3	0	9	4	53	80%
15BD1A1203	AKULA AKHILA		16	14	9	7	1	0	9	4	60	91%
15BD1A1204	AMRUTHAKAVI DIVYASREE		14	13	9	8	2	0	10	3	59	89%
15BD1A1205	ANNAPARTHI VYSHNAVI		11	10	8	7	0	0	8	4	48	73%
15BD1A1206	APOORVA RAMARAPU		15	13	9	8	1	0	9	4	59	89%
15BD1A1207	BHARGAVI NAINI		16	13	9	8	2	0	10	4	62	94%
15BD1A1208	BYROSU KISHAN		13	13	9	8	3	0	11	4	61	92%
15BD1A1209	CH D STAN ROBINS		11	10	6	6	0	0	8	4	45	68%
15BD1A1210	CHAPALAMADUGU JANARDHAN NITIN		11	10	6	8	0	0	8	4	47	71%
15BD1A1211	CHEGI REDDY ABHAY SIMHA REDDY		11	10	6	7	0	0	8	4	46	70%
15BD1A1212	CHENNOJU NAGARJUNA		13	14	9	7	3	0	8	4	58	88%
15BD1A1213	CHEVVA KUSHAL		16	14	9	8	2	0	11	4	64	97%
15BD1A1214	CHIKILE YUVA MAITHRI		16	14	9	8	2	0	11	4	64	97%
15BD1A1215	DIVYA GATTANI		16	14	9	8	4	0	11	3	65	98%
15BD1A1216	DUDDALA KARTHIK		16	14	9	8	4	0	11	4	66	100%
15BD1A1217	GODAVARTHI SAI SAHITHI		15	14	9	7	3	0	9	4	61	92%
15BD1A1218	GUDUR NIKHIL SARMA		12	10	8	6	1	0	8	4	49	74%
15BD1A1219	HARSH DESAI		12	11	9	8	1	0	9	4	54	82%
15BD1A1220	HEENA		15	12	6	7	2	0	8	4	54	82%
15BD1A1221	HIRDESH ARORA		15	14	9	8	3	0	10	3	62	94%
15BD1A1222	ISHA KHAKHAR		16	14	9	8	4	0	10	4	65	98%
15BD1A1223	JAIDI SRAVANI		13	13	8	7	3	0	10	4	58	88%
15BD1A1224	JANAMANCHI GANESH		15	14	9	8	3	0	10	4	63	95%
15BD1A1225	K R SPANDANA		15	13	9	8	2	0	11	3	61	92%
15BD1A1226	K SAI KALYAN		15	11	9	8	4	0	9	4	60	91%
15BD1A1227	K.SWAPNIL		11	10	7	7	1	0	8	4	48	73%
15BD1A1228	KALLURI ANVITHA REDDY		14	13	9	8	4	0	9	4	61	92%
15BD1A1229	KANCHARLA NISHAL CHAKRAVARTHY		15	13	9	8	4	0	8	3	60	91%
15BD1A1230	KANDIKONDA BALA HARSHITH		15	13	8	6	3	0	9	4	58	88%
15BD1A1231	KANDUKURI REENA		12	10	8	7	0	0	11	4	52	79%
15BD1A1232	KASIREDDY SANJANA REDDY		13	14	9	8	4	0	8	4	60	91%
15BD1A1233	KATIPALLY RAJESH		15	12	9	8	4	0	11	4	63	95%
15BD1A1234	KOMATI EFRAIM RAJ		12	10	8	8	2	0	9	4	53	80%

15BD1A1235	KOMURAVELLI SAI NIKHIL	15	14	9	8	3	0	11	4	64	97%
15BD1A1237	KUNCHAM MADHU KUMAR	14	14	7	7	1	0	10	4	57	86%
15BD1A1238	MAHANKALI GAYATRI	13	14	9	8	2	0	11	4	61	92%
15BD1A1239	MARUKUKULA PAVAN SAI	16	12	9	7	2	0	9	4	59	89%
15BD1A1240	MARYADA SUCHET REDDY	16	13	9	7	3	0	10	3	61	92%
15BD1A1241	MUKUNDA VANI SRIVAISHNAVI	14	14	9	8	4	0	10	4	63	95%
15BD1A1242	N SAI SAMYUKTHA	16	14	9	8	4	0	10	3	64	97%
15BD1A1243	NALLAGONDA VENKATA HARI KRISHN	12	13	9	8	1	0	9	4	56	85%
15BD1A1244	NAVYA TUBATI	16	14	9	8	4	0	11	4	66	100%
15BD1A1245	NENAVATH MAHESH	13	13	9	8	4	0	11	4	62	94%
15BD1A1246	NIMMAGUDEM SRI VARSHA	16	13	8	7	3	0	9	4	60	91%
15BD1A1247	P SURYA TEEJA	11	10	6	7	0	0	8	4	46	70%
15BD1A1248	PAMULA SAI AVINASH	14	14	9	8	3	0	11	4	63	95%
15BD1A1249	PASUPULETI SHIVA SAI DEEPTHI	15	12	9	8	4	0	8	4	60	91%
15BD1A1250	PONUGOTI PREETHAM	14	13	7	7	2	0	9	4	56	85%
15BD1A1252	SHASAM RANGA PAVAN	14	14	9	8	4	0	10	4	63	95%
15BD1A1253	T V PRANAV VAMSHA TILAK	11	10	6	7	0	0	8	4	46	70%
15BD1A1254	TALLURI ABHIJIT	16	14	9	8	3	0	11	3	64	97%
15BD1A1255	THATIKONDA NIHAR	15	14	9	8	4	0	9	3	62	94%
15BD1A1257	V AKSHAY REDDY	11	10	8	7	0	0	8	4	48	73%
15BD1A1258	VARUN SAXENA	14	13	9	8	1	0	9	4	58	88%
15BD1A1259	VEMULA SRINITH REDDY	11	10	9	7	0	0	8	4	49	74%
15BD1A1260	ZOHEBUDDIN KHAN	12	10	8	7	0	0	8	4	49	74%