STEP TOWARDS SUCCESS

AKASH'S

'Guru Gobind Singh Indra Prastha University Series

SOLVED PAPERS

[PREVIOUS YEARS SOLVED QUESTION PAPERS]

[B.Tech]
THIRD SEMESTER
Data Structure
(CIC-209)

Rs.81.00/-

AKASH BOOKS
NEW DELHI

Third Semester					
Group	**Paper Code**	**Paper**	**L**	**P**	**Credits**
Theory Papers					
ES	ES-201	Computational Methods	4		4
HS/MS	HS-203	Indian Knowledge System*	2		2
PC	CIC-205	Discrete Mathematics	4		4
PC	ECC-207	Digital Logic and Computer Design	4		4
PC	CIC-209	Data Structures	4		4
PC	CIC-211	Object-Oriented Programming using C++	4		4
Practical/Viva Voce					
ES	ES-251	Computational Methods Lab		2	1
PC	ECC-253	Digital Logic and Computer Design Lab		2	1
PC	CIC-255	Data Structures Lab		2	1
PC	CIC-257	Object-Oriented Programming using C++ Lab		2	1
Total			**22**	**8**	**26**

***NUES:** All examinations to be conducted by the concerned teacher as specified in the detailed syllabus of the paper.

© ALL RIGHTS RESERVED WITH THE PUBLISHERS

(All rights resolved, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher. Any violation thereof shall lead to legal recourse without any further notice.)

Published by :
AKASH BOOKS
4278/3, Ansari Road, Darya Ganj,
New Delhi-110002
Ph. : 0-9818257423, 9971086337

Note : As publishers, we a re committed to serve the student community with the best of our resources. We take every care to eliminate errors during the course of editing printing of books. Therefore, we state that authors, publishers and sellers should not be held responsible for any mistake that might have crept in advertently.

Printed at : Delhi

SYLLABUS
(ACADEMIC SESSION 2014-15)

DATA STRUCTURES

INSTRUCTIONS TO PAPER SETTERS **Maximum Marks: 75**

1. Question No. 1 should be compulsory and cover the entire syllabus. This question should have objective or short answer type questions. It should be of 25 marks.

2. Apart from Question No. 1, the rest of the paper shall consist of four units as per the syllabus. Every unit should have two questions. However, the student may be asked to attempt only 1 question from each unit. Each question should be 12.5 marks.

UNIT-I

Introduction to programm ing methodologies and design of algorithms. Abstract Data Type, array, array organization, sparse array. Stacks and Stack ADT, Stack Manipulation, Prefix, infix and postfix expressions, their interconversion and expression evaluation. Queues and Queue ADT, Queue manipulation. General Lists and List ADT, List manipulations, Single, double and circular lists. **[T1,T2] [No. of hrs. 12]**

UNIT-II

Trees, Properties of Trees, Binary trees, Binary Tree traversal, Tree manipulation algorithms, Expreession trees and their usage, binary search trees, AVL Trees, Heaps and their implementation. **[T1,T2] [No. of hrs. 12]**

UNIT-III

Multiway trees, B-Trees, 2-3 trees, 2-3-4 trees, B* and B+ Trees. Graphs, Graph representation, Graph traversal. **[T1,T2] [No. of hrs. 12]**

UNIT-IV

Sorting concept, order, stability, Selection sorts (straight, heap), insertion sort (Straight Insertion, Shell sort), Exchange Sort (Bubble, quicksort), Merge sort (only 2-way merge sort). Searching – List search, sequential search, binary search, hashing concepts, hashing methods (Direct, subtraction, modulo-division, midsquare, folding, pseudorandom hashing), collision resolution (by open addressing: linear probe, quadratic probe, pseudorandom collision resolution, linked list collision resolution), Bucket hashing. **[T1,T2] [No. of hrs. 12]**

NEW TOPICS ADDED FROM ACADEMIC SESSION 2022-23
THIRD SEMESTER [B.TECH]
DATA STRUCTURE (CIC-209)

Q.1. What is the Time – Space Trade Off in Algorithm & also explain its type.

Ans. Time-Space Trade-Off in Algorithms

A trade off is a situation where one thing increases and another thing decreases. It is a way to solve a problem in:

- Either in less time and by using more space, or
- In very little space by spending a long amount of time.

The best Algorithm is that which helps to solve a problem that requires less space in memory and also takes less time to generate the output. But in general, it is not always possible to achieve both of these conditions at the same time. The most common condition is an algorithm using a lookup table. This means that the answers to some questions for every possible value can be written down. One way of solving this problem is to write down the entire lookup table, which will let you find answers very quickly but will use a lot of space. Another way is to calculate the answers without writing down anything, which uses very little space, but might take a long time. Therefore, the more time-efficient algorithms you have, that would be less space-efficient.

Types of Space-Time Trade-off

(a) Compressed or Uncompressed data: A space-time trade-off can be applied to the problem of data storage. If data stored is uncompressed, it takes more space but less time. But if the data is stored compressed, it takes less space but more time to run the decompression algorithm. There are many instances where it is possible to directly work with compressed data. In that case of compressed bitmap indices, where it is faster to work with compression than without compression.

(b) Re-Rendering or Stored images: In this case, storing only the source and rendering it as an image would take more space but less time i.e., storing an image in the cache is faster than re-rendering but requires more space in memory.

(c) Smaller code or Loop Unrolling: Smaller code occupies less space in memory but it requires high computation time that is required for jumping back to the beginning of the loop at the end of each iteration. Loop unrolling can optimize execution speed at the cost of increased binary size. It occupies more space in memory but requires less computation time.

(d) Lookup tables or Recalculation: In a lookup table, an implementation can include the entire table which reduces computing time but increases the amount of memory needed. It can recalculate i.e., compute table entries as needed, increasing computing time but reducing memory requirements.

Q.2. What is algorithm and why analysis of it is important?

Ans. In the analysis of the algorithm, it generally focused on CPU (time) usage, Memory usage, Disk usage, and Network usage. All are important, but the most concern is about the CPU time. Be careful to differentiate between:

- **Performance:** How much time/memory/disk/etc. is used when a program is run.

This depends on the machine, compiler, etc. as well as the code we write.

- **Complexity:** How do the resource requirements of a program or algorithm scale, i.e. what happens as the size of the problem being solved by the code gets larger.

Note: Complexity affects performance but not vice-versa.

Algorithm Analysis:

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Why Analysis of Algorithms is important?

- To predict the behavior of an algorithm without implementing it on a specific computer.
- It is much more convenient to have simple measures for the efficiency of an algorithm than to implement the algorithm and test the efficiency every time a certain parameter in the underlying computer system changes.
- It is impossible to predict the exact behavior of an algorithm. There are too many influencing factors.
- The analysis is thus only an approximation; it is not perfect.
- More importantly, by analyzing different algorithms, we can compare them to determine the best one for our purpose.

Q.3. Explain the different types of Algorithm analysis?

Ans. Types of Algorithm Analysis are as follows:

(a) Best case: Define the input for which algorithm takes less time or minimum time. In the best case calculate the lower bound of an algorithm. Example: In the linear search when search data is present at the first location of large data then the best case occurs.

(b) Worst Case: Define the input for which algorithm takes a long time or maximum time. In the worst calculate the upper bound of an algorithm. Example: In the linear search when search data is not present at all then the worst case occurs.

(c) Average case: In the average case take all random inputs and calculate the computation time for all inputs.

And then we divide it by the total number of inputs.

Average case = all random case time / total no of case

Q.4. What is the Asymptotic Analysis of Algorithms? Explain the Different types of Asymptotic Notations?

Ans. Asymptotic Analysis - Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Usually, the time required by an algorithm falls under three types –

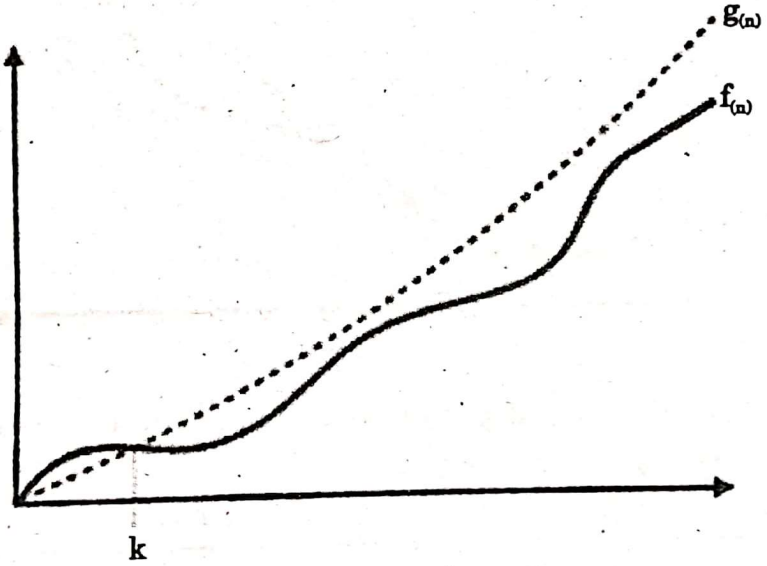
- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution.

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

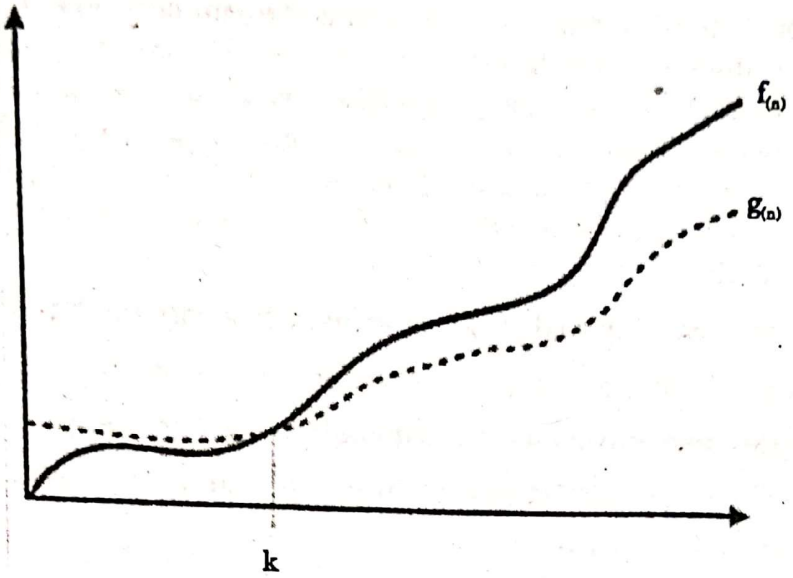
Big Oh Notation, O : The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



For example, for a function f(n)

$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$

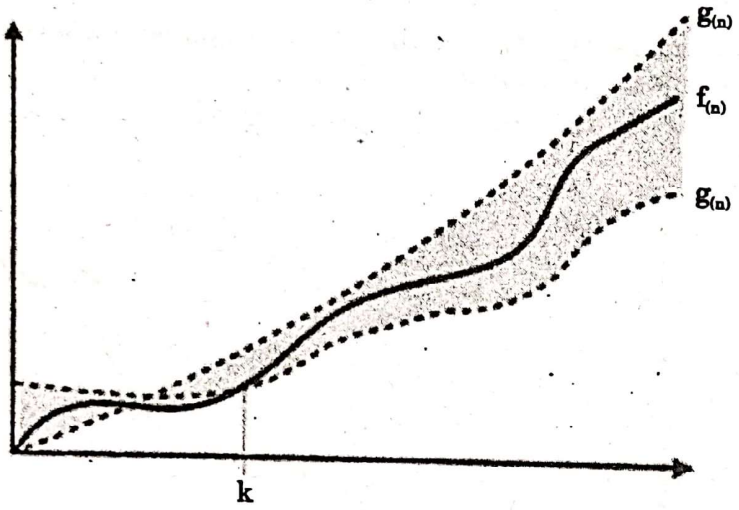
Omega Notation, Ω : The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



For example, for a function f(n)

$\Omega(f(n)) \geq \{ g(n) :$ there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0. \}$

Theta Notation, θ : The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows –

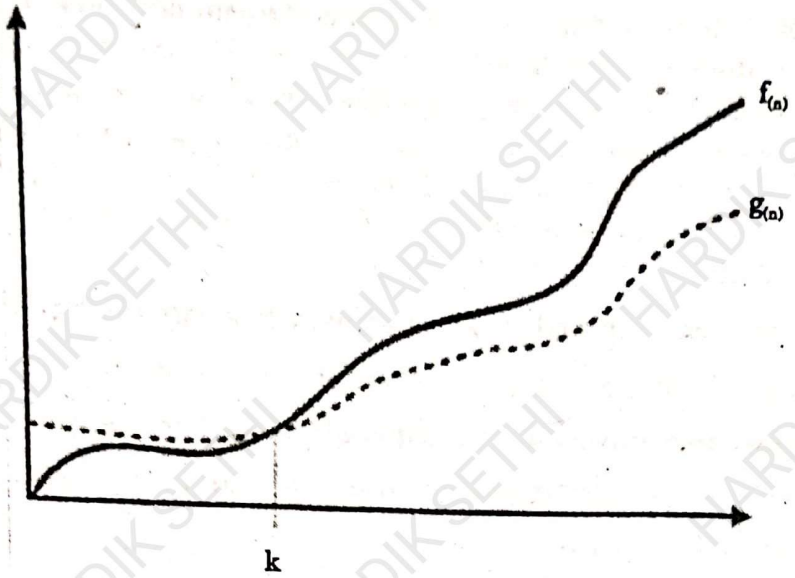


$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0. \}$

Common Asymptotic Notations

Following is a list of some common asymptotic notations –

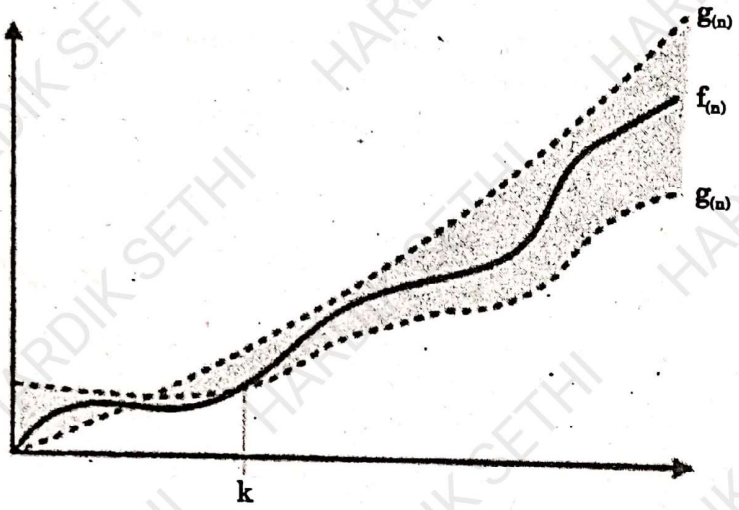
constant	–	O(1)
logarithmic	–	O(log n)
linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$



For example, for a function f(n)

$\Omega(f(n)) \geq \{ g(n) :$ there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0. \}$

Theta Notation, θ : The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows –



$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0. \}$

Common Asymptotic Notations

Following is a list of some common asymptotic notations –

constant	–	O(1)
logarithmic	–	O(log n)
linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Q.5. What is the Sparse Matrix? Explain the Array & Linked List representation of Sparse Matrix.

Ans. A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m × n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Example:

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples- (Row, Column, value).**

Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation 2. Linked list representation

Method 1: Using Arrays: 2D array is used to represent a sparse matrix in which there are three rows named as

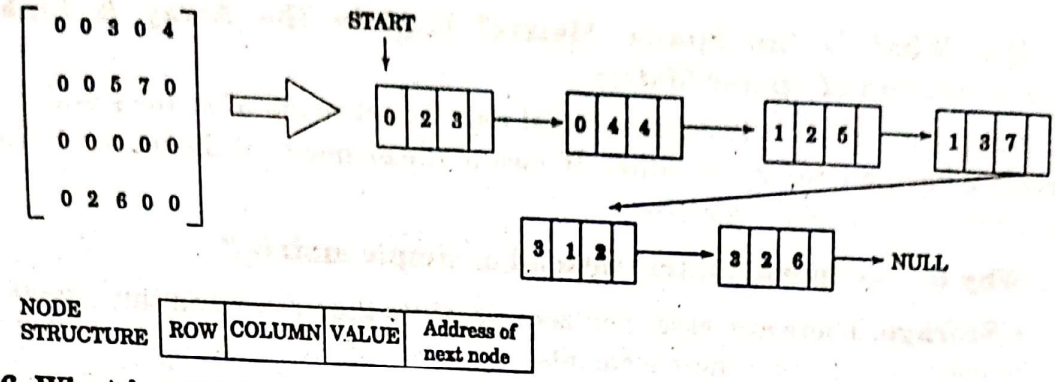
- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row, column)

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Method 2: Using Linked Lists : In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)
- **Next node:** Address of the next node



Q.6. What is a Disjoint set data structure & also explain its operations?

Ans. Two sets are called disjoint sets if they don't have any element in common, the intersection of sets is a null set.

A data structure that stores non overlapping or disjoint subset of elements is called disjoint set data structure. The disjoint set data structure supports following operations:

- Adding new sets to the disjoint set.
- Merging disjoint sets to a single disjoint set using Union operation.
- Finding representative of a disjoint set using Find operation.
- Check if two sets are disjoint or not.

Consider a situation with a number of persons and the following tasks to be performed on them:

- Add a new friendship relation, i.e. a person x becomes the friend of another person y i.e adding new element to a set.
- Find whether individual x is a friend of individual y (direct or indirect friend)

A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

- **Find:** Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
- **Union:** Join two subsets into a single subset. Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union.

Q.7. What is the Polynomial?

Ans. A polynomial p(x) is the expression in variable x which is in the form ($ax^n + bx^{n-1} + ... + jx + k$), where a, b, c ...., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:

- one is the coefficient
- other is the exponent

Example:

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

Points to keep in Mind while working with Polynomials:

- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- Additional terms having equal exponent is possible one
- The storage allocation for each term in the polynomial must be done in ascending and descending order of their exponent

Q.8. How to represent a Polynomial?

Ans. Polynomial can be represented in the various ways. These are:

- By the use of arrays
- By the use of Linked List

Representation of Polynomials Using Arrays

There may arise some situation where you need to evaluate many polynomial expressions and perform basic arithmetic operations like addition and subtraction with those numbers. For this, you will have to get a way to represent those polynomials. The simple way is to represent a polynomial with degree 'n' and store the coefficient of n+1 terms of the polynomial in the array. So every array element will consist of two values:

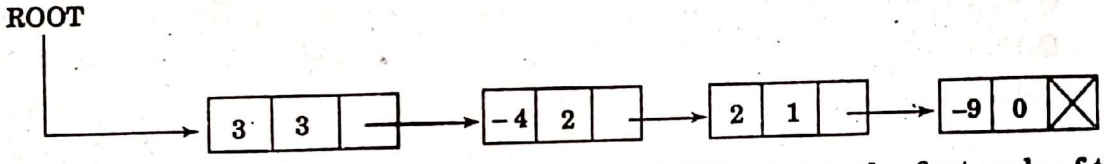
- Coefficient and
- Exponent

Polynomial representation using Linked List

The linked list can be used to represent a polynomial of any degree. Simply the information field is changed according to the number of variables used in the polynomial. If a single variable is used in the polynomial the information field of the node contains two parts: one for coefficient of variable and the other for degree of variable. Let us consider an example to represent a polynomial using linked list as follows:

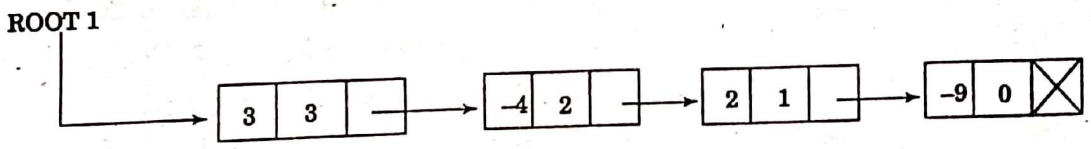
Polynomial: $3x^3 - 4x^2 + 2x - 9$

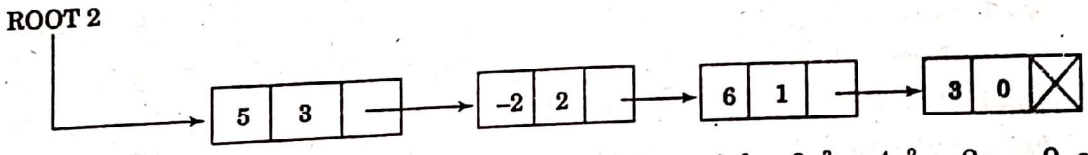
Linked List:



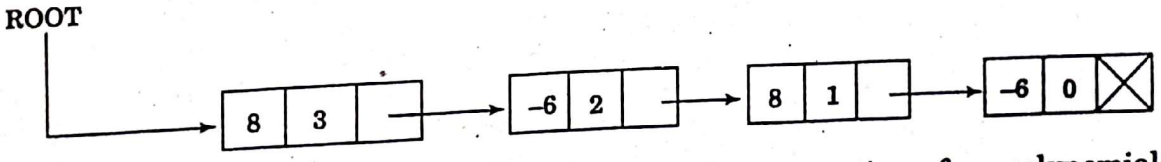
In the above linked list, the external pointer 'ROOT' point to the first node of the linked list. The first node of the linked list contains the information about the variable with the highest degree. The first node points to the next node with next lowest degree of the variable.

Representation of a polynomial using the linked list is beneficial when the operations on the polynomial like addition and subtractions are performed. The resulting polynomial can also be traversed very easily to display the polynomial.





The above two linked lists represent the polynomials, $3x^3 - 4x^2 + 2x - 9$ and $5x^3 - 2x^2 + 6x + 3$ respectively. If both the polynomials are added then the resulting linked will be:



The linked list pointer ROOT gives the representation for polynomial, $8x^3 - 6x^2 + 8x - 6$.

UNIT - I

Q.1. Refer to Q.1 (a) of First Term Examination 2014 (Pg. No. 1-2014)
Q.2. Refer to Q.1 (b) of First Term Examination 2014 (Pg. No. 1-2014)
Q.3. Refer to Q.1 (c) of First Term Examination 2014 (Pg. No. 2-2014)
Q.4. Refer to Q.1 (d) (i),(ii) of First Term Examination 2014 (Pg. No. 2-2014)
Q.5. Refer to Q.2 (a), (b), (c) of First Term Examination 2014 (Pg. No. 3-2014)
Q.6. Refer to Q.3 (a), (b), (c) of First Term Examination 2014 (Pg. No. 4-2014)
Q.7. Refer to Q.4 (a), (b) of First Term Examination 2014 (Pg. No. 7-2014)
Q.8. Refer to Q.1 (b) of End Term Examination 2014 (Pg. No. 18-2014)
Q.9. Refer to Q.2 of End Term Examination 2014 (Pg. No. 19-2014)
Q.10. Refer to Q.1 (d), (e), (f), (h) of First Term Examination 2015 (Pg. No. 1-2015)
Q.11. Refer to Q.2 (b) of First Term Examination 2015 (Pg. No. 6-2015)
Q.12. Refer to Q.3 (a), (b) of First Term Examination 2015 (Pg. No. 7-2015)
Q.13. Refer to Q.4 (a), (b), (c) of First Term Examination 2015 (Pg. No. 8-2015)
Q.14. Refer to Q.1 (e) of End Term Examination 2015 (Pg. No. 18-2015)
Q.15. Refer to Q.2 (a), (b) of End Term Examination 2015 (Pg. No. 18-2015)
Q.16. Refer to Q.1 (c) of End Term Examination 2015 (Pg. No. 30-2015)
Q.17. Refer to Q.1 (a), (b), (c) of First Term Examination 2016 (Pg. No. 1-2016)
Q.18. Refer to Q.2 (a), (b) of First Term Examination 2016 (Pg. No. 2-2016)
Q.19. Refer to Q.3 (a), (b) of First Term Examination 2016 (Pg. No. 3-2016)
Q.20. Refer to Q.1 (e) of End Term Examination 2016 (Pg. No. 15-2016)
Q.21. Refer to Q.2 (b) of End Term Examination 2016 (Pg. No. 16-2016)
Q.22. Refer to Q.8 (b) of End Term Examination 2016 (Pg. No. 31-2016)
Q.23. Refer to Q.1 (a) of First Term Examination 2017 (Pg. No. 1-2017)
Q.24. Refer to Q.1 (c), (d) of First Term Examination 2017 (Pg. No. 2-2017)
Q.25. Refer to Q.2 (a), (b) of First Term Examination 2017 (Pg. No. 3-2017)
Q.26. Refer to Q.3 (a) of First Term Examination 2017 (Pg. No. 3-2017)
Q.27. Refer to Q.1 (a), (b), (c) of End Term Examination 2017 (Pg. No. 11-2017)
Q.28. Refer to Q.2 of End Term Examination 2017 (Pg. No. 16-2017)
Q.29. Refer to Q.3 of End Term Examination 2017 (Pg. No. 17-2017)
Q.30. Refer to Q.1 (a), (b), (c), (d) of First Term Examination 2018 (Pg. No. 1-2018)
Q.31. Refer to Q.2 (a), (b) of First Term Examination 2018 (Pg. No. 2-2018)
Q.32. Refer to Q.3 (a) of First Term Examination 2018 (Pg. No. 3-2018)
Q.33. Refer to Q.1 (b), (c), (e), (f), (g) of End Term Examination 2018 (Pg.No.9,10, 11-2018)
Q.34. Refer to Q.2 (a), (b) of End Term Examination 2018 (Pg. No. 12-2018)
Q.35. Refer to Q.3 (a), (b) of End Term Examination 2018 (Pg. No. 13-2018)
Q.36. Refer to Q.4 (a) of End Term Examination 2018 (Pg. No. 15-2018)

UNIT - II

Q.1. Refer to Q.1 (e) of First Term Examination 2014 (Pg. No. 3-2014)
Q.2. Refer to Q.1 (a) of End Term Examination 2014 (Pg. No. 18-2014)
Q.3. Refer to Q.4 (a), (b) of End Term Examination 2014 (Pg. No. 19-2014)
Q.4. Refer to Q.5 of End Term Examination 2014 (Pg. No. 21-2014)
Q.5. Refer to Q.7 of End Term Examination 2014 (Pg. No. 22-2014)
Q.6. Refer to Q.1 (a), (b), (c) of First Term Examination 2015 (Pg. No. 1-2015)
Q.7. Refer to Q.1 (g) of First Term Examination 2015 (Pg. No. 2-2015)
Q.8. Refer to Q.2 (a) of First Term Examination 2015 (Pg. No. 2-2015)
Q.9. Refer to Q.1 (a), (b) of Second Term Examination 2015 (Pg. No. 10-2015)
Q.10. Refer to Q.2 (a) of Second Term Examination 2015 (Pg. No. 11-2015)
Q.11. Refer to Q.1 (b) of End Term Examination 2015 (Pg. No. 16-2015)
Q.12. Refer to Q.3 of End Term Examination 2015 (Pg. No. 23-2015)
Q.13. Refer to Q.5 of End Term Examination 2015 (Pg. No. 26-2015)
Q.14. Refer to Q.8 (b) of End Term Examination 2015 (Pg. No. 30-2015)
Q.15. Refer to Q.1 (d), (e) of First Term Examination 2016 (Pg. No. 1-2016)
Q.16. Refer to Q.4 (a), (b) of First Term Examination 2016 (Pg. No. 4-2016)
Q.17. Refer to Q.1 (c) of End Term Examination 2016 (Pg. No. 11-2016)
Q.18. Refer to Q.4 of End Term Examination 2016 (Pg. No. 18-2016)
Q.19. Refer to Q.5 of End Term Examination 2016 (Pg. No. 19-2016)
Q.20. Refer to Q.6 of End Term Examination 2016 (Pg. No. 25-2016)
Q.21. Refer to Q.1 (b), (e) of First Term Examination 2017 (Pg. No. 1, 2-2017)
Q.22. Refer to Q.3 (b) of First Term Examination 2017 (Pg. No. 4-2017)
Q.23. Refer to Q.4 (a), (b) of First Term Examination 2017 (Pg. No. 7-2017)
Q.24. Refer to Q.1 (d), (f), (g), (h), (i) of End Term Examination 2017 (Pg. No. 12, 13,14,15-2017)
Q.25. Refer to Q.4 of End Term Examination 2017 (Pg. No. 17-2017)
Q.26. Refer to Q.5 of End Term Examination 2017 (Pg. No. 23-2017)
Q.27. Refer to Q.6 of End Term Examination 2017 (Pg. No. 25-2017)
Q.28. Refer to Q.1 (e) of First Term Examination 2018 (Pg. No. 2-2018)
Q.29. Refer to Q.3 (b) of First Term Examination 2018 (Pg. No. 4-2018)
Q.30. Refer to Q.4 (a), (b) of First Term Examination 2018 (Pg. No. 6-2018)
Q.31. Refer to Q.1 (d) of End Term Examination 2018 (Pg. No. 10-2018)
Q.32. Refer to Q.1 (i), (j) of End Term Examination 2018 (Pg. No. 11, 12-2018)
Q.33. Refer to Q.4 (b) of End Term Examination 2018 (Pg. No. 15-2018)
Q.34. Refer to Q.5 (a), (b) of End Term Examination 2018 (Pg. No. 19-2018)
Q.35. Refer to Q.7 (b) of End Term Examination 2018 (Pg. No. 24-2018)

UNIT - III

Q.1. Refer to Q.1 (c), (d), (e) of End Term Examination 2014 (Pg. No. 18-2014)
Q.2. Refer to Q.3 (a), (b) of End Term Examination 2014 (Pg. No. 19-2014)
Q.3. Refer to Q.6 of End Term Examination 2014 (Pg. No. 21-2014)
Q.4. Refer to Q.8 of End Term Examination 2014 (Pg. No. 24-2014)
Q.5. Refer to Q.1 (d), (e) of Second Term Examination 2015 (Pg. No. 10-2015)
Q.6. Refer to Q.3 (a), (b) of Second Term Examination 2015 (Pg. No. 13-2015)
Q.7. Refer to Q.4 (a), (b) of Second Term Examination 2015 (Pg. Nc. 14-2015)
Q.8. Refer to Q.1 (c) of End Term Examination 2015 (Pg. No. 17-2015)
Q.9. Refer to Q.4 of End Term Examination 2015 (Pg. No. 24-2015)
Q.10. Refer to Q.7 (a), (b), (c) of End Term Examination 2015 (Pg. No. 29-2015)
Q.11. Refer to Q.8 (a) of End Term Examination 2015 (Pg. No. 30-2015)
Q.12. Refer to Q.1 (a) of End Term Examination 2016 (Pg. No. 10-2016)
Q.13. Refer to Q.3 (b) of End Term Examination 2016 (Pg. No. 17-2016)
Q.14. Refer to Q.7 (b) of End Term Examination 2016 (Pg. No. 30-2016)
Q.15. Refer to Q.8 (a) of End Term Examination 2016 (Pg. No. 30-2016)
Q.16. Refer to Q.1 (e) of End Term Examination 2017 (Pg. No. 12-2017)
Q.17. Refer to Q.8 of End Term Examination 2017 (Pg. No. 29-2017)
Q.18. Refer to Q.9 of End Term Examination 2017 (Pg. No. 31-2017)
Q.19. Refer to Q.1 (a) of End Term Examination 2018 (Pg. No. 9-2018)
Q.20. Refer to Q.8 (a), (b) of End Term Examination 2018 (Pg. No. 26-2018)
Q.21. Refer to Q.9 (a), (b) of End Term Examination 2018 (Pg. No. 30-2018)

UNIT - IV

Q.1. Refer to Q.1 (c) of Second Term Examination 2015 (Pg. No. 10-2015)
Q.2. Refer to Q.2 (b) of Second Term Examination 2015 (Pg. No. 13-2015)
Q.3. Refer to Q.1 (a), (d) of End Term Examination 2015 (Pg. No. 16-2015)
Q.4. Refer to Q.6 of End Term Examination 2015 (Pg. No. 26-2015)
Q.5. Refer to Q.1 (b) of End Term Examination 2016 (Pg. No. 10-2016)
Q.6. Refer to Q.1 (d) of End Term Examination 2016 (Pg. No. 14-2016)
Q.7. Refer to Q.7 (a) of End Term Examination 2016 (Pg. No. 28-2016)
Q.8. Refer to Q.7 of End Term Examination 2017 (Pg. No. 26-2017)
Q.9. Refer to Q.1 (e) of End Term Examination 2018 (Pg. No. 10-2018)
Q.10. Refer to Q.1 (h) of End Term Examination 2018 (Pg. No. 11-2018)
Q.11. Refer to Q.6 (a), (b) of End Term Examination 2018 (Pg. No. 21-2018)
Q.12. Refer to Q.7 (a) of End Term Examination 2018 (Pg. No. 22-2018)

FIRST TERM EXAMINATION
THIRD SEMESTER (B.TECH) [ETCS-209]
DATA STRUCTURES—SEPT. 2014

Time : 1.30 hrs. **M.M. : 30**

Note: *Attempt any three questions in total including Question No. 1 which is compulsory.*

Q.1.(a) Define a Double ended queue? What are its type?

Ans. A deque is a list in which the element can be inserted or deleted at either end. It is also known as a head-tail linked list, because elements can be added to or removed from either the front (head) or back (tail) end.

Consider the deque shown in fig.

			29	37	45	54	63		
0	1	2	Left=3	4	5	6	Right=7	8	9

42	56						63	27	18
Right=0	1	2	Left=3	4	5	6	Left=7	8	9

Double-ended queues

Basically, there are two variants of double-ended queue.

(1) Input Restricted deque: In this dequeue, insertions can be done at one of the dequeue, while deletion can be done from both ends.

(2) Output Restricted deque: In this dequeue, deletions can be done only at one of the dequeue, while insertion can be done on both ends.

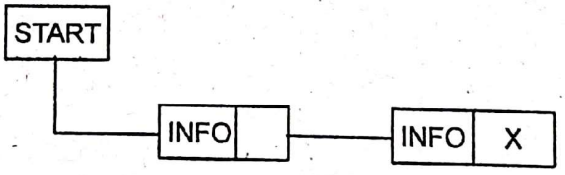
Q.1.(b) What are linear list? What are it types?

An. A linear list is a non-sequential collection of data items called nodes. These nodes in principle are structures containing fields, each node in a linked list has basically two field.

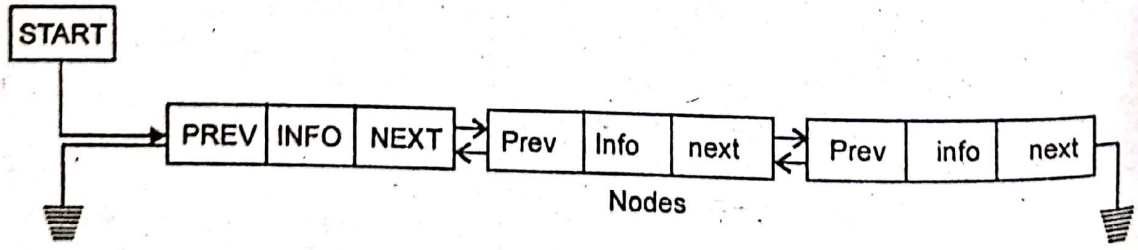
(1) Data field

(2) Link field.

The data field contains an actual value to be stored and processed. And the link field contain the address of the next data item in the linked list.



Single linked list



Doubly linked list

Types of linked list
(1) Singly linked list
(2) Doubly linked list
(3) Circular linked list
(4) Circular doubly linked list

Q.1.(c) Reverse a linked list without using the pointer?

Ans. To reverse a linear linked list, three pointer fields are used. These are tpt, ptr, cpt which hold the address of the previous node, current node and next node. To begin with the address of the first node, which is held in pointer variable FIRST, is assigned to PTR and typt is assigned value NULL.

Algorithm:

Step 1	PTR = FIRST
Step 2	TPT = NULL
Step 3	Repeat step 4 while PTR! = NULL
Step 4	(a) CPT = LINK (PTR)
	(b) LINK (PTR) = TPT
	(c) TPT = PTR
	(d) PTR = CPT
Step 5	STOP.

Q.1.(d)(i) Convert the following in fix expression into a postfix using a stack.

$$\{a + (b - c)\} * \{(d - e)/(\delta + g - h\}$$

Ans.

	Symbol Scanned	Stack	Express (Post fix)
1.	{	{	a
2.	a	{	a
3.	+	{+	a
4.	(	{+(	ab
5.	b	{+(	ab
6.	–	{+(–	abc
7.	(	{+(–	abc –
8.	)	{+	abc –
9.	}		abc – +
10.	*	*	abc – +
11.	{	*{	abc – +
12.	(	*{(	abc – +
13.	d	*{(	abc – + d
14.	–	*{(–	abc – + d
15.	e	*{(-	abc – + de
16.	)	*{	abc – + de –
17.	/	*{/	abc – + de –
18.	(	*{/(	abc – + de –
19.	δ	*{/(	abc – + de – f
20.	+	*{/(+	abc – + de – f
21.	g	*{/(+	abc – + de – fg

22.	–	*{ :/(+-	abc – + de – δg
23.	h	*{/(+–	abc – + de – fgh
24.	)	*{/	abc – + de – fgh – +
25.	}	–	abc – + de – fgh – + / *

Q.1.(*d*)(*ii*) If you are using C language to implement the heterogenous linked list, what pointer type will you use?

Ans. We use the void pointer to implement the heterogeneous linked list. The heterogenous linked list contain different data types in its nodes and we need a link, pointer, to connect them. Since we can't use ordinary pointer for this, we use the void pointer. Void pointer is a generic pointer type, and capable of storing pointer to any type.

Q.1.(*e*) What are threaded binary Trees? How are threads different from pointers? And how we can determine the next node to which a thread points?

Ans. In a linked representation of any binary tree, half of the entries in the pointer field LEFT and RIGHT will contain null entries. This space may be more efficiently used by replacing the null entries by special pointers called threads. Which points to the nodes higher in the trees. Such trees are called Threaded Tree.

Difference between threads and pointer: Threads in a binary tree must be distinguished from normal pointer. In a graphical representation of a threaded binary tree, threads are shown by dotted lines. In a computer memory, an extra field called tag or flag is used to distinguish a thread from a normal pointer.

Tree can be threaded using one way threading or two way threading. In a one way threading, a thread will appear in the right field of the node and will point to the successor node in the in order traversal of tree; and in two way threading of tree a thread will also appear in the left field of a node and will point to the preceding node in the in order traversal of tree.

Q.2.(*a*) Write an algorithm or pseudo code to extract characters from a given string from a given position for n characters and store it another string which is initialized at run time.

Ans.

```
# include <stdio.h>
# define SIZE 100
    void print_str (char S []);
    void read_str (char S[]);
    main ( )
    {
    char str [SIZE];
    do {
        read_str (Str);
        print_str (Str);
    } while (Str [0]];
}
void read_str (char * S)
{
    int i;
    char c ;
    for (i = 0; (c = get char ( ) ) ! = '\n' ; i ++)
        S [i] = c;
        S[i] = NULL;
```

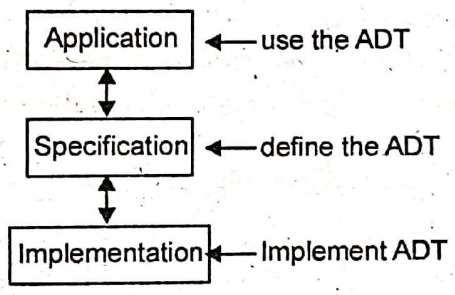
```
}
Void print_str (char*S)
{
    int i;
    for (i = 0; S [i]; i ++)
        putchar (S[i]); putchar ('\n');
        }
```

Q.2.(*b*) What is Abstract Data type and its implementation through data structures?

Ans. Abstract Data Type: Abstract data type is the specifications of the data type which specifies the logical and mathematical model of the delta type.

To implement a fraction data type:

```
Type day strict { int numerator, denomination; fraction; }
main ()
{
  fraction f;
  f. numerator = 1;
  f. denomination = 2;
  ....
  }
```



When we use abstract data type, our program divide into two:

The Application: The part that use Abstract data type.

The Implementation: The part that implement the Abstract data type.

Q.2.(*c*) How will you check the validity of an expression containing nested parenthesis?

Ans. To check the validity of an expression containing nested parenthesis we use the stack. Stack works fine for this. The steps are:

(1) Maintain a stack of characters.

(2) Whenever you find opening braces '(', '{', or '[' push it on the stack.

(3) Whenever you find closing traces ')', '}', ']' check if top of stack is corresponding opening bracket, if yes then pop the stack, else break the loop and return false.

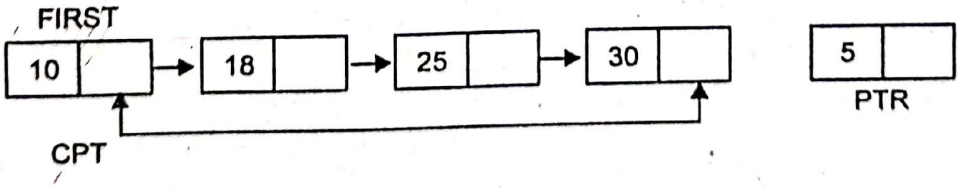
(4) Repeat step 2-3 until end of the string.

Q.3.(*a*) Write a pseudo code/algorithm for

(*i*) insertion at the end of a circular linked list.

(*ii*) Deletion of a node with a given data in a doubly linked list.

Ans.

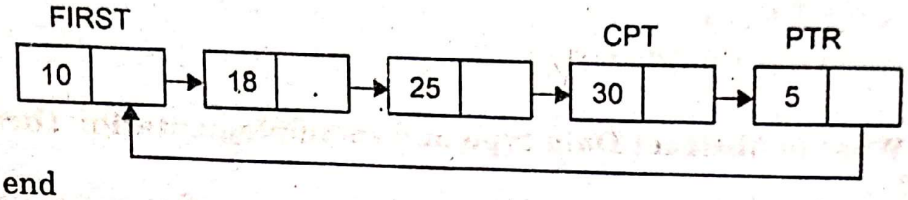


Repeat CPT = Link (CPT) till link (CPT) ≠ FIRST

i.e.

LINK (CPT) = PTR

LINK (PTR) = FIRST



At the end

1. If Avail = NULL then linked list is overflow and STOP.
2. PTR ← AVAIL
   AVAIl ← LINK (AVAIL)
   Read INFO (PTR)
3. CPT ← FIRST
4. Repeat step 5 while LINK (CPT) ! = FIRST
5. CPT ← LINK (CPT)
6. LINK (CPT) ← PTR
7. LINK (PTR) ← FIRST
8. STOP.

(*ii*) If a specific node is given.

1. if FIRST = NULL then write UNDERFLOW and STOP.
2. Read DATA as information of node to be deleted.
3. PTR ← FIRST
4. Repeat step (5) while INFO (PTR) = DATA
5. PTR ← RPT (PTR)
6. CPT ← LPT (PTR)
   TPT ← RPT (PTR)
   RPT (CPT) ← TPT
   LPT (TPT) ← CPT
7. RPT (PTR) ← AVAIL
   AVAIL ← PTR
8. STOP.

Q.3.(*b*) How can we use linked list to store a sparse matrix?

Ans. The most advantage of linked list is manageability of memory and it is very important for large programs which are used to process a lot of data.

To represent a sparse matrix we use a linked list for each row and each column. So its needs two pointers : Next in this row and next in this column.

```
Struct node {
    node * next (01;
    node * next row;
    int row;
    int col;
    int value;
};
Class matrix
{
```

```
    node **colhead; // Head element of columns
    node ** rowhead; // the head element of rows
    int rows cols; // number of rows & columns of matrix;
    operations of the class;
}
```

Q.3.(c) Write a pseudo code/algorithm to remove duplicate element from a list?

Ans. Transverse the list from the head (or start) node. While Traversing, compare each node with its next node. If data of next node is same as current node then delete the next node. Before we delete a node, we need to store next pointer of the node.

Program:

```
    # include < stdio.h>
    # include < stdlib.h>
Struct node
    { int data;
    struct node * next;
};
Void remove Duplicates (Struct node * head)
{
    Struct node * current = head;
    Struct node * next – next;
    if (Current == NULL)
      return;
    While (Current > next! = NULL)
    {
      if (Currnet → 7 data = = current – > next – > data)
    {
    next – next = current – > next – > next;
      free (current _ next);
    }
    else
    {
      current = current – > next;
    }
  }
}
void Push (Struct node ** head – ref, int new-data)
{
    Struct node * new_node = (Struct node*) malloc (size of (Struct node));
      new_node – > next = (*head-ref);
      (*head-ref) = new-node;
    }
    Void printlist (Struct node * node)
    {
      While (node! = NULL)
      {
        printf ("/d", node – > data);
        node = node – > next;
      }
```

```
}
int main ()
{
    struct node * head = NULL;
      Push (& head, 20);
      Push (& head, 13);
      Push (& head, 13);
      Push (& head, 11);
      Push (& head, 11);
      Push (& head, 11);
  print f ("/ln linked list before duplicate removal");
  print List (head);
  remove Duplicates (head);
  printf ("ln linked list after duplicate Removal);
    printlist (head);
    getchar ();
}
```

Q.4.(*a*) Consider a float array APR (2 : 8, – 4 : 1), 6 : 10) whose base address is 200. Find the address of APR (5, – 1, 8]

Ans. APR (2 : 8, – 4 : 1), 6 : 10)

length of three dimensional of APR are

$4 = 8 - 2 + 1 = 7 \quad L_2 = 1 - (-4) + 1 = 6 \quad L_3 = 10 - 6 + 1 = 5$

Accordingly APR contains $L_1 : L_2 : L_3 = 7 \times 6 \times 5 = 210$ elements.

Suppose the programming language store APR in memory in row major order, and Base (APR) = 200 w = 4 words per memory cell.

The effective indices of the subscripts are:

$E_1 = 5 - 2 = 3 \quad E_2 = -1 - (-4) = 3 \quad E_3 = 8 - 6 = 2.$

For row major order, we have:

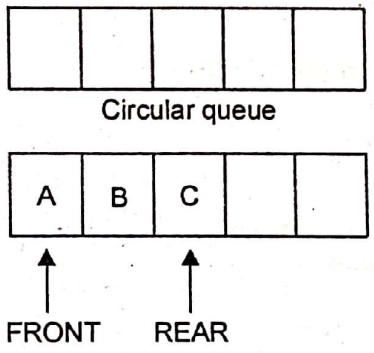
$$E_1L_2 = 3 \times 6 = 18$$
$$E_1L_2 + E_2 = 18 + 3 = 21$$
$$(E_1L_2 + E_2)\,L_3 = 21 \times 5 = 105$$
$$(E_1L_2 + E_3)\,L_3 + E_3 = 105 + 2 = 107$$

Add (APR [5, – 1, 8] = 200 + 4 (107) = 200 + 428 = 628.

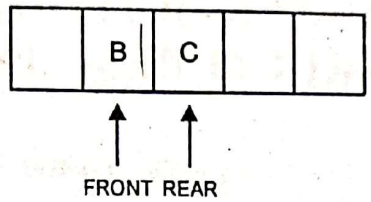
Q.4.(*b*) Consider a circular queue with fixed 5 position. Find the value of Front & REAR when following operations are performed

(*i*) A, B, C, inserted. **(*ii*) A is deleted.**
(*iii*) D & E are inserted. **(*iv*) B & C deleted.**
(*v*) F inserted. **(*vi*) D deleted.**
(*vii*) G & H inserted.

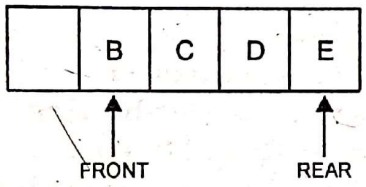
Ans. (*i*) A, B, C, inserted.



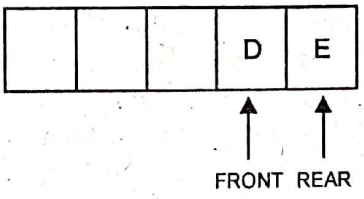
(*ii*) A is deleted



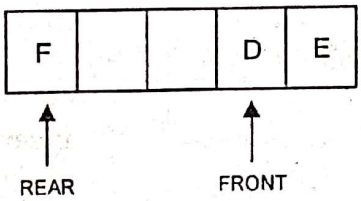
(*iii*) D & E are inserted.



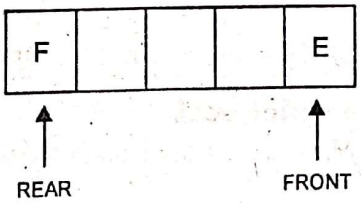
(*iv*) B & C deleted.



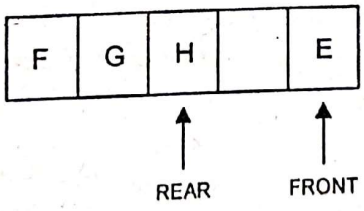
(*v*) F inserted



(*vi*) D is deleted.



(*vii*) G & H inserted.



END TERM EXAMINATION
THIRD SEMESTER (B.TECH) [ETCS-209]
DATA STRUCTURES—JAN. 2015

Time : 3.00 hrs. **M.M. : 75**

Note: *Attemptany five questions including Q.No. 1 which is compulsory.*

Q.1. Differentiate between

(*a*) B+ & B⁻ tree

(*b*) Stack and queue

(*c*) Merge Sort and Quick sort

(*d*) Sequential search and binary search

(*e*) Direct Hasing and Digit extraction hasting.

Ans. (*a*) Difference between B^+ and B^- tree.

Original B-tree had Record pointers in all of the index nodes; B^+ tree only in leaf nodes.

Given a key K and the two node pointers L and R around it

→ all key K and the two node pointer L and R around it

(1) all key values pointed to by L are < K.

(2) All key values pointed to by R are > = K.

→ B+ tree data pages are linked together to form a sequential file.

Q.1.(*b*) Stack and queue.

Stack	Queue
(1) Stack is a heap of elements placed one over the other.	(1) Queue is a set of elements one after the other.
(2) Stack Mechanism is last in First Out (LIFO).	(2) queue mechanism is First in First out (FIFO).
(3) Both insertion and deletion occur at the same end.	(3) Insertion and deletion occur at two different ends.
(4) If there is no value at the top, stack is empty.	(4) if front = rear queue is said to empty.
(5) This has no subtype.	(5) It has subtype such as queue, circular queue, double ended queue, priority queue.

Q.1.(*c*) Merge Sort and Quick sort.

Ans. (1) In worst case, Merge sort still maintain a complexity of o (n log n) while quick sort becomes O (n^2).

(2) In quick sort there is pivot selection but there is no pivot selection in merge sort. The array are partitioned equally which continues recursively and finally its merged.

(3) Merge sort is not an in place sorting while quick sort is (without regard to the internal stack space that taken by each recursive algorith).

Q.1.(*d*) Difference between sequential search & binary search.

Binary search	Sequential search
(1) Search all.	(1) Search one after other.
(2) Data should be in sorted.	(2) Data can be in any order.
(3) Only 1 when condition is used.	(3) Any no. of when condition.
(4) Only '=' relational operator used.	(4) Any relational operator
(5) Access is faster.	(5) Access is slow.

Q.1.(e) Direct Hashing and Digit extraction Hashing.

Sol. Direct Hashing: In direct Hashing, the key is the data file address without any algorithm manipulation. The file must therefore contain a record for every possible key. Although situation suitable for direct Hashing are limited. It can be very powerful because it generates that there are no synonyms or collision.

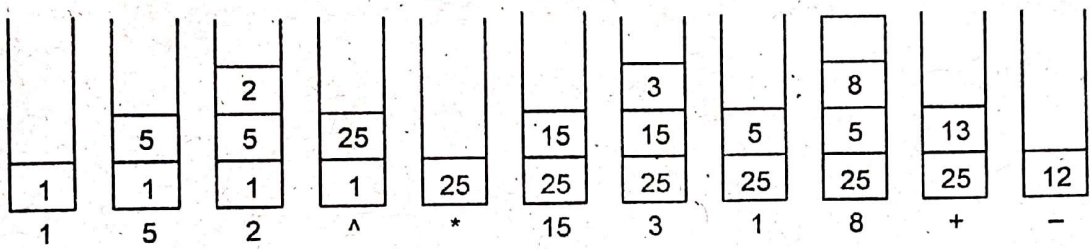
Digit Extraction Hashing: Using the digit extraction hasting, selected digit are extracted from the key and used as the address. for e.g. 6 digit Number to hash to a three-digit address. We could select the first, third and fourth digit and use them as the address.

$$\boxed{125870 \rightarrow 158}$$

Q.2. Use a Stack to evaluate the following arithmetic expression. Show the changing stack of the stack of the stack in tabular form.

XYZ ^* AB/C + – for X = 1, Y = 5, Z = 2, P = 3, A = 15, B = 3, C = 8

Ans. Substitute the value : 152 ^ * 153/8 + –



Q.3.(a) Discuss any two hash function using suitable example?

Ans. Hash Function: Hash function is a function which, when applied to the key, produced an integer which can be used as an address in a hash table.

Hash Function:

(1) Division Method: In the division method for creating hash function, we map a key k into one of m slots by taking the remainder of k divided by m.

$$h(k) = K \bmod m \text{ or } h(k) = K \bmod m + 1$$

for e.g. if the hash table has size $m = 12$ and the key is $k = 100$ then $h(k) = 4$ since it require only a single division operation, hashing by divisionis quite fast.

(2) Mid Square Method: In mid square method, we square the key, after getting the number, we take some digit from the middle of that number as an address. Let us take some 4 digit number as a key.

$$1337 \xrightarrow{\text{on squaring}} 1787569$$

now take 3rd and 4th digit from each number and that will be the hash address of these keys.

Q.3.(b) Which sorting algorithm is easily adaptable to singly linked lists? Explain your answer.

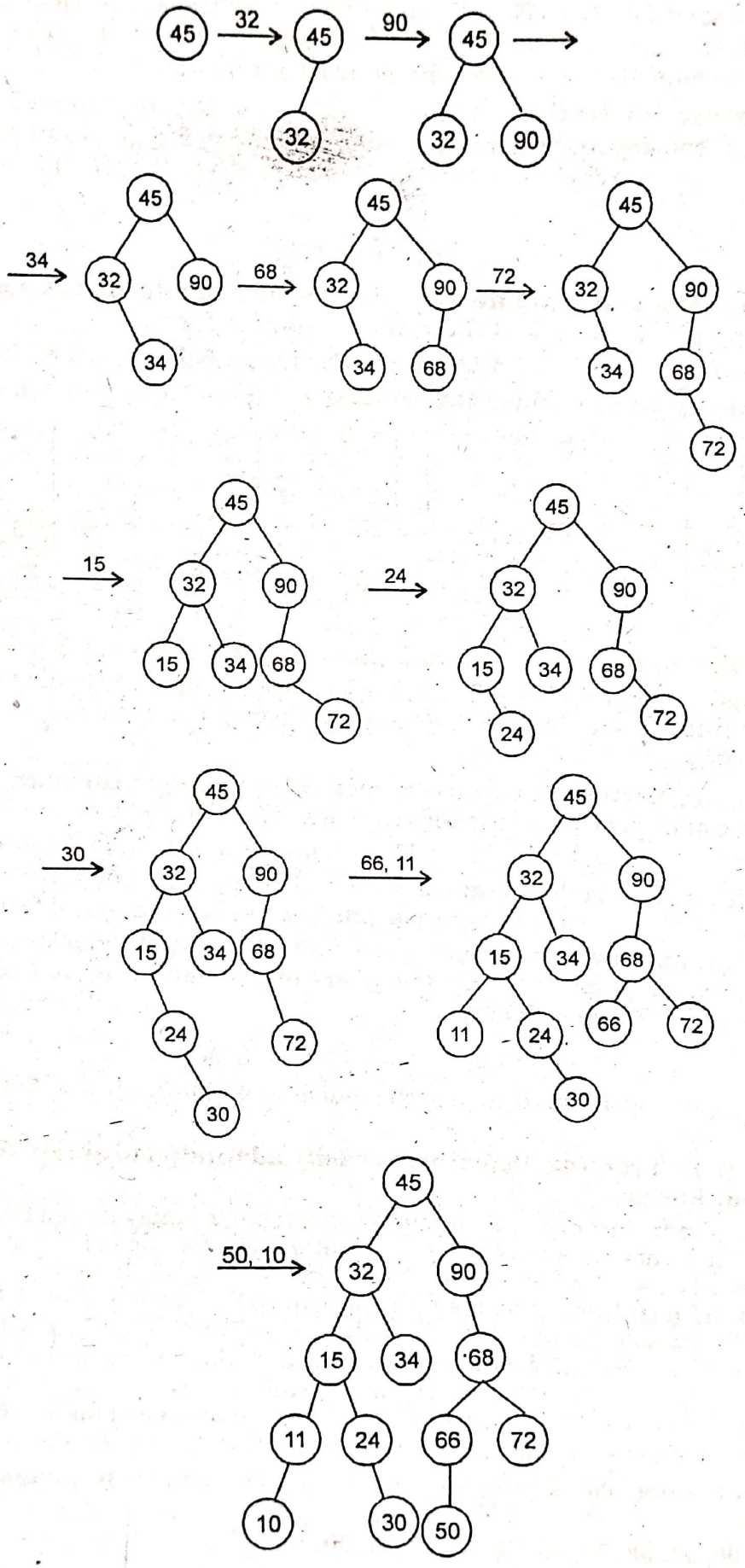
Sol. The simple insertion sort is simply adaptable to singly linked list. This is the method in which there is an array link of pointers, one for each of the original array. Initially link $[i] = i + 1$ for $0 <= i < n - 1$ and link $[n - 1] = -1$. Hence the array can be thought of as a linear link list pointed to by an external pointer first initialized to 0. To insert the kth element the linked list is traversed until the proper position for $x[k]$ is found or until the end of the list is reached. At that point $x[k]$ can be inserted into the list by simply adjusting the list pointers without shifting any element in the array. This decreases the time needed for insertion but not the time needed for searching for the proper position. The number of replacements in the link array is $O(n)$.

Q.4. Following list of numbers is inserted in order into an empty binary search tree.

45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10

Q.4. (*a*) Construction the binary search tree.

Ans.



Q.4. (*b*) Find the in order, pre-order and post-order traversal of BST created.

Ans. Inorder Traversal: (1) Traverse the left subtree of the root.
(2) Visit the root.
(3) Traverse the right subtree of root.
→ 10, 11, 15, 24, 30, 32, 34, 45, 50, 66, 68, 72, 90.

Preorder: (1) Visit the root.
(2) Traverse the left subtree of root.
(3) Traverse the right subtree of root.
→ 45, 32, 15, 11, 10, 24, 30, 34, 90, 68, 66, 50, 72

Postorder:
1. Traverse the Left Subtree of root.
2. Traverse the right subtree of root.
3. Visit the root.

10, 11, 30, 24, 15, 34, 32, 50, 66, 72, 68, 90, 45.

Q.5. For the following sequence determine the binary heap obtained when the keys are inserted one by one in order given into an initially empty heap 16, 14, 10, 8, 7, 9, 3, 2, 4, 1

Ans. Binary Heap: The Binary heap data structure is an array that can be viewed as a complete binary tree. Each node of the binary tree corresponds to an element of the carry. The array is completely filled on all levels except possibly lowest.

We represent heap in level order, going from left to right. The array corresponding to the heap.

	1	2	3	4	5	6	7	8	9	10
A	16	14	10	8	7	9	3	2	4	1

If an array a contain key value of nodes in a heap length [A] is the total number of elements.

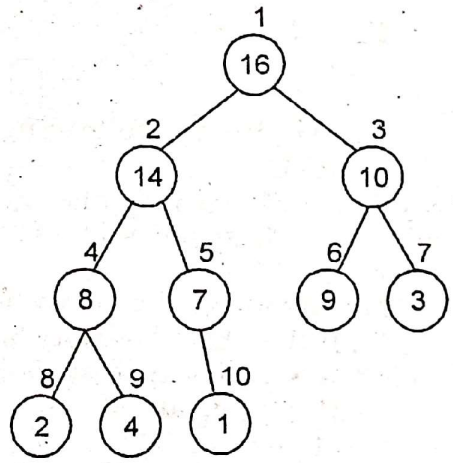
heap-size [A] = length [A] = number of elements

The root of the tree A[1] and given index i of a node the indices of its parent, left child and right child can be computed.

Parent (i)
    return floor (i/2)
Left (i)
    return 2i
Right (i)
    return 2i + 1

The index of 16 is 1. To find the index of left child we calculate 1 × 2 = 2.

This takes us correctly to 14.



Binary-Heap

Q.6. Sort the following numbers using exchange sort and Merge Sort. Illustrate the intermediate steps as well. 100, 300, 95, 60, 900, 800

Ans. Exchange Sort: Exchange sort attempt to improve the ordering by comparing the elements in pairs and exchanging them if they are not in sorted order. This operation is repeated till the table is sorted. The following method compare adjacent elements and is known as Bubble Sort.

First Pass

100 no swapping	100	100	100	100	100
300	95 (swapped)	95	95	95	95
95	300	60 (swapped)	60	60	60
60	60	300	10 (swap)	10	10
10	10	10	300	300 (No swap)	300
900	900	900	900	900	800 (swap)
800	800	800	800	800	900

2nd Pass

95 (swap.)	95	95	95	95	95
100	60 (Swap)	60	60	60	60
60	100	10 (swap)	10	10	10
10	10	100	100 (No swap)	100	100
300	300	300	300	300 (No swap)	300
800	800	800	800	800	800 (No swap)
900	900	900	900	900	900

3rd Pass:

60 (Swap)	60	10
95	10	60
10	95	95
100	100	100
300	300	300
800	800	800
900	900	900

4th Pass

No further swaps possible. This is sorted array.

Merge Sort: Merge sort is a sorting technique that divides the array into subarray of size 2 and merge adjacent pairs we than have approximately n/2 array of size 2. This process is repeated until there is only one array remaining of size n.

100, 300, 95, 60, 10, 900, 800

Pass 1:

100	300	60	95	10	900	800

Pass 2: Merge each pair of pairs to obtain the list of elements.

60	95	100	300	10	800	900

Pass 3: Merge the two subarray into one array.

10	60	95	100	300	800	900

Q.7. Define the following tree:

(*a*) AVL

Ans. AVL tree is a binary search tree that has an additional balance condition. This balance condition must be easy to maintain and ensure that the depth of the tree is *O* (log N).

Definition (1) An empty binary tree B is an AVL tree.

(2) If B is the non-empty binary tree with B_L and B_R are its left and right subtree then B is an AVL tree if and only if:

(*a*) B_L and B_R are AVL

(*b*) $|h_L - h_R| \leq 1$

Where h_L and h_R are the heights of B_L and B_R sub-tree.

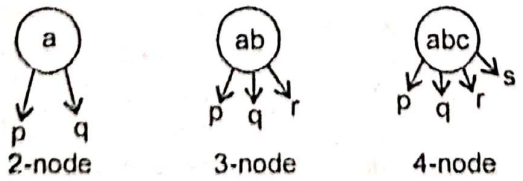
Balance Factor: To implement a AVL tree, each node must contain a balance factor.

BF = Height of left sub-tree – Height of Right sub-tree.

(*b*) 2-3-4 tree.

Ans. In Computer Science, a 2-3-4 tree (also called 2-4 tree) is a self balancing data structure that is commonly used to implement dictionaries. The numbers means a tree where every node with children (internal node) has either two, three or four child nodes.

- a 2-node has one data element, and its internal node has two child nodes.
- a 3-node has two data element, and its internal has three child nodes.
- a 4-node has 3 data element and its internal has 4-child nodes.

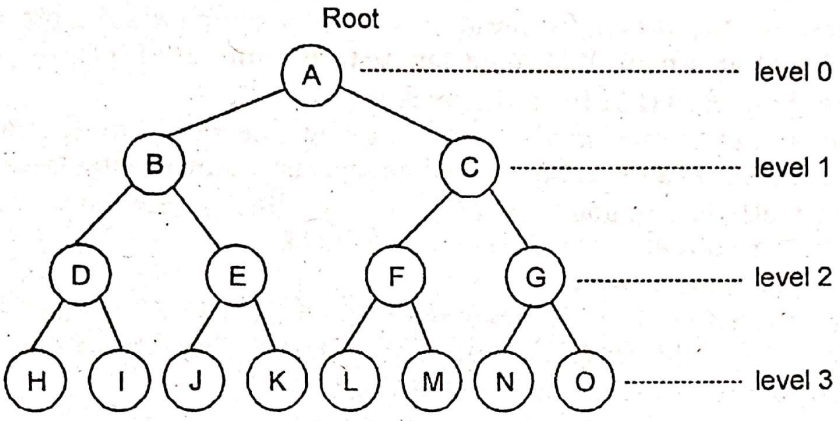


2-3-4 tree are B-tree of order 4 like B-tree in general they can search, insert or delete in O (log n) time.

- One property of 2-3-4 tree is all external enod are at the same depth.
- 2-3-4 tree are isometry of Red-black tree.

(c) Complete tree

Ans. A complete binary tree of depth d is the strictly binary tree all of whose leaves are at level *d*.



If a binary tree contain m nodes at level I, it contain at most 2m nodes at level $I + 1$, since a binary tree can contain almost one node at level O, it can contain almost $_2I$ node at level I.

Thus the total Number of node in a complete binary tree of depth d equals the sum of the number of nodes at each level between o and d i.e.

Number of nodes at level 0 is $2° = 1$

Number of nodes at level 1 is $2' = 1$

Number of nodes at level d is $2^d \Rightarrow 2^d$.

The total number of node is a complete binary tree is

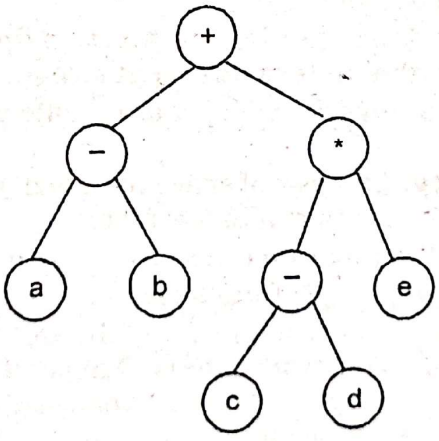
$$2^0 + 2^1 + 2^2 ... + 2^d = \sum_{j=0}^{d} d^j$$

(d) Expression tree.

Ans. Algebric expression such as

$$a/b + (c - d)\,e \qquad (1)$$

have an inherent tree structure for example following figure is a representation of the expression in equation (1). This kind of tree is called expression tree.



- Terminals node of an expression tree are the variable or constants.
- The non-Terminals of the expression tree are the operators.
- Parenthesis does not appear in the tree.
- The common algebric operators are either unary or binary.

Q.8. Write short notes on any two of the following:

(a) Bucket Hasing

Ans. Closed hashing stores all records directly in the hash table. Each record with the key value value K_R has a home position that is h (K_R). The slot computed by hash function.

When a record is inserted, the bucket to which it is mapped has available space to store the record. If the bucket does not have enough space, however it indicates an error condition called 'Bucket Overflow'. Bucket overflow occur done to following reasons.

Insufficient Buckets: The number of bucket which we denote by nb, must be choosen such that nb > nr/fr. Where nr denote total number of record.

fr denote the record that fit in the bucket.

Skewness: If the selection of a bucket, during insertion is more frequent than that of others, the bucket is said to be skewed as against being symmetrical.

- Bucket methods are good for implementing hash table stored on disk, because the bucket size can be set to the size of a disk block.

(b) Collision Resolution.

Ans. Suppose we want to add a new record R with the key K to our file F, but suppose the memory location address $H(k)$ is already occupied. This situation is called collision.

(1) Collision resolution by open addressing.

(2) Collision resolution by separate chaining.

Open Addressing: In this, all elements are stored in the hash table itself. That is each table entry contains either an element of the dynamic set or NIL.

Advantage: The advantage of open addressing is that it avoids pointers together. Instead of following pointers, we compute the sequence of slot to be examined. The extra memory freed by not storing pointers provides the hash table with a large number of slots for the same amount of memory.

Hashing with Changing: This method maintain a chain of elements which has the same hash address. We can take the hash table as an array of pointers. Size of hash table can be number of records. Here each pointer will point to one linked list and the elements which have the same hash address will be maintained in the linked list. We can maintained the linked list in sorted order and each elements of linked list will contain the whole record with key.

Advantage: The main advantage of this hashing is saving of memory space and perfect collision resolution.

- Here hash table contain only pointer which point to linked list-containing records.

Disadvantage: The main disadvantage is also wastage of memory space. In case where records are very small or contains only key.

(c) Complexity of program.

Ans. The complexity of an algorithm is a function $f(n)$ which measure the time and for space used by an algorithm in terms of input size n.

The choice of particular algorithm depends on following performance analysis and measurements.

(1) Space complexity: Analysis of space complexity of an algorithm or program is the amount of memory it needs to run to complete.

The space needed by the program consist of following components.

(1) Instruction space (2) Data space (3) Environment stack space.

(2) Time complexity: Time complexity of an algorithm is the amount of time it needs to run to completion. The exact time will depend on the implementation of the algorithm programming language, optimising the capability of the compiler used, the CPU speed, other hardware and so on.

- To measure the time complexity accurately, we have to count all sorts of operations performed in an algorithm.
- The time complexity also depend on the amount of data input to an algorithm.

HARDIK SETHI

FIRST TERM EXAMINATION [SEPT.-2015]
THIRD SEMESTER [B. TECH]
DATA STRUCTURE [ETCS-209]

Time: 1½ Hrs. **MM : 30**

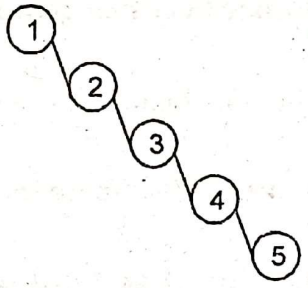
Note: *Attempt Q.No.1 which is compulsory any two more questions .*

Q.1. (*a*) How the resulting BST will look if the input is already sorted in non-decreasing order. **(1)**

Ans. If all nodes are already sorted in non decreasing order then all nodes are inserted into the right side of a binary search tree, for example →

nodes are 1, 2, 3, 4, 5

the BST will look like →



Q.1. (*b*) Construct Binary tree T from following sequence. **(2)**

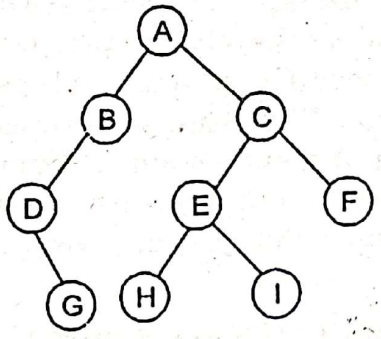
PREORDER: A B D G C E H I F

INORDER: D G B A H E I C F

Ans.

PREORDER:ABDGCEHIF

INORDER:DGBAHEICF



Q.1. (*c*) Number of nodes in complete tree is 1000000. Find its depth. **(1)**

Ans. The depth of complete binary tree is log(n) where no. of nodes is n. So the depth of complete binary tree which has the 1000000 nodes is.

$$\log(1000000) = 6$$

Q.1. (*d*) Write algorithm/program to perform POP operation on stack using singly linked list. **(2)**

Ans. Void pop (c)

```
{
    Struct node * ptr;
    If (top = = NULL)
{ printf ("Underflow")
 return;
}
ptr = top;
top = ptr → link;
free (ptr);
}
```

Q.1. (*e*) What is the difference between ground header linked list and circular header linked list? (1)

Ans. A grounded header list is a header list where the last node contains the null pointer.

A circular header list is a header list where the last node points back to the header node.

Q.1. (*f*) What is the condition that a circular queue is full if the queue is implemented using arrays? (1)

Ans. If ((front = = 0) && (rear == Max – 1) || (front == rear + 1))

that means queue is overflow or full.

Q.1. (*g*) Binary tree with N nodes has exactly ..... Null branches (1)

Ans. Binary tree with N nodes has exactly $n + 1$ null branches.

Q.1. (*h*) What is Time Space Trade off?

Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q.2. (*a*) Why height balancing tree is required? Create AVL tree of following. *a, z, b, y, c, x, d, w, e, v, f.* (8)

Ans. The first balanced binary search tree or height balancing tree was the AVL tree with has an additional balancing condition. The simplest idea is to require that the left and right subtrees have the same height, so height balancing tree is required.

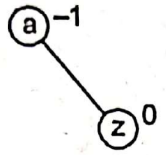
AVL tree of nodes– *a, z, b, y, c, x, d, w, e, v, f.*

Creation of AVL Tree

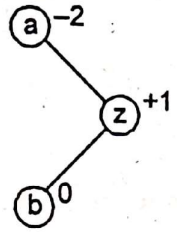
Nodes: *a, z, b, y, c, x, d, w, e, v, f*

Insert *a*

Insert *z*

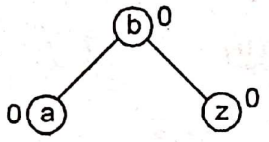


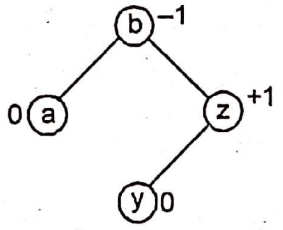
Insert *z*



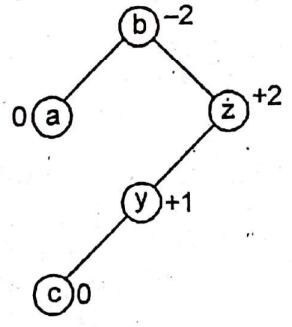
There is Balance factor of a is –2 so after *R-L* rotation the tree is.

Insert *y*

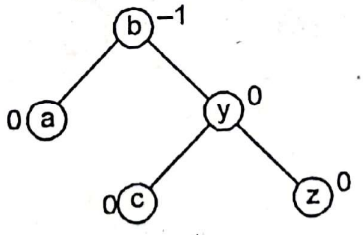




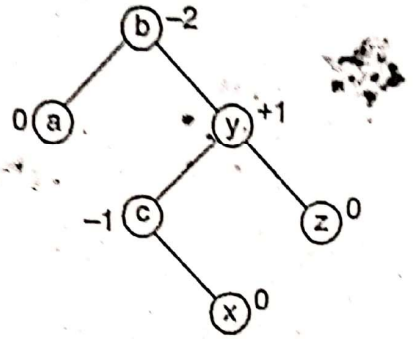
Insert *c*



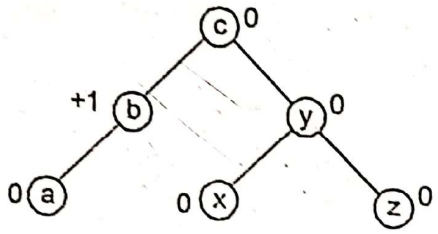
There is Balance factor of *z* is + 2 so after *L – L* rotation the tree is –



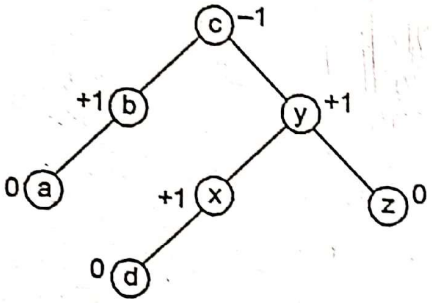
Insert x



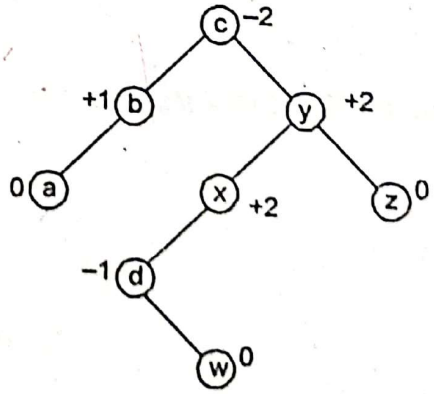
There is the Balance factor of b is – 2 So after L–R rotation the tree is–



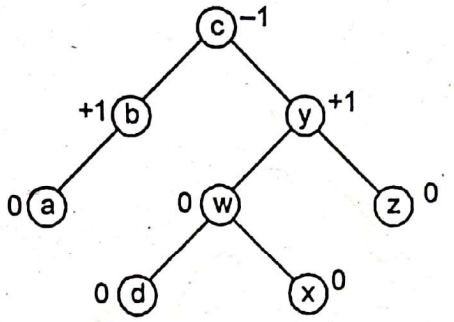
Insert d



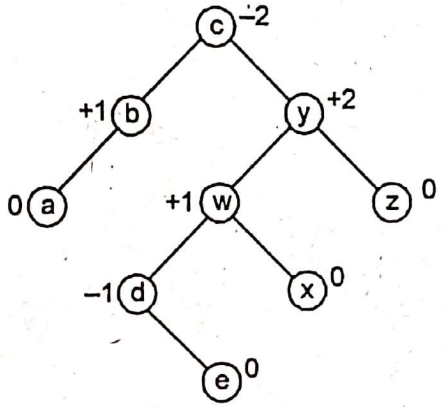
Insert w



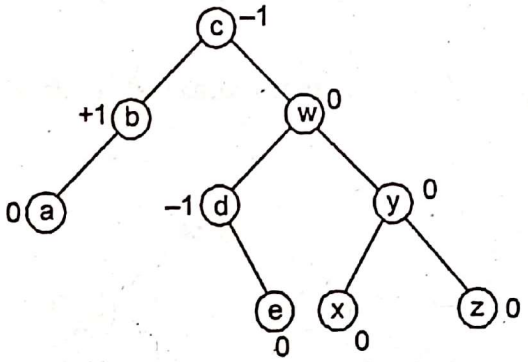
After Insertion of w the Balance factor is unbalanced so after applying the L-R rotation the tree is–



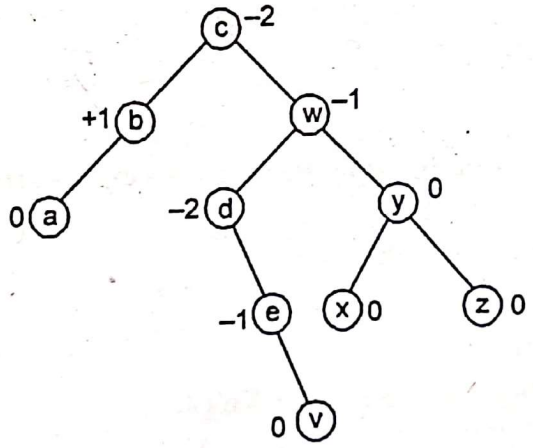
Insert e



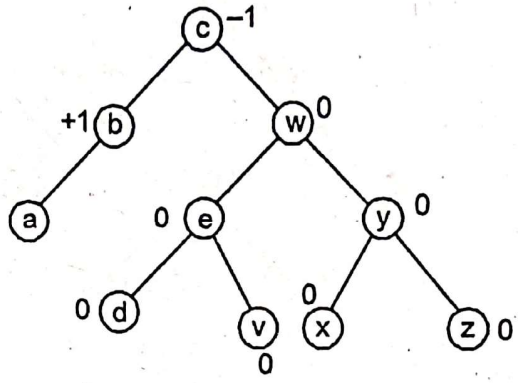
After Insertion of node e the tree is unbalanced so after applying the L-L rotation the tree is–



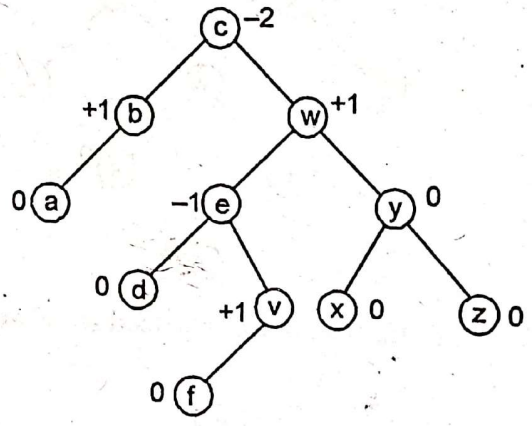
Insert v



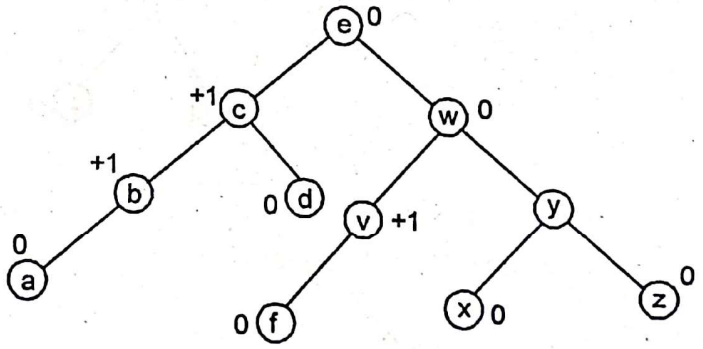
After Insert the node *v* the tree is unbalanced so after apply the R.R rotation the tree is–



Insert *f*



After Insert the node *f* the tree is unbalanced so after apply the *R–L* rotation the final tree is–



Q. 2. (*b*) Write a procedure/program to reverse a singly linked without using any more memory? **(2)**

Ans. Reverse a singly linked list.

Step 1 PTR = FIRST

Step 2 TPT = NULL

Step 3 Repeat step 4 while PTR! = NULL

Step 4 (a) CPT = LINk (PTR)

(*b*) LINK (PTR) = TPT

(*c*) TPT = PTR

(*d*) PTR = CPT

Step 5 Stop.

Q.3. (*a*) Consider the following in fix expression and convert into reverse polish notation using stack. **(5)**

Ans. (A + (B*C–(D/E^F)*G)*H)

Character Scanned	Stack	Reverse polish Notatic Postfix Notation
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
(	(+(*	ABC
–	(+(–	ABC*
(	(+(–(	ABC*
D	(+(–(	ABC* D
/	(+(–(/	ABC * D
E	(+(–(/	ABC * DE
^	(+(–(/^	ABC * DE
F	(+(–(/^	ABC * DEF
)	(+(–	ABC * DEF ^
*	(+(–*	ABC * DEF ^ /
G	(+(–*	ABC * DEF ^ / G
)	(+	ABC * DEF ^ / G * –
*	(+*	ABC * DEF ^ / G * –
H	(+*	ABC * DEF ^ / G * –
)	—	ABC * DEF ^ / G * – H * + Reverse polish notation

Q.3. (*b*) What is an algorithm/program for inserting and deleting a node at a given location in circular linked list. **(5)**

Ans. Inserting a node at given location.

Step 1. If AVAIL = NULL then write overflow and stop

Step 2. Read DATA as information of node after which insertion will be made.

Step 3. PTR = AVAIL

AVAIL = LINK (AVAIL)

Read Info (PTR)

Step 4. CPT = FIRST

Step 5. Repeat Step 6 while Info (CPT) ! = DATA

Step 6. CPT = LINK (CPT)

Step 7. LINK (PTR) = LINK (CPT)

LINK (CPT) = PTR

Step 8. Stop.

Deletion of node at given Location.

Step 1. If FIRST = NULL then write underflow and stop.

Step 2. Read DATA as information of node to be deleted.

Step 3. PTR = FIRST

Step 4. Repeat step 5 while info (PTR) ≠ DATA

Step 5. CPT = PTR

PTR = LINK (PTR)

Step 6. LINK (CPT) = LINK (PTR)

Step 7. LINK (PTR) = AVAIL

AVAIL = PTR

Step 8. Stop.

Q.4. (*a*) Suppose multidimensional arrays A and B are declared as A(-2:2, 2:22) and B (1:8, -5:5, –10:5) stored in column major order.

(*i*) Find the length of each dimensions of A and B. **(2.5)**

(*ii*) The number of elements in A and B. **(1)**

(*iii*) Consider the elements B(3, 3, 3) in B. Find the effective indices E1, E2, E3, and the address of the element assuming Base(*b*) = 400 and there are w = 4 words per memory location. **(2.5)**

Ans. (*i*) The length of a dimension is obtained by:

Length = upper bound – Lower bound + 1

Hence the length Li of the dimensions of A are:

L1 = 2 – (–2) + 1 = 5 and L2 = 22 – 2 + 1 = 21

L1 = 8 – 1 + 1 = 8, L2 = 5 – (–5) + 1 + 11, L35 – (–10) + 1 = 16

(*ii*) Accordingly A has 5*21 = 105 elements and B has 8*11*16 = 1408 elements.

(*iii*) The effective index Ei is obtained from Ei = ki–LB, where ki is the given index and LB is the lower bound.

Hence E1 = 3 – 1 = 2, E2 = 3 – (–5) = E3 = 3 – (–10) = 13

Considering *B* is stored in column-major order, the address of *B* [3,3,3] will be calculated as.

E3L2 = 13*11 = 143, E2L2 + E2 = 143 + 8 = 151

(E3L2 + E2) L1 = 151*8 = 1208, (E3L2 + E2) L1 + E1 = 1208 + 2 = 1210

Therfore address of B [3,3,3] = 400 + 4(1210) = 400 + 4840 = 5240.

Q.4 (*b*) What is Garbage Collections?

Ans. Gabage Collection: Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory block and reallocates storage for reuse. The primary purpose of garbage collection is to reduce memory leaks.

Q.4. (c) Write algorithm/program for insertion operation on queue using Singly linked list.

Ans.

```
void insert ()
{
struct node * ptr;
ptr = (struct node *) malloc (size of (struct node));
int item;
prinf ("Input the element for Inserting");
scanf ("/d", and item);
ptr → info = item;
ptr → Link = NULL;
If (front = = NULL)
   front = Ptr;
else
rear → link = ptr;
rear = ptr;
}
```

SECOND TERM EXAMINATION [NOV.-2015]
THIRD SEMESTER [B. TECH]
DATA STRUCTURE [ETCS-209]

Time: 1½ Hrs. **MM : 30**

Note: *Attempt Q.No.1 which is compulsory any two more questions .*

Q.1. (*a*) Difference between B– tree and B+ tree? **(2)**

Ans. Original B-tree had record pointers in all of the index nodes; B+ tree only in leaf nodes.

Given a key K and the two node pointers L and R around it.

→ All key K and the two node pointer L and R around it

1. All key values pointed to by L are < K.
2. All key values pointed to by R and > = K.

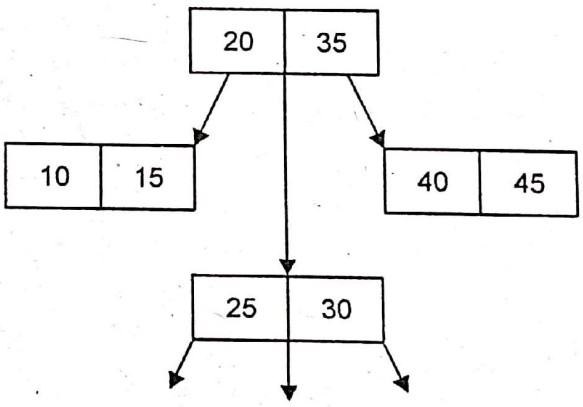
→ B+ tree data pages are linked together to form a sequential file.

Q.1. (*b*) Construct 3–way search tree for the following list of keys.

List: 20, 35, 40, 10, 15, 25, 30, 45

Ans. List: 20, 35, 40, 10, 15, 25, 30, 45

3 way Search tree.



Q.1. (c) Consider the following specification of a graph.

V(G) = {1, 2, 3, 4, 5} and E(G) = {(1, 2), (1, 3), (3, 3), (3, 4), (4, 1), (4, 5), (5, 2
Draw its adjacency matrix. **(**

Ans. V(G) = {1, 2, 3, 4, 5}

E(G) = {(1, 2), (1, 3), (3, 3), (3, 4), (4, 1), (4, 5), (5, 2)}

Adjacency Matrix Representation.

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	0	0
3	0	0	1	1	0
4	1	0	0	0	1
5	0	1	0	0	0

Q.1. (*d*) Compare the complexities in average case and worst case of th following sorting:

Merge sort, Heap sort, Quick sort, Selection sort

Ans.

Name	Best	Average	Worst
Quick Sort	N logN	N logN	N^2
Merge Sort	N logN	N logN	N logN
Heap Sort	N logN	N logN	N logN
Selection Sort	N^2	N^2	N^2

Q.1. (*e*) What is pseudorandom hashing method?

Ans. Pseudorandom Hashing

A common random-number generator is shown below.

$$y = ax + c$$

To use the pseudorandom-number generator as a hasing method, we set x to the key, multiply it by the coefficient a, and then add the constant c. The result is then divided by the list size, with the remainder being the hashed address.

Example:

$$Y = ((17 \times 122267) \text{ modulo } 307$$
$$Y = (2061539 + 7) \text{ modulo } 307$$
$$Y = 2061546$$
$$Y = 41$$

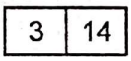
Q.2. (*a*) Construct B-TREE of order 5 by inserting the following elements. 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25 and 19. And also delete values 6, 23, 3 from above constructed tree.

Ans. B-Tree of order 5:

Elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19

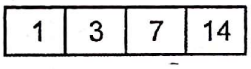
Insert 3

Insert 14

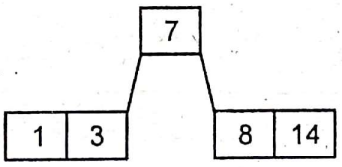


Insert 7

Insert 1

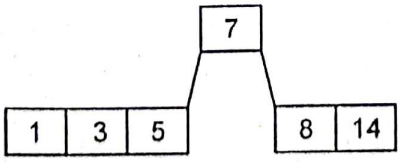


Insert 8

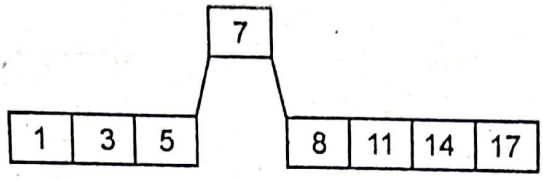


Here node was already full 50 after insert 8, it is splitted into 2 node and 7 is the median key.

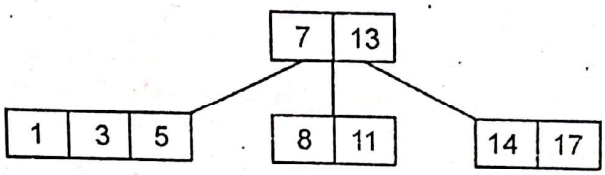
Insert 5



Insert 11 and 17

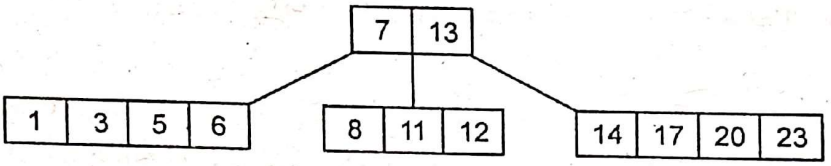


Insert 13

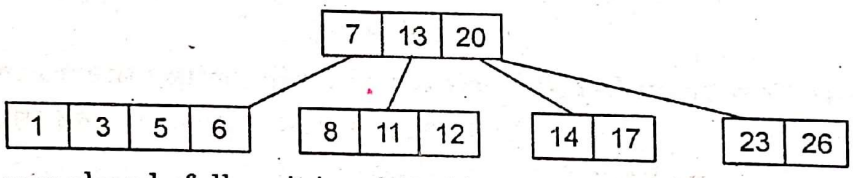


Here node was already full so after insert 13, It is splitted into 2 nodes and 13 is the median key.

Insert 6, 23, 12, 20

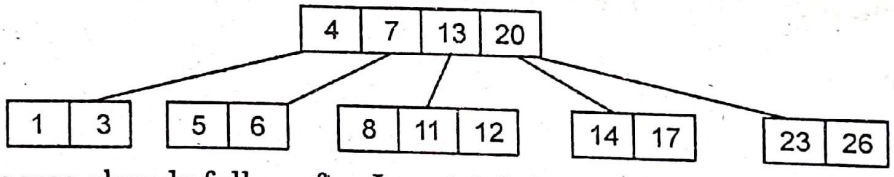


Insert 26



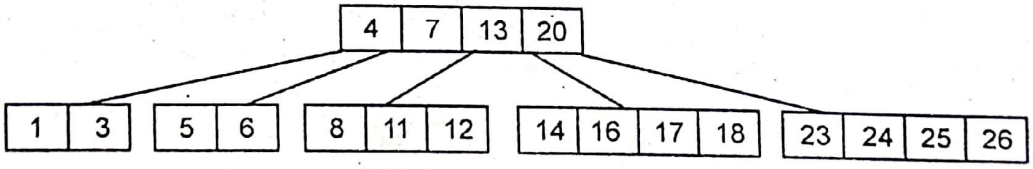
Here node was already full, so it is splitted into 2 nodes and 20 is the median key.

Insert 4

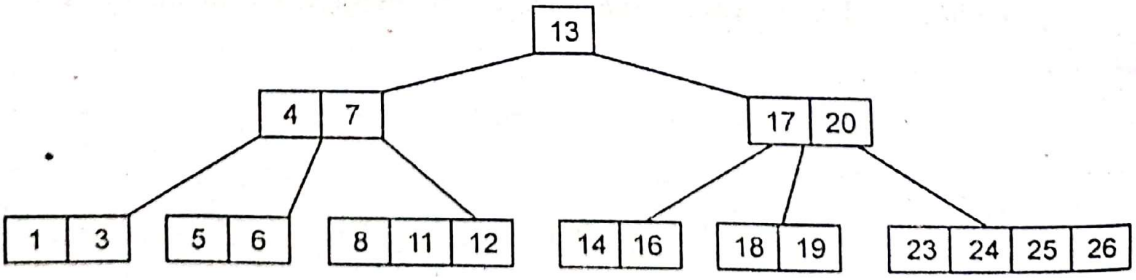


Here node was already full so after Insert 4, It is splitted into 2 nodes and 4 is the median key.

Insert 16, 18, 24, 25

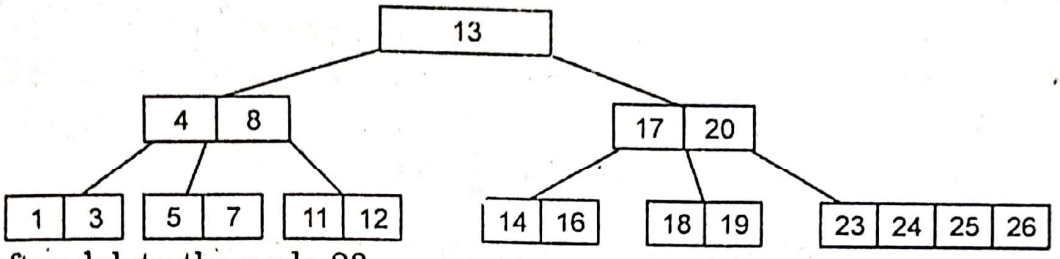


Insert 19

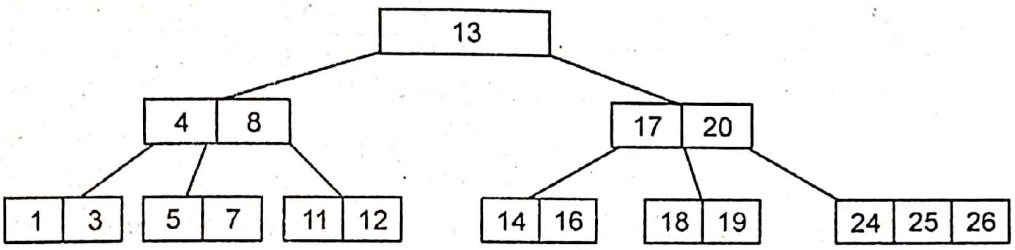


Here node was already full so after Insert 19 it is splitted, into 2 nodes. 17 is the median key so it will go into parent node. Here perent node is already full, so it is splitted in 2 nodes and 13 is the median key and it will become the new root.

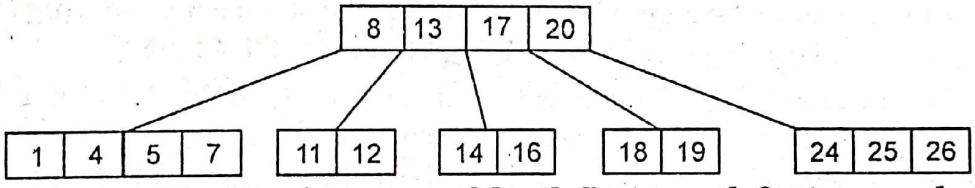
After delete the node 6



After delete the node 23



After delete the node 3



Q.2. (*b*) Write algorithm/program of depth first search for traversal of graph?

Ans. Step 1: Set status = 1 (ready state) for each node in G.

Step 2 : Push the starting node A on the stack and set its status = 2 (writing state).

Step 3 : Repeat steps 4 and 5 until stack is empty.

Step 4 : Pop the top Node N. Process it and set its status = 3 (processed state).

Step 5 : Push on to the stack all the neighbour of N that are in the ready state (whose status = 1) and set their status = 2 (waiting state).

Step 6 : Exit.

Q.3. (*a*) Suppose the table T has 11 memeory locations T {1}, T{2},.....T[11] and suppose the File F consist of 8 records *A, B, C, D, E, X, Y, Z* Record: *A, B, C, D, E, X, Y, Z* and its hash address U(K): 4, 8, 2, 11, 4, 11, 5, 1 resp. How file F will appear in memory when 8 records are entered into table T in above order using linear probing and quadratic probing collision resolution technique? Find average successful search probes (*S*) and average unsuccessful search probes (S) and average unsuccessful search probes (*U*) also for both? (6)

Ans.

Record	A,	B,	C,	D,	E,	X,	Y,	Z
H (*k*)	4,	8,	2,	11,	4,	11,	5,	1

Suppose the 8 records are entered into the table T in above order using the liner probing and Quadratic probing collision resolution technique then the file F will appear in memory as follows.

Table T:	X,	C,	Z,	A,	E,	Y,	–,	B,	–,	–,	D
Address:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11

Although Y is the only record with hash address $H(k) = 5$, the record is not assigned to T [5], since T [5] has already been filled by E because of a previous collision at T(4). Similarly Z does not appear in T (1).

The average no. S of probes for a successful search follows:

$$S = \frac{1+1+1+1+2+2+2+3}{8}$$

$$= \frac{13}{8} = 1.6$$

The average no. U of probes for an unsuccessful search follows.

$$U = \frac{7+6+5+4+3+2+1+2+1+1+8}{11}$$

$$= \frac{40}{11} = 3.6$$

The first sum adds the no. of probes to find each of he 8 records, and the second sum adds the no. of probes to find as empty location for each of the 11 locations.

Q.3. (*b*) Difference between internal sorting and external sorting? Sort the following elements using shell sort: 11, 88, 22, 77, 33, 66, 44, 55. **(1+3)**

Ans. Internal sorting: Which deals with sorting the data stored in computer's main memory. Example: Bubble, selection etc. sorting.

External sorting: Which deals with sorting the data stored in files. External sorting is applied when data can not be stored in the memory. Example: Files and its organization.

Shell sort: 11, 88, 22, 77, 33, 66, 44, 55

Input to pass 1 with distance = 2

11	88	22	77	33	66	44	55

Output of pass 1 is input to pass 2 and distance = 1

11	55	22	66	33	77	44	88

Output of pass 2 is

11	22	33	44	55	66	77	88

Q.4. (*a*) Consider a company with 68 employee. Each has been assigned a 4 digit employee number which has been assigned a 4 digit employee number which is used as the primary key in the company's employee file. Suppose L (set of memory addresses of the location) consist of 100 two digit addresses 00,01,02---99. Then apply the division method using prime no. closest to 99, mid square method and folding method to find out 2-digit hash address for each of the following employee no. 9614, 5882, 1825. **(2+2+2)**

Ans. (*i*) Division Method: Choose a prime no. M close to 99, such as M = 97 Then.

H (9614) = 11, H (5882) = 62 H (1825) = 79

That is dividing 9614 by 97 gives a remainder 11. dividing 5882 by 97 gives a remainder 62 and so on. In the case that the memory address begin with 01 rather than 00, we choose that the function H(k) = k (mod M) + 1 to obtain:

H (9614) = 11 + 1 = 12, H (5882) = 62 + 1 = 63 H (1825) = 79 + 1 = 80

(*ii*) Midsquare Method:

K	9614	5882	1825
k^2	92428996	34597924	3330625
H(k)	28	97	06

Observe 4th and 5th digit are choosen for Hash address.

(*iii*) Folding Method: Choping the keys K into 1, 2, 1 digits and add them. So the address will be:

H (9614) = 9 + 61 + 4 = 74

H (5882) = 5 + 88 + 2 = 95

H (1825) = 1 + 82 + 5 = 88

Q.4. (*b*) Write algorithm/program of anyone sorting technique which is used when input elements are small? Difference between comparison based sorting and non-comparison based sorting? (3+1)

Ans. When Input elements are small then many sorting techniques may be used such as, bubble, sort, insertion sort, selection sort etc.

Algorithm of Selection Sort

1. n = length [A]
2. For j = 1 to $n - 1$
3. smallest = j
4. for $i = j + 1$ to n
5. If $A[i] < A$ smallest]
6. then smallest = i
7. exchange (A [j], A [smallest])

Comparison based and non comparison based sorting

Comparison based sorting is a type of soring that only reads the list of elements through a single abstract comparison operation that determines which of 2 elements should occur first in the final sorted list. While in non comparison based sorting algorithm there are no need of comparison between elements. Example of comparison based sorting are bubble, insertion, selection etc and radix sort is the non comparison based sorting algorithm.

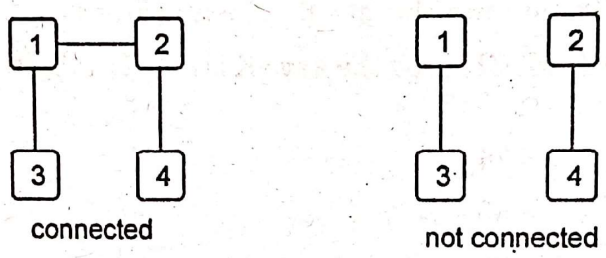
END TERM EXAMINATION [DEC.-2015]
THIRD SEMESTER [B. TECH]
DATA STRUCTURE [ETCS-209]

Time: 3 Hrs. **MM:7**

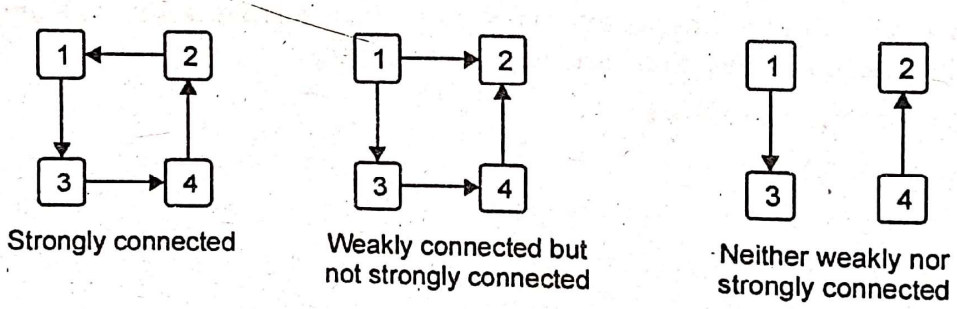
Note: *Attempt any five questions including Q.No.1 which is compulsory.*

Q.1. (*a*) What does it mean to say that a "graph G is connected" and wh does it mean to say that "graph is strongly connected"? (

Ans. An ***undirected*** graph is **connected** if there is a path from every node to eve other node. For example:



A ***directed*** graph is **strongly connected** if there is a path from every node to eve other node. A ***directed*** graph is **weakly connected** if, treating all edges as bei ***undirected,*** there is a path from every node to every other node. For example:



Q.1 (*b*) Define B tree with example. (

Ans. B-Tree Properties

A B-tree is a self-balancing tree data structure that keeps data sorted and allov searches, sequential access, insertions, and deletions in logarithmic time. The B-tree a generalization of a binary search tree in that a node can have more than two childre Unlike self-balancing binary search trees, the B-tree is optimized for systems that rea and write large blocks of data B-trees are a good example of a data structure for extern memory. It is commonly used in databases and filesystems.

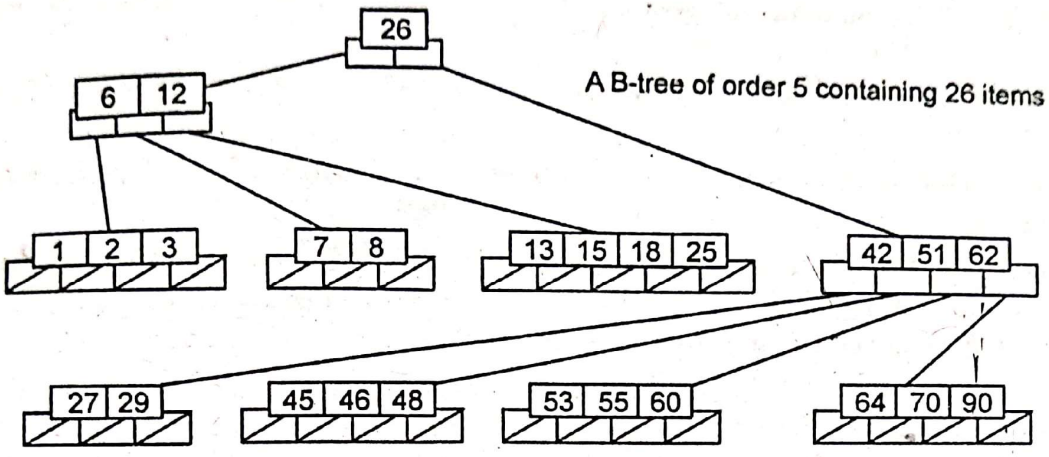
Properties of B-Tree:

1. Data is stored at the leaves - All leaves are at the same depth and contai between L/2 and L data items

2. Internal nodes store up to M-1 keys

3. Internal nodes have between M/2 and M children

4. Root (special case) has between 2 and M children (or root could be a leaf)



A B-tree of order 5 containing 26 items

Q.1. (*c*) What is complexity of selection sort and insertion sort for average case? **(5)**

Ans. Time Complexity:

Algorithm	Data structure	Time complexity		
		Best	Average	Worst
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)

Q.1. (*d*) What are the various graph traversal techniques? Explain. **(5)**

Ans. Graph Traversal Techniques:

1. Breadth-First Search: The bread-first-search algorithm starts at a vertex i, and visits, first the neighbours of i , then the neighbours of the neighbours of i , then the neighbours of the neighbours of the neighbours of i, and so on.

This algorithm is a generalization of the breadth-first traversal algorithm for binary trees, and is very similar; it uses a queue (*q*) , that initially contains only i. It then repeatedly extracts an element from q and adds its neighbours to q, provided that these neighbours have never been in q, before.

Algorithmic Steps

Step 1: Push the root node in the Queue.

Step 2: Loop until the queue is empty.

Step 3: Remove the node from the Queue.

Step 4: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

2. Depth-First Search: The depth-first-search algorithm is similar to the standard algorithm for traversing binary trees, it first fully explores one subtree before returning to the current node and then exploring the other subtree. Another way to think of depth-first-search is; by saying that it is similar to breadth-first search except that it uses a stack instead of a queue.

Algorithmic Steps:

Step 1: Push the root node in the Stack.

Step 2: Loop until stack is empty.

Step 3: Peek the node of the stack.

Step 4: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.

Step 5: If the node does not have any unvisited child nodes, pop the node from the stack.

Q.1. (e) Define infix expression and postfix expression with example. Mention its advantage. **(5)**

Ans. Infix, Postfix and Prefix: Infix, Postfix and Prefix notations are three different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands. Infix notation: X + Y

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A* (B + C)/D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

Postfix notation (also known as "Reverse Polish notation"): X Y +

Operators are written after their operands. The infix expression given above is equivalent to ABC + *D/

The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "*" in the example above, the addition must be performed before the multiplication. Operators act on values immediately to the left of them. For example, the "+" above uses the "B" and "C". We can add brackets to make this explicit:

((A (B C +) *) D/)

Thus, the "*" uses the two values immediately preceding: "A", and the result of the addition. Similarly, the "/" uses the result of the multiplication and the "D".

Infix notation:

Advantages:

1. Commonality of expression.

2. We don't need separation symbols for numbers, since symbols for functions effectively do this.

3. We don't have to have a clear idea about the arity of functional symbols before using them.

Postfix notation:

Advantages:

1. Can mechanically use the rule of replacement.

2. So far as I can tell, it comes as consonant with how we practically use mathematics in everyday life, *e.g.* when we buy something with say x dollars which costs y dollars, we *then* figure out how much x and y differ by. Or if we start off walking at time j and finish walking at time i without a stopwatch, *then* we figure out how long we've walked for by figuring out how much j and i differ by.

3. We can quickly learn about the infinity of arithmetical expressions.

Q.2. (a) Write an algorithm/program which convert a postfix express to a prefix expression. Give one example. **(6)**

Ans. Algorithm for converting postfix to prefix expression.

Step 1 : Accept a postfix string from user.

Step 2 : Start scanning the string from left to right one char. at a time.

Step 3 : If it is an operand push it in stack.

Step 4 : If it is an operator, pop opnd 2 and opnd 1 and con caterate then in the order (optr, opnd 1, opnd 2) and push the result into stack.

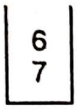
Step 5: Repeat these steps unitl array of input postfix string ends.

Step 6: Pop the remaining element of the stack which is the required prefix notation.

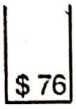
Example: 76 $ 5 * 5 –

Step 1 : Scan char is digit 7 so push into stack.

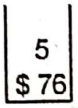
Step 2 : Scan char is digit 6 so push into stack.



Step 3 : Scan char is $ which is operator so pop 2 elements prom stack and concatenate them and result push into stack.

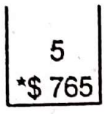


Step 4 : Scan char is digit 5 so push into stack.

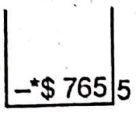


Step 5 : Scan char is an operator * so pop 2 elements and after concatenation add into stack.

Step 6 : Scan char is digit 5 so push into stack.



Step 7 : Scan char is an operator – so again pop elements and after concatenations add into stack.



Resultant prefix expression is –*$7655.

Q.2. (*b*) Write an implementation of the Queue Abstract Data Type using two stacks. Write algorithm for Enqueue and Dequeue operations. (6.5)

Ans. Implementation of Queue using 2 Stacks

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

```
struct node
{
    int data;
    struct node *next;
};
void push(struct node** top, int data);
int pop(struct node** top);
struct queue
{
    struct node *stack l;
    struct node *stack2;
};
void enqueue(struct queue *q., int x)
{
    push(&q->stack1, x);
}
void dequeue(struct queue *q)
{
    int x;
    if (q->stack1 == NULL && q->stack2 == NULL) {
        printf("queue is empty");
        return;
    }
    If (q->stack2 == NULL) {
        while (q->stack1 != NULL) {
        x = pop(&q->stack1);
        push(&q->stack2, x);
        }
    }
    x = pop(&q->stack2);
    printf("%d\n",, x);
}
void push(struct node** top, int data)
{
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Stack overflow \n");
```

```
        return;
    }
newnode->data = data;
newnode->next = (*top);
(*top) = newnode;
}
int pop(struct node** top)
{
int buff;
struct node *t;
if (*top == NULL) {
    printf("Stack underflow \n");
    return:
}
else {
    t = *top;
    buff = t->data;
    *top = t->next;
    free(t);
    return buff;
}
}
void display(struct node *top 1 struct node *top2)
{
while (top1 != NULL) {
    printf("%d\n", top1->data);
    top1 = top1->next;
}
while (top2 != NULL) {
    printf("%d\n", top2->data);
    top2 = top2->next;
}
}
int main ( )
{
struct queue *q = (struct queue*)malloc(sizeof (struct queue));
int f = 0,  a;
char ch - 'y';
```

```
q->stackl = NULL;
q->stack2 = NULL;
while (ch == 'y' | |ch == 'Y') {
    printf ("enter ur choice\n1.add to queue\n2. remove
            from queue\n3.display\n4.exit\n");
    scanf("%d"., &f);
    switch(f) {
        case 1 : printf ("enter the element to be added to queue\n");
                 scanf("%d", &a);
                 enqueue(q, a);
                 break;
        case 2 : dequeue(q);
                 break;
        case 3 : display(q->stack1, q->stack2);
                 break;
        case 4 : exit(1);
                 break;
        default : printf ("invalid\n");
                 break;
        }
    }
}
```

Algorithm for enqueue operation

```
if queue is full
return overflow
endif
rear ← rear + 1
queue [rear] ← data
return trun
end procedure
```

Algorithm for dequeue operation

```
if queue is empty
return underflow
endif
data = queue[front]
front ← front + 1
return trun
end procedure
```

Q.3. Draw Tree with following information: **(12.5)**

Preorder: **G, B, Q, A, C, K, F, P, D, E, R, H**

Inorder: **Q, B, K, C, F, A, G, P, E, D, H, R**

Find Postorder of the above Tree.

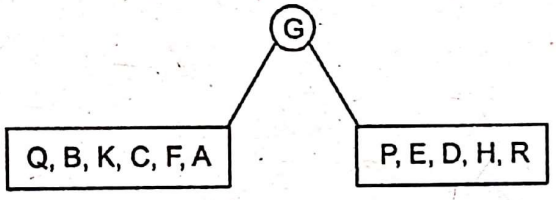
Ans.

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H

Inorder: Q, B, K, C, F, A, G, P, E, D, H, R

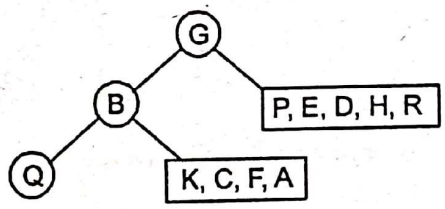
Root node is G

Inorder. $\underbrace{\text{Q, B, K, C, F, A}}_{\text{Left subtree}}$ G $\underbrace{\text{P, E, D, H, R}}_{\text{Right subtree}}$



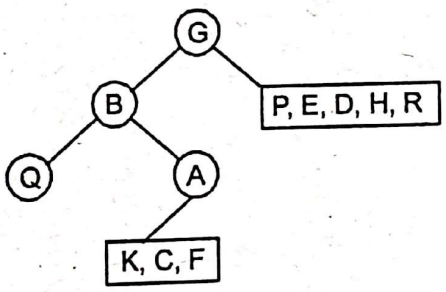
Now in preorden next node is B.

So



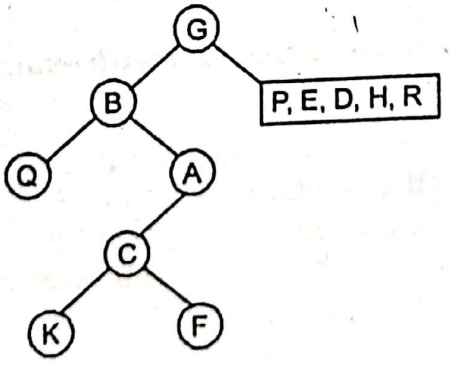
In inorder *Q* is left node and right subtree is K, C, F, A.

Now next node in preorder is A and in inorder left subtree of A is K, C, F, so the next parent is A.

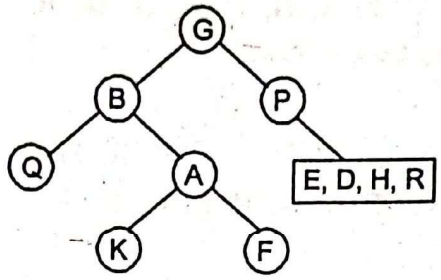


Now in preorder the next node is C so the parent node is C and In inorder the left node of C is K and right node of C is F.

So

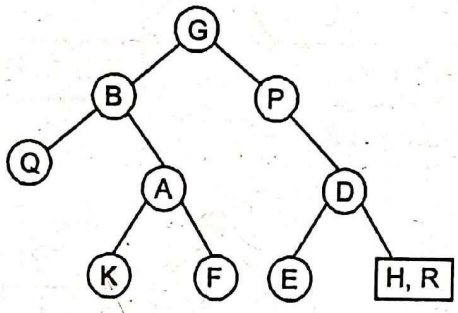


Now in preorder the next node is P so the P will be the parent node and node, E,D,H,R are the right side node in inorder.

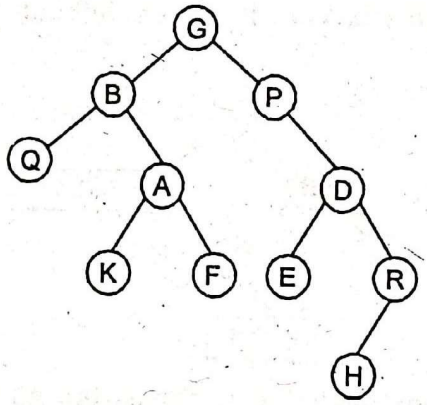


Now next node in preorder is D so the D will be the parent node and In inorder left node of D is E and Right subtree of D is H, R node

so



Now in preorder the next node is R so R will be the parcent node and In inorder H is the left side node of R. So the final tree is



Post order of above tree is–

Q, K, F, A, B, E, H, R, D, P, G

Q.4. Define Bubble sort and Quick sort with example. Which one is better if data set is small ? **(12.5)**

Ans. Bubble sort: Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are no. of items.

Initial stage–List is unsorted. We are going to sort this in ASCENDING order using bubble sort

5	4	3	2	1

1st Pass of the first For loop.

J=0, a[i]=5, a[j+1] = 4 (Second for loop starts working)

5	4	3	2	1

Here a[j]=5 and a[j+1] = 4 are compared. Thair positions are then interchanged.

Second for loop gets iterated. Now J = 1 a[j]=5 and a[j + 1] = 3

4	5	3	2	1

Now j=2 a[j] = 5 and a[j + 1] = 2

4	3	5	2	1

Now j=3 a[j] = 5 and a[j + 1] = 1

4	3	2	5	1

Largest element of the list is moved to last position after first pass.

4	3	2	1	5

After the first pass of first *For* loop. See the largest element of the list is moved to last position.

4	3	2	1	5

***After the second pass.* See the second largest element is moved to second last position.**

3	2	1	4	5

***After the third pass*, See the third largest element moved to..**

2	1	3	4	5

***After the fourth pass.* The list gets sorted in ascending order.**

1	2	2	1	5

So an array of 5 elements gets sorted in 5 passes. So the number of passes required to sort an array of 'n' elements using Bubble sort is = *n*

Quick Sort:

Quick sort is *a* highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than specified value say pivot based on which the partition is made and another array holds values greater than pivot value.

The quick sort partitions an array and then calls itself recursively twice to sort the resulting two subarray. This algorithm is quite efficient for large sized data sets as its average and worst case complexity are of O(nlogn) where n are no. of items.

Partitioning is illustrated on the above example.

1. The first action is to get the pivot out of the way - swap it with the last element

5, 3, 25, 6, 10, 17, 1, 2, 18, 8

2. We want larger elements to go to the right and smaller elements to go to the left.

Two "fingers" are used to scan the elements from left to right and from right to left:

[5, 3, 25, 6, 10, 17, 1, 2, 18, 8]

^ ^

i j

- While *i* is to the left of **j**, we move **i** right, skipping all the elements less than the pivot. If an element is found greater then the pivot, **i** stops
- While **j** is to the right of **i**, we move **j** left, skipping all the elements greater than the pivot. If an element is found less then the pivot, **j** stops
- When both **i** and **j** have stopped, the elements are swapped.
- When **i** and **j** have crossed, no swap is performed, scanning stops, and the element pointed to by **i** is swapped with the pivot .

In the example the first swapping will be between 25 and 2, the second between 10 and 1.

3. Restore the pivot.

After restoring the pivot we obtain the following partitioning into three groups:

[5, 3, 2, 6, 1] [8] [10, 25, 18, 17]

STEP 3. Recursively quicksort the left and the right parts.

Quick sort is better for small data elements because its average case complexity is O(n log n).

Q.5. Suppose you have an AVL Node class that stores integers: **(12.5)**

Public class AVLNode {

Public int item; public AVLNode left;

Public AVLNode right;

Public AVLNode (int *i*, AVLNode 1, AVLNode *r*)

item = *i*; left = 1; right = *r*; ;} :

Write a complete method that takes a height h, and returns a reference to the roof of an AVL tree of height h that contains the minimum number of nodes. The integers stored in the nodes (the item instance variables) should satisfy the binary search tree property, and they should all be distinct (but they do not have to be consecutive). You can define helper methods and/or classes if you wish.

Ans. (Data Missing)

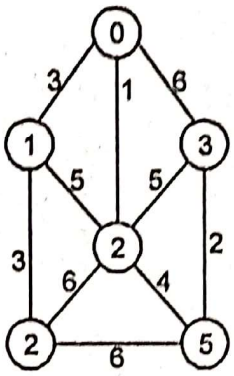
Q.6. Write Algorithm for Prims Algorithm. Show how to generate MST using Prims algorithm. **(12.5)**

Ans. Prim's Algorithm: Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph.

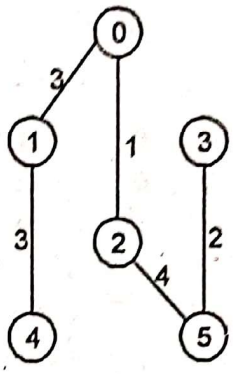
It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

This algorithm is directly based on the MST(minimum spanning tree) property.

Example:



A simple weighted graph



Minimum-cost spanning tree

Prim's Algorithm

MST-PRIM(G, w, r)

for each u V [G]

do key[u] ← ∞

π[u] ← NIL

key[r] ← 0

Q ← V[G]

while $Q \neq \varnothing$

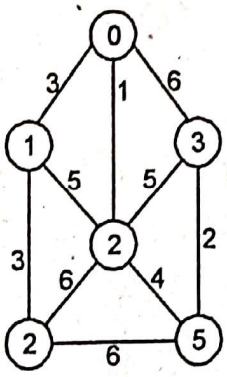
do u ← EXTRACT-MIN(Q)

for each v Adj[u]

do if v Q and w(u, v) < key[v]

then π[v] ← u

key[v] ← w(u, v)



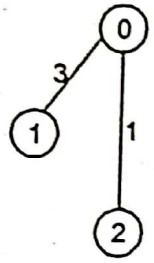
Procedure for finding Minimum Spanning Tree

Step 1.

No. of Nodes	0	1	2	3	4	5
Distance	0	3	1	6	∞	∞
Distance From		0	0	0		

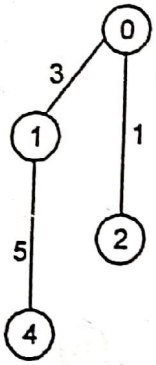
Step 2.

No. of Nodes	0	1	2	3	4	5
Distance	0	3	0	5	6	4
Distance From		0		2	2	2



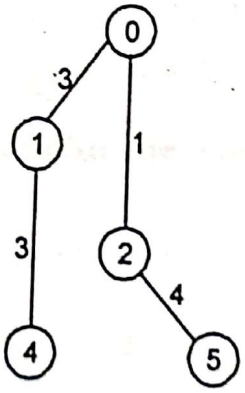
Step 3

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	3	4
Distance From				2	1	2



Step 4

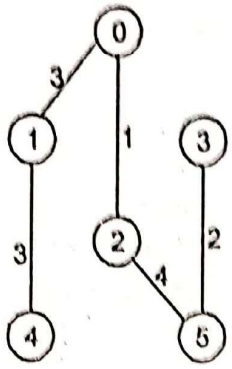
No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	0	4
Distance From				2		2



Step 5.

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	3	0	0
Distance From				2		2

Minimum Cost = 1 + 2 + 3 + 3 + 4 = 13



Minimum cost = 1 + 2 + 3 + 3 + 4 = 13

Q.7 . Define following collision resolution method with example. **(12.5)**

(a) Open addressing (b) Liner Probe (c) Quadratic probe

Q.7 . (a) Open Addressing.

Ans. Open Addressing: In Open Addressing all entry records are stored in the bucket array itself. When a new entry has to be inserted, the buckets are examined, starting with the hashed-to slot and proceeding in some *probe sequence*, until an unoccupied slot is found. When searching for an entry, the buckets are scanned in the same sequence, until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table. The name "open addressing" refers to the fact that the location ("address") of the item is not determined by its hash value. (This method is also called **closed hashing**). Well-known probe sequences include:

- Linear probing, in which the interval between probes is fixed (usually 1).
- Quadratic probing, in which the interval between probes is increased by adding the successive outputs of a quadratic polynomial to the starting value given by the original hash computation.
- Double hashing, in which the interval between probes is computed by a second hash function.

Q.7 . (*b*) Linear probe.

Ans. Linear Probe: When using a linear probe, the item will be stored in the next available slot in the table, assuming that the table is not already full.

This is implemented via a linear search for an empty slot, from the point of collision. If the physical end of table is reached during the linear search, the search will wrap around to the beginning of the table and continue from there.

If an empty slot is not found before reaching the point of collision, the table is full.

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Add the keys 10, 5, 15 to the previous table.

Hash key = key % table size

2 = 10 % 8
5 = 5 % 8
7 = 15 % 8

[0]	72
[1]	15
[2]	18
[3]	43
[4]	36
[5]	10
[6]	6
[7]	5

A problem with the linear probe method is that it is possible for blocks of data to form when collisions are resolved. This is known as **primary clustering.**

This means that any key that hashes into the cluster will require several attempts to resolve the collision.

Q.7. (c) Quadratic Probe

Ans. Quadratic Probe: To resolve the primary clustering problem, **quadratic probing** can be used. With quadratic probing, rather than always moving one spot, move i^2 spots from the point of collision, where i is the number of attempts to resolve the collision.

89 % 10 = 9

15 % 10 = 8

49 % 10 = 9 – 1 attempt needed – 1^2 = 1 spot

58 % 10 = 8 – 3 attempt – 3^2 = 9 spots

69 % 10 = 9 – 2 attempt – 2^2 = 4 spots

[0]	72
[1]	
[2]	
[3]	69
[4]	
[5]	
[6]	
[7]	58
[8]	18
[9]	89

Limitation: At most half of the table can be used as alternative locations to resolve collisions.

This means that once the table is more than half full, it's difficult to find an empty spot. This new problem is known as **secondary clustering** because elements that hash to the same hash key will always probe the same alternative cells.

Q.8. (a) Write short notes on any two of following with example: (12.5)

(a) 2-way merge sort (6.25 × 2 = 12.5)

(b) B+ Tree

(c) Circular List.

Ans. Two-way merge sort: A 2-Way Merge, or a binary merge, has been studied extensively due to its key role in Merge sort. An example of such is the classic merge

that appears frequently in merge sort examples. The classic merge outputs the data item with the lowest key at each step; given some sorted lists, it produces a sorted list containing all the elements in any of the input lists, and it does so in time proportional to the sum of the lengths of the input lists. There are algorithms that exist that can operate in better than linear times such as the Hwang-Lin Merging Algorithm.

Example: Assume, that we have two arrays A [0..m – 1] and B [0.. n – 1] that are sorted in ascending order and we want to merge them into an array C[0..m + n -1] with the same order.

1. Introduce read-indices **i, j** to traverse arrays A and B, accordingly. Introduce write-index k to store position of the first free cell in the resulting array. By default **i** $= j = k = 0$.

2. At each step: if both indices are in range (**i** < m and **j** < **n**), choose minimum of (A [**i**], B[**j**]) and write it to C[**k**]. Otherwise go to step 4.

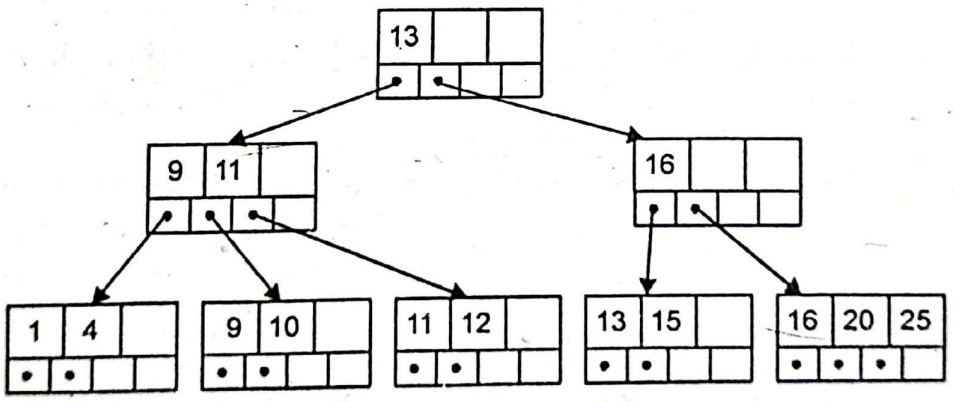
3. Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.

4. Copy the rest values from the array, which index is still in range, to the resulting array.

Q.8. (*b*) B+ Tree

Ans. A B^+ tree maintains the following invariants:

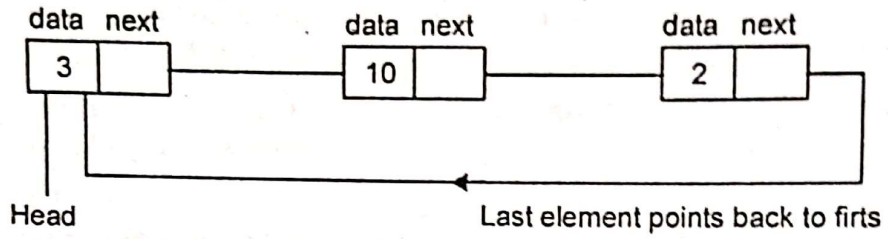
- Every node has one more references than it has keys.
- All leaves are at the same distance from the root.
- For every non-leaf node N with k being the number of keys in N: all keys in the first child's subtree are less than N's first key; and all keys in the ith child's subtree ($2 \leq i \leq k$) are between the (i – 1)th key of n and the ith key of n.
- The root has at least two children.
- Every non-leaf, non-root node has at least floor (d/2) children.
- Each leaf contains at least floor (d/2) keys.
- Every key from the table appears in a leaf, in left-to-right sorted order.



Q.8. (*c*) Circular List

Ans. Circular Linked List is little more complicated linked data structure. In the circular linked list we can insert elements anywhere in the list whereas in the array we cannot insert element anywhere in the list because it is in the contiguous memory. In the circular linked list the previous element stores the address of the next element and the last element stores the address of the starting element. The elements points to

each other in a circular way which forms a circular chain. The circular linked list has a dynamic size which means the memory can be allocated when it is required.



Circular lists are usually preferable to non-circular ones, for two reasons:

- They provide easy access to both ends of a list: notice that we do not need the first pointer in the header, because we can easily reach the first node with L->last->next
- The code for processing a circular list is often simpler than the code for processing a non-circular list-mainly because you don't have to constantly check if the next node is defined-it is always defined.

FIRST TERM EXAMINATION [SEPT. 2016]
THIRD SEMESTER [B.TECH]
DATA STRUCTURE [ETCS–209]

Time : 1.30 hrs. **M.M. : 30**

Note: *Attempt any three questions including Q.no. 1 which is compulsory.*

Q.1. (a) What are two main factors for measuring the performance of algorithm? Explain in brief. **(2)**

Ans. Space and Time complexities are the parameters by which we can analyze the performance of a program or algorithm.

Space Complexity: It is the amount of memory it needed to run to completion. It has a fixed pact that includes space for the cock, space for simple variable and fixed size component variables, constants etc. and a variable part that includes the space needed by component variable whose size is dependent on the particular problem instance being solved and the stack space.

Time Complexity: It is the amount of computer time needs to run to completion. It is the sum of the compile time & the run time. The compile time does not depend on the instance characteristics major concern is the run time of a program.

Q.1. (b) What do you mean by garbage collection? **(2)**

Ans. Garbage Collection: Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory block and reallocates storage for reuse. The primary purpose of garbage collection is to reduce memory leaks.

Q.1. (c) Define time-space trade off? **(2)**

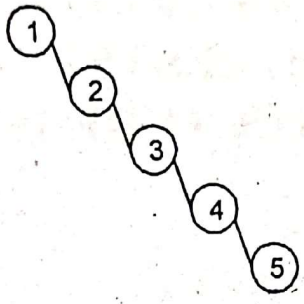
Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q.1. (d) How the resulting binary search tree will look like if the input is already sorted in non-decreasing order. **(2)**

Ans. If all nodes are already sorted in non decreasing order then all nodes are inserted into the right side of a binary search tree, for example →

nodes are 1, 2, 3, 4, 5

the BST will look like →



Q.1. (c) Number of nodes in a complete tree is 1000000. Find its depth? (2)

Ans. The depth of complete binary tree is log(n) where no. of nodes is n. So the depth of complete binary tree which has the 1000000 nodes is.

$$\log(1000000) = 6$$

Q.2. (a) Write a program/algorithm to evaluate a postfix expression using stack? Consider the following infix expression and convert into postfix expression using stack. (A + (B*C – (D\E ^ F)*G)*H) (2+3)

Ans. Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator O, pop twice and get A and B respectively. Calculate BOA and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Character Scanned	Stack	Reverse polish Notatic Postfix Notation
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
(	(+(*	ABC
–	(+(–	ABC*
(	(+(–(	ABC*
D	(+(–(	ABC* D
/	(+(–(/	ABC * D
E	(+(–(/	ABC * DE

^	(+ (– (/^	ABC * DE
F	(+ (– (/^	ABC * DEF
)	(+ (–	ABC * DEF ^
*	(+ (– *	ABC * DEF ^ /
G	(+ (– *	ABC * DEF ^ / G
)	(+	ABC * DEF ^ / G * –
*	(+ *	ABC * DEF ^ / G * –
H	(+ *	ABC * DEF ^ / G * –
)	—	ABC * DEF ^ / G * – H * +
		Reverse polish notation

Q.2. (b) Write a procedure/program to reverse a singly linked list without using any more memory? **(5)**

Ans. Refer Q.2. (a) End Term Examination 2016.

Q.3. (a) Suppose multidimensional arrays A and B are declared as A (–2.2, 2.22) and B(1 : 8, –5 : 5, –10 : 5) stored in column major order. **(2+2+3)**

(*i*) Find the length of each dimensional of A and B

(*ii*) The number of elements in A and B.

(*iii*) Consider the elements B[3, 3, 3] in B. Find the effective indices E1, E2, E3 and the address of the element assuming Base (b) =400 and there are w=4 words per memory location.

Ans. (*i*) The length of a dimension is obtained by:

Length = upper bound – Lower bound + 1

Hence the length Li of the dimensions of A are:

L1 = 2 – (–2) + 1 = 5 and L2 = 22 – 2 + 1 = 21

L1 = 8 – 1 + 1 = 8, L2 = 5 – (–5) + 1 + 11, L35 – (–10) + 1 = 16

(*ii*) Accordingly A has 5*21 = 105 elements and B has 8*11*16 = 1408 elements.

(*iii*) The effective index Ei is obtained from Ei = ki–LB, where ki is the given index and LB is the lower bound.

Hence E1 = 3 – 1 = 2, E2 = 3 – (–5) = E3 = 3 – (–10) = 13

Considering *B* is stored in column-major order, the address of *B* [3,3,3] will be calculated as.

E3L2 = 13*11 = 143, E2L2 + E2 = 143 + 8 = 151

(E3L2 + E2) L1 = 151*8 = 1208, (E3L2 + E2) L1 + E1 = 1208 + 2 = 1210

Therfore address of B [3, 3, 3] = 400 + 4(1210) = 400 + 4840 = 5240.

Q.3. (b) What is an algorithm/program for deleting a node at a given location in circular linked list. **(3)**

Ans.

Deletion of node at given Location.

Step 1. If FIRST = NULL then write underflow and stop.

Step 2. Read DATA as information of node to be deleted.

Step 3. PTR = FIRST

Step 4. Repeat step 5 while info (PTR) ≠ DATA

Step 5. CPT = PTR

PTR = LINK (PTR)

Step 6. LINK (CPT) = LINK (PTR)

Step 7. LINK (PTR) = AVAIL

AVAIL = PTR

Step 8. Stop.

Q.4. (a) Why height balancing of tree is required? Create AVL tree of following:

a, z, b, y, c, x, d, w, e, v, f **(2+6)**

Ans. The first balanced binary search tree or height balancing tree was the AVL tree with has an additional balancing condition. The simplest idea is to require that the left and right subtrees have the same height, so height balancing tree is required.

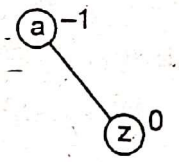
AVL tree of nodes– *a, z, b, y, c, x, d, w, e, v, f.*

Creation of AVL Tree

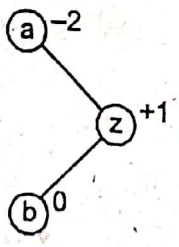
Nodes: *a, z, b, y, c, x, d, w, e, v, f*

Insert *a*

Insert *z*

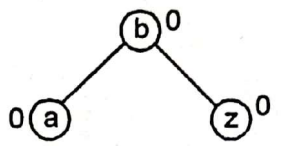


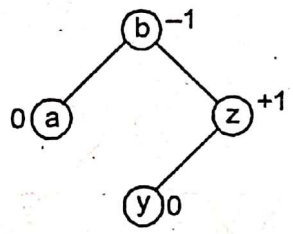
Insert *z*



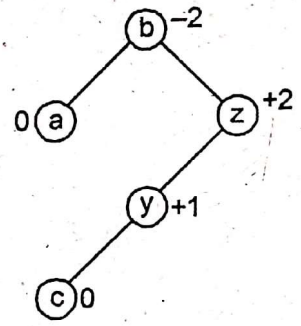
There is Balance factor of a is –2 so after R-L rotation the tree is.

Insert y

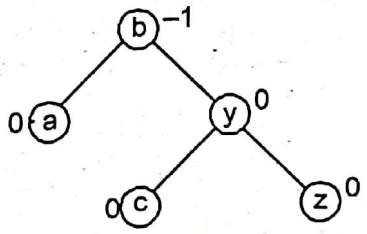




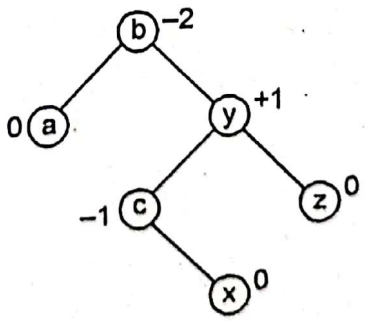
Insert c



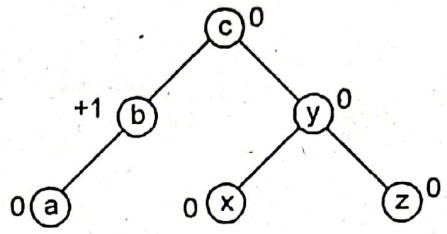
There is Balance factor of z is + 2 so after $L - L$ rotation the tree is –



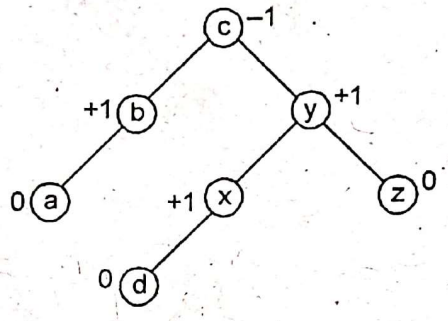
Insert x



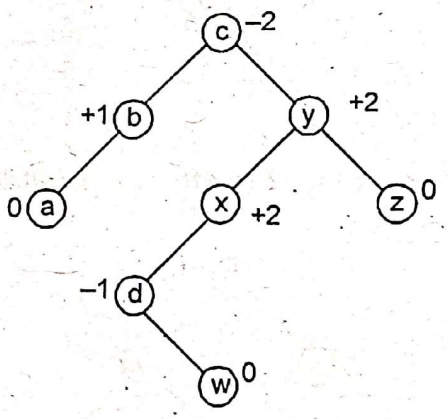
There is the Balance factor of b is – 2 So after L–R rotation the tree is– .



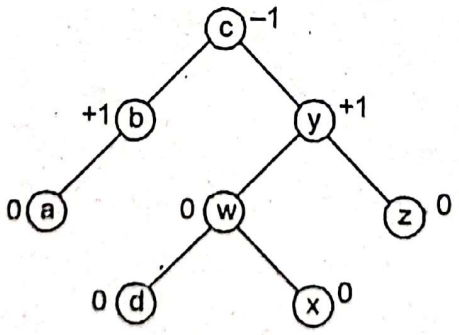
Insert d



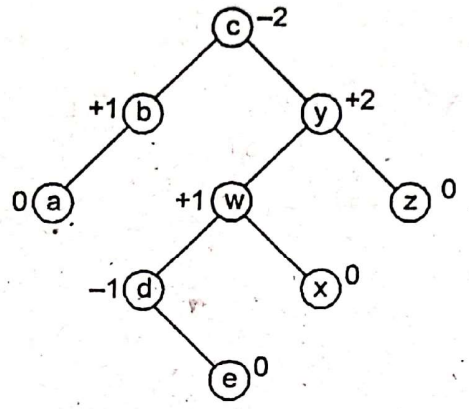
Insert w



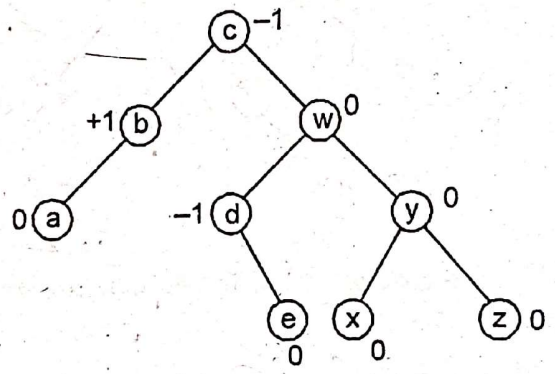
After Insertion of w the Balance factor is unbalanced so after applying the L-R rotation the tree is–



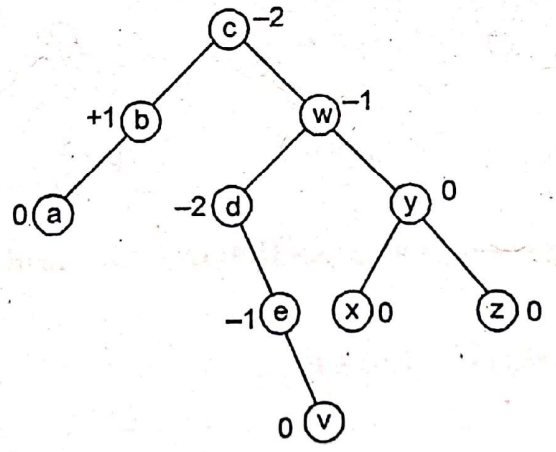
Insert *e*



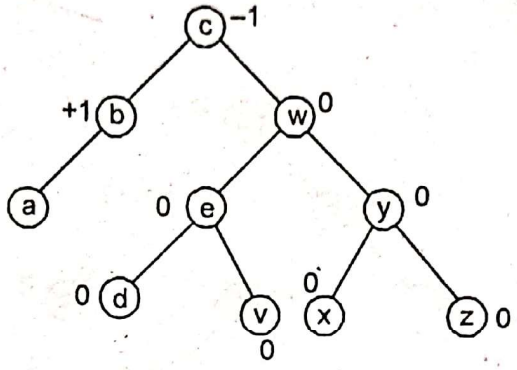
After Insertion of node *e* the tree is unbalanced so after applying the *L-L* rotation the tree is–



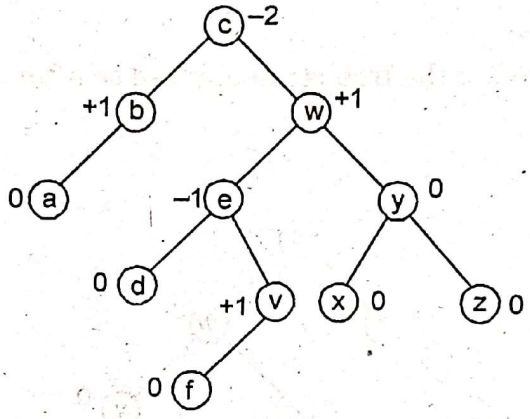
Insert *v*



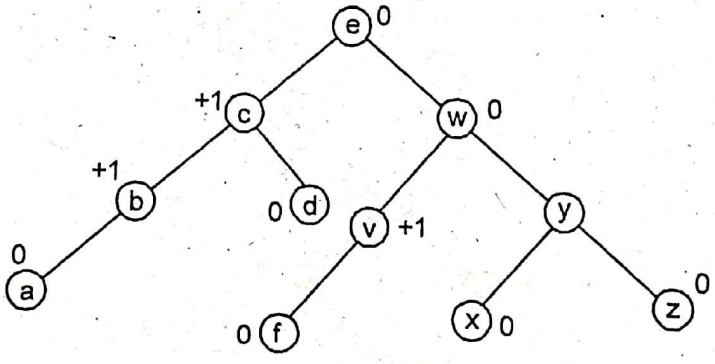
After Insert the node *v* the tree is unbalanced so after apply the R.R rotation the tree is–



Insert *f*



After Insert the node *f* the tree is unbalanced so after apply the *R–L* rotation the final tree is–



Q.4. (b) Following sequence gives the preorder and inorder of the Binary Tree T. **(2)**

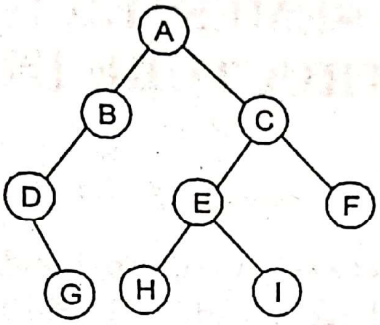
PREORDER : A B D G C E H I F

INORDER : D G B A H E I C E

Construct a binary tree for the above sequence.

Ans. P R E O R D E R : A B D G C E H I F

INORDER: D G B A H E I C F



END TERM EXAMINATION [DEC. 2016]
THIRD SEMESTER [B.TECH]
DATA STRUCTURE [ETCS–209]

Time : 3 hrs. **M.M. : 75**

Note: *Attempts any five questions including Q.no. 1 which is compulsory.*

Q.1. (*a*) Differentiate between merge Sort and Quick Sort. (5)

Ans. (1) In worst case, Merge sort still maintain a complexity of o (n log n) while quick sort becomes O (n^2).

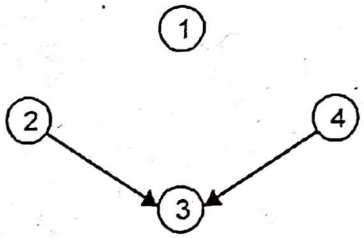
(2) In quick sort there is pivot selection but there is no pivot selection in merge sort. The array are partitioned equally which continues recursively and finally its merged.

(3) Merge sort is not an in place sorting while quick sort is (without regard to the internal stack space that taken by each recursive algorithm).

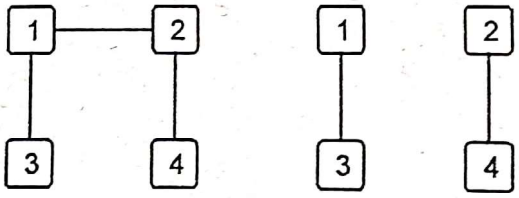
Q.1. (*b*) Explain terms: sink node, source node, connected and strongly connected graph. (5)

Ans. Sink Node: A sink node is a node such that no edge emerges out of it or we can say that a sink vertex is a vertex with outdegree zero. In this figure the node 1 and node 3 are the sink nodes.

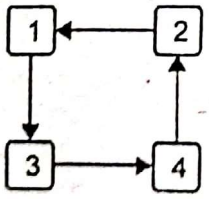
Source Node: A Source node is a node such that no edge emerges. In of it or we can say that A source vertex is a vertex with indegree zero. In figure the node 1, node 2 and node 4 are the source node.



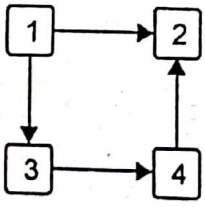
An undirected graph is connected if there is a path from every node to every other node. For example.



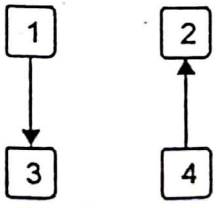
A directed graph is strongly connected if there is a path from every node to every other node. A directed graph is weakly connected if, treating all edges as being undirected, there is a path from every node to every other node. For example.



Strongly connected



Weakly connected but not strongly connected

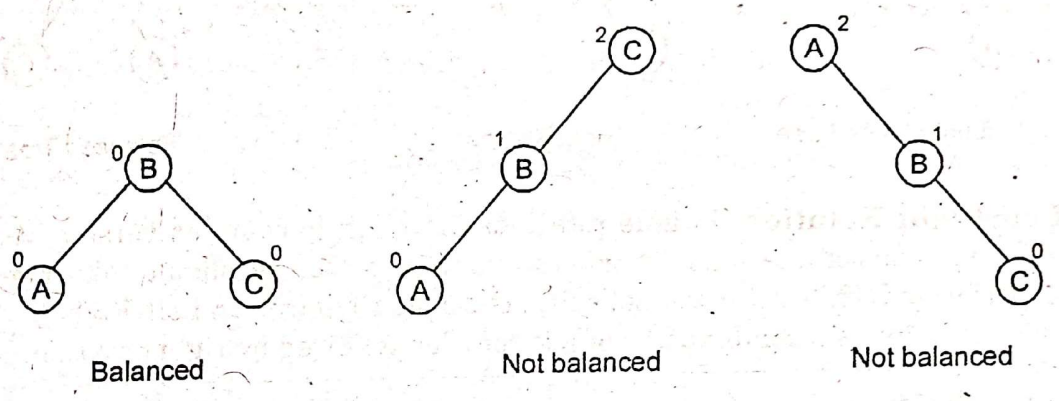


neither weakly nor strongly connected

Q.1.(c) What is a height balanced tree? Explain with the help of suitable example? **(5)**

Ans. AVL Trees are height balancing binary search tree. AVL tree checks the height of the left and the right subtrees and assures that the difference is not more than 1. This difference is called the Balance factor.

Here we see that the first tree is balanced and the next two trees are not balanced-



In the second tree, the left subtree of *C* has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of *A* has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

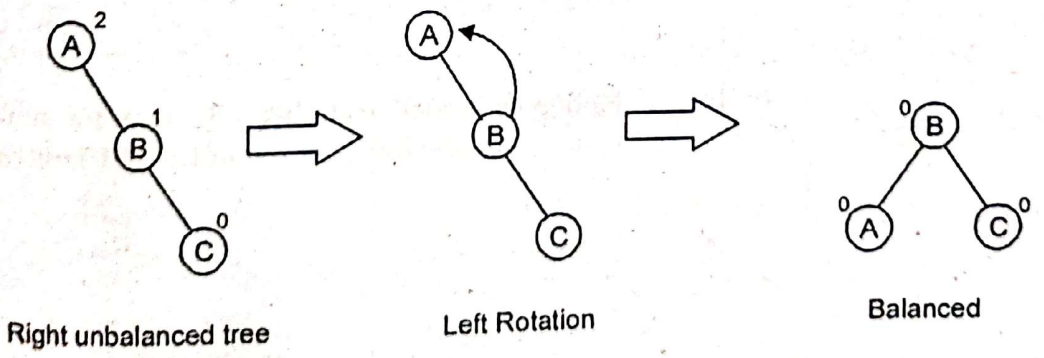
Balance factor = height (left-subtree) – height (right-subtree)

If the difference in the height of left and right sub-tree is more than 1, the tree is balanced using some rotation techniques.

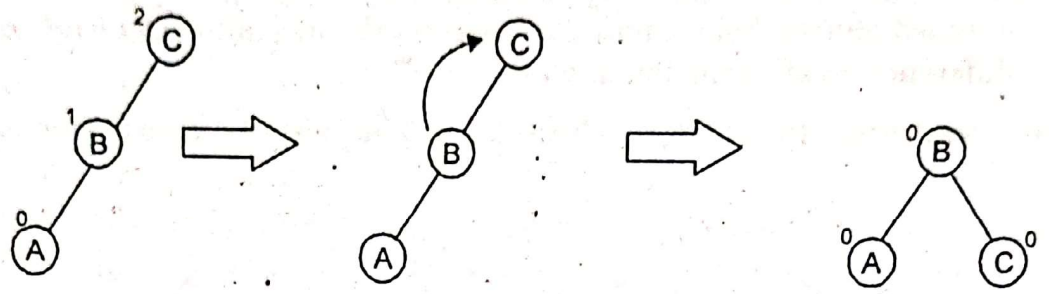
AVL Rotations: To balance itself, an AVL tree may perform the following four kinds of rotations:

- Left rotation
- Right rotation
- Left-Right rotation
- Right-left rotation

Left Rotation: If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation.

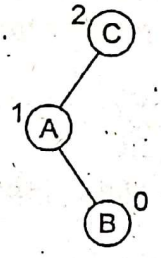
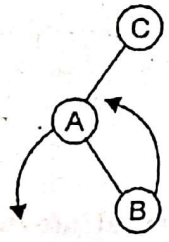
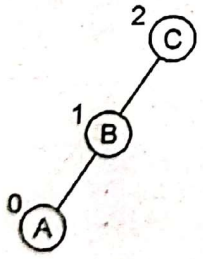


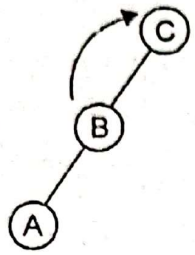
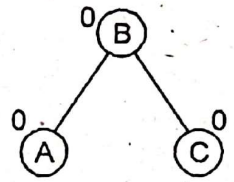
Right Rotation: AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



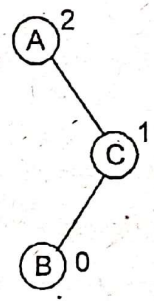
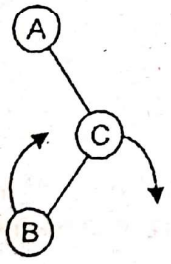
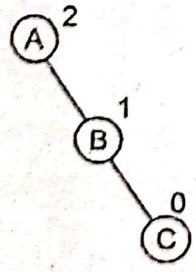
Left unbalanced tree Right Rotation Balanced Tree

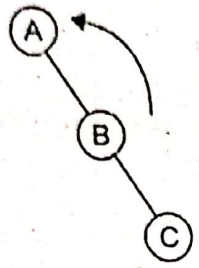
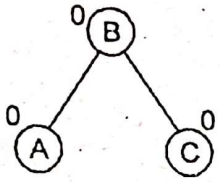
Left-Right Rotation: Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

State	Action
	A node has been inserted into the right subtrece of the left subtree. This makes *C* an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of *C*. This, makes *A*, the left subtree of *B*.
	Nodes *C* is still unbalanced, however now, it is because of the left-subtree of the left-subtree

	We shall now Right-Rotate the tree, making *B* the new root node of this subtree. *C* now becomes the right subtree of its own left subtree.
	The tree is now balanced.

Right-Left Rotation: The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes *A*, an unbalanced node with balance factor 2.
	First, we perform the right rotation along *C* node, making *C* the right subtree of its own left subtree B. Now, B becomes the right subtree of *A*.
	Node *A* is still unbalanced because of the right subtree of its right subtree and requires a left rotation.

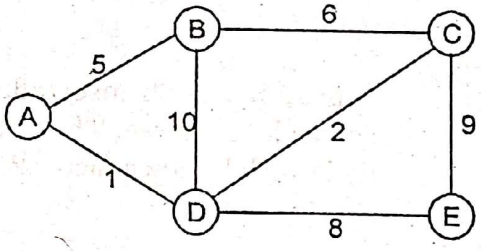
	A left rotation is performed by making *B* the new root node of the subtree. *A* becomes the left subtree of its right subtree *B*.
	The tree is now balanced.

Q.1.(d) What are the different ways of representing a graph? Generate following graph in them. **(5)**

Ans. There are total 3 ways of representing a graph:

1. Adjacency matrix: Adjacency matrix is a 2D array of size V × V where V is the no. of vertices in a graph. For Example:

Given graph is:



Adjacency matrix

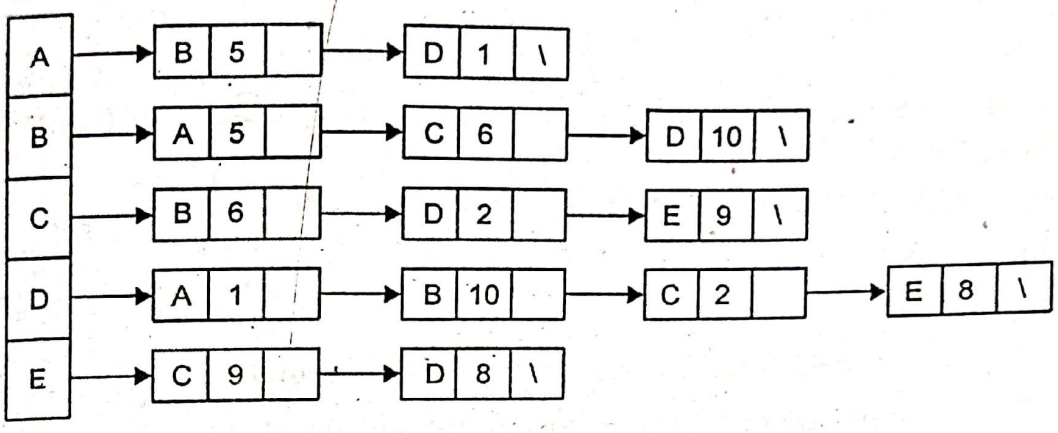
	A	*B*	*C*	*D*	*E*
A	0	5	0	1	0
B	5	0	6	10	0
C	0	6	0	2	9
D	1	10	2	0	8
E	0	0	9	8	0

2. Incidence matrix: Incidence matrix is a 2D array of V × E, where V is the no. of vertices and E is the total no. of edges.

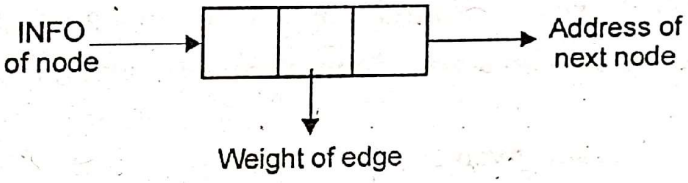
The incidence matrix of above graph is:

	1	5	2	6	8	9	10
A	1	1	0	0	0	0	0
B	0	1	0	1	0	0	1
C	0	0	1	1	0	1	0
D	1	0	1	0	1	0	1
E	0	0	0	0	1	1	0

3. List Representation: In this representation the List is the array [] of linked list, where array size is same as number of vertices in the graph. Every vertex has a linked list. Each node in this linked list represents the reference to the other vertices which share an edge with the current vertex. The weights can also be stored in the linked list.



This is the list representation of above graph and the structure of node has 3 fields:



Q.1. (e) What is meant by term Row Major Order and Column Major Order in Array? How do we calculate the address of an element in a row major two dimensional array? **(5)**

Ans. Array Representation:

- **Column-major**
- **Row-major**

Arrays may be represented in Row-major form or column-major form. In row-major form, all the elements of the first row are printed, then the elements of the second row and so on upto the last row. In column-major form all the elements of the first column are printed, then the elements of the second column and so on upto the last column. The 'C' program to input an array of order m × n and print the array contents in row major and column major is given below. The following array elements may be entered during run time to test this program:

Input: Rows: 3, Columns: 3;

1 2 3

4 5 6

7 8 9

Output:

Row Major:

1 2 3

4 5 6

7 8 9

Column Major:

1 4 7

2 5 8

3 6 9

Row Major System: The address of a location in Row Major System is calculated using the following formula:

Address of A [I] [J] = B + W* [N* (I-Lr) + (J-Lc)]

Where

B = Base address

I = Row subscript of element whose address is to be found

J = Column subscript of element whose address is to be found

W = Stroage size of one element stored in the array (in byte)

Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)

Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)

M = Number of row of the given matrix.

N = Number of column of the given matrix.

Q. 2. (a) Write a procedure/ program to reverse a singly linked without using any more memory? **(6)**

Ans. Reverse a singly linked list.

Step 1 PTR = FIRST

Step 2 TPT = NULL

Step 3 Repeat step 4 while PTR! = NULL

Step 4 (a) CPT = LINK (PTR)

(b) LINK (PTR) = TPT

(c) TPT = PTR

(d) PTR = CPT

Step 5 Stop.

Q.2. (b) Write a program/algorithm to evaluate a postfix expression using stack? Consider the following infix expression and convert into postfix expression using stack. (A + (B*C – (D\E ^ F)*G)*H) **(6.5)**

Ans. Refer Q.2. (a) First Term Examination 2016.

Q. 3. (a) Consider the Linear Array aaa(5 : 50), bbb(–5 : 10) and cc(18).

(*i*) Find the number of elements in each array.

(*ii*) Suppose Base (aaa) = 300 and w = 4, words for memory cell for aaa. Find the address of aaa[15], aaa[35] and aa[55]. **(8.5)**

Ans. (*i*) No. of elements in array is equal to the length of array. So

Length = UB – LB + 1

Thus Length of (aaa) = 50 – 5 + 1 = 46

Length of (bbb) = 10 – (– 5) + 1 = 16

Length of (cc) = 18 – 1 + 1 = 18

(*ii*) We know the formula

LOC (A [k]) = base (A) + W (k – LB)

here W = 4, base (aaa) = 300

∴ LOC (aaa [15]) = 300 + 4 (15 – 5)

= 300 + 4 × 10 = 340

LOC (aaa [35]) = 300 + 4 (35 – 5)

= 300 + 4 × 30 = 420

LOC (aa [55]) - Here aa is not an array the actual array is aaa. So LOC (aa [55]) is not possible.

Q. 3. (b) Derive the Time Complexity of the linear search in average and worst case. **(4)**

Ans. Time Complexity of Linear Search: We can have three cases to analyze an algorithm:

(1) Wrost case

(2) Average case

(3) Best case

Wrost case analysis (Usually Done): In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search () functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be Θ (n).

Average Case Analysis (Sometimes done): In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n + 1). Following is the value of average case time complexity.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{0((n+1)\times(n+2)/2)}{(n+1)}$$

$$= \Theta(n)$$

Q. 4. Explain the following terms used in tree with example- **(12.5)**

(a) Terminal node/leaf. (b) Sibling/ brother (c) Level number (d) Height / Depth (e) Degree of Node (f) Generation (g) Root (h) Ancestors and Descendent.

Ans. Tree Terminology

Let us consider a tree:

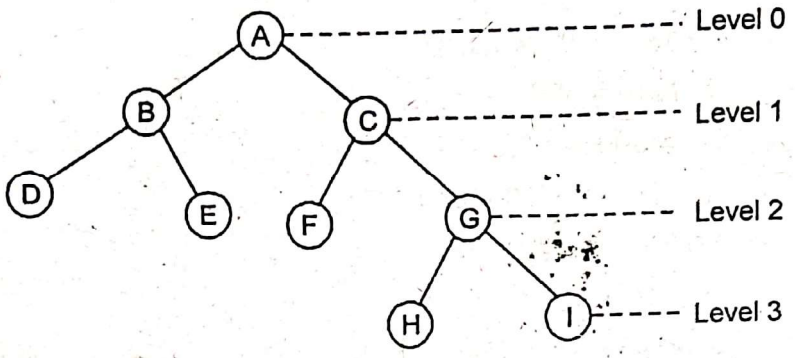


Fig.1.

(a) Terminal Node/ Leaf: A Leaf node which does not have any child node. In Fig. D, E, F, H and I are the leaf nodes.

(b) Sibling/ Brother: The child node of a given parent node are called sibling. In Fig. node D, E are siblings, node H, I are siblings.

(c) Level Number: The level of a node is the no. of edges along the unique path between it and the root. The level of the root is defined as zero. If a node is at level I then its child is at level $(I+1)$ and its parent is at level $(I-1)$. In Fig.

The root node A is at level 0.

Node B and C are at level 1.

Node D, E, F, G are at level 2.

Node H and I are at level 3.

(d) Height/ Depth: The height of a rooted tree is the maximum level of node +1 in the tree. The height of Empty tree is zero. The height of a tree containing single node is 1.

Formula to find the height of a tree is:

$$h = I_{max} + 1$$

In Fig.

$$h = I_{max} + 1 = 3 + 1 = 4$$

Where I_{max} is the maximum level of a tree.

(e) 7 Degree of a node: In a tree data structure, the total no. of children of a node is called the degree of that node. In simple words, the degree of a node is total no. of children it has. In Fig. 4.1 the degree of node A is 2 the degree of node B is 2 and So on because these nodes has the 2 children.

(f) Generation: Each node in a binary tree is assigned a level number , as follows. The root of the tree is assigned the level number zero, and every other node is assigned a level number which is one more than the level number of its parent. Further more, those nodes with the same level number are said to belong to the same generation.

(g) Root: In a tree data structure, the first node is called as root node. Every tree must have root node. In any tree, there must be only one root node. Root node is the origin of tree data structure. In Fig. the node A is the root node.

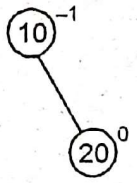
(h) Ancestor and Descendant: An ancestor is any element which is connected further up in the hierarchy tree no matter how many levels higher. An ancestor is a parent, grandparent, great grandparent and so on.

A descendant refers to any element that is connected lower down the hierarchy tree no matter how many levels lower. A descendant is a child, grandchild, great grand child and so on. In Fig. node *A* is the ancestor of node *B* and *F* and *C* node, and node *C* has the descendants *G*, *H*, *I* nodes.

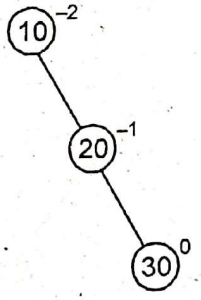
Q. 5. Construct the AVL tree if following elements are inserted in order.
10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 130, 15 **(12.5)**

Ans. Step 1 Insert 10

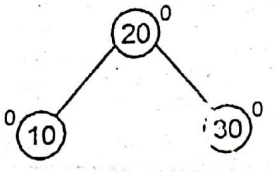
Step 2 Insert 20



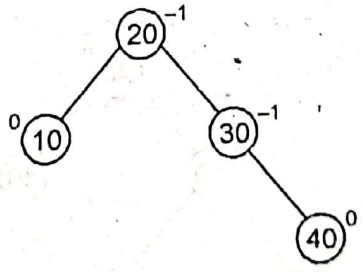
Step 3 Insert 30



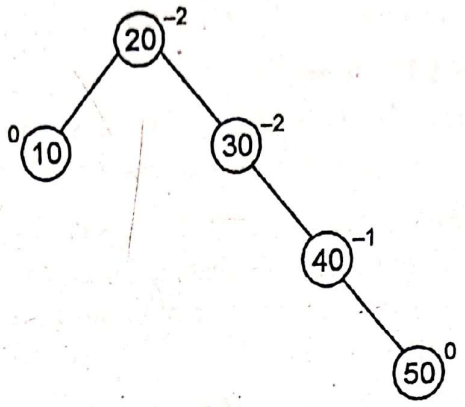
Now the tree is unbalance so after rotation the tree is:



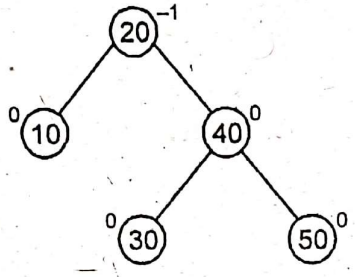
Step 4 Insert 40



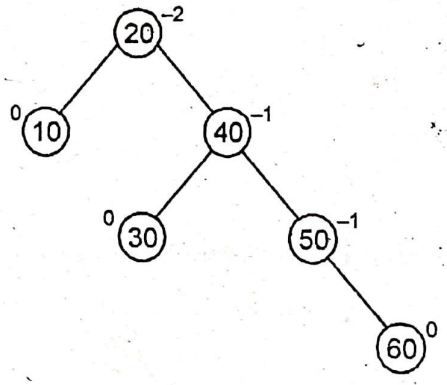
Step 5 Insert 50



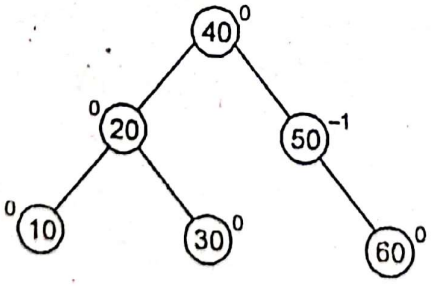
Now the tree is unbalance so after rotation the tree is:



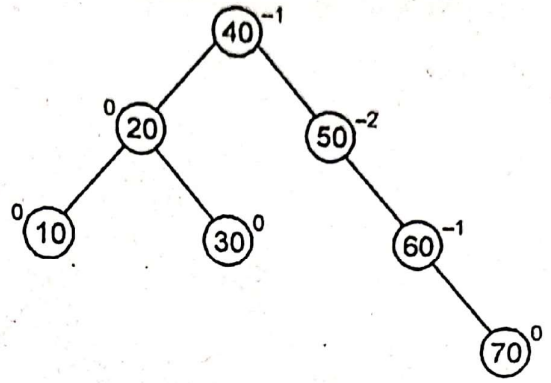
Step 6 Insert 60



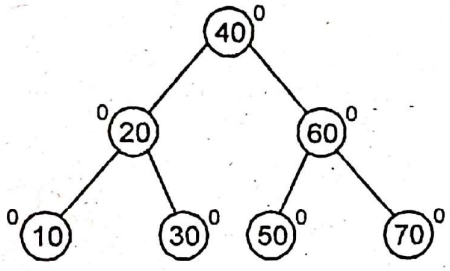
Now the tree is unbalance so after rotation the tree is:



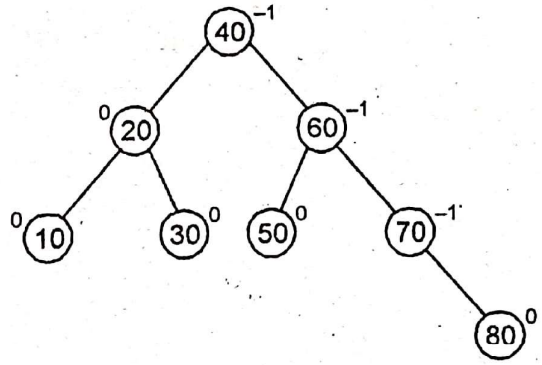
Step 7 Insert 70



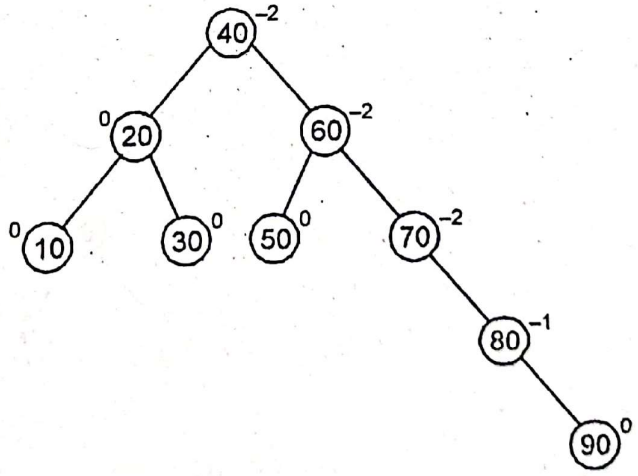
Now the tree is unbalance so after rotation the tree is:



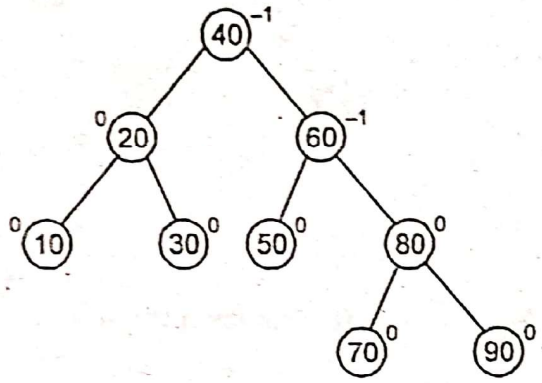
Step 8 Insert 80



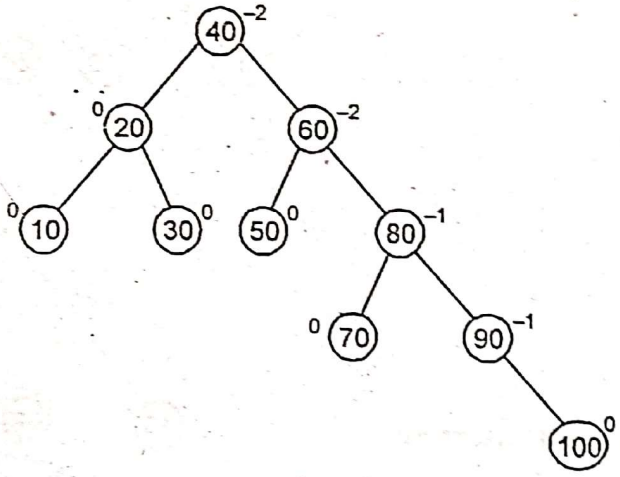
Step 9 Insert 90



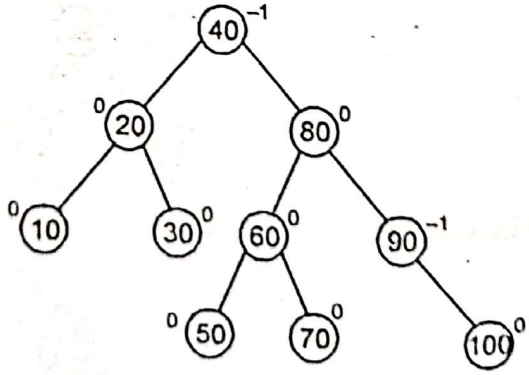
Now tree is unbalance so after rotation the tree is:



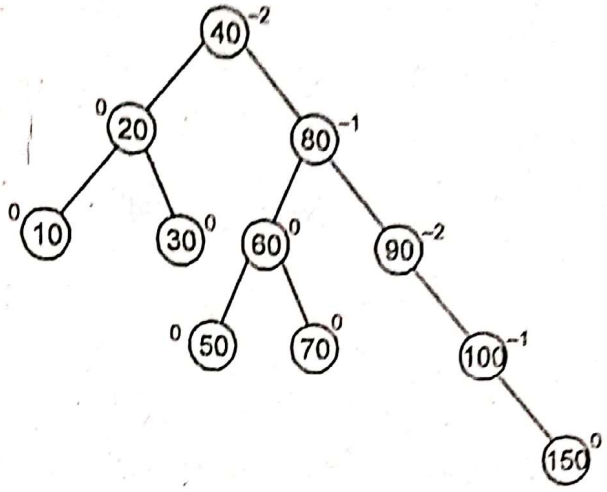
Step 10 Insert 100



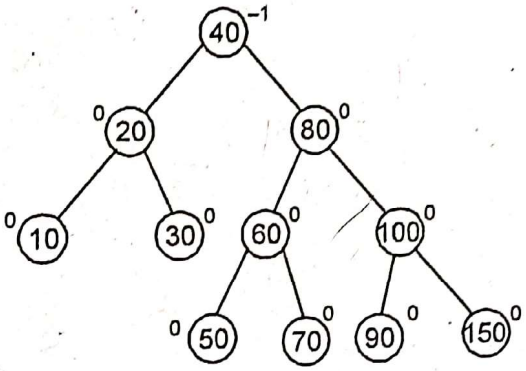
Now the tree is unbalance so after rotation the tree is:



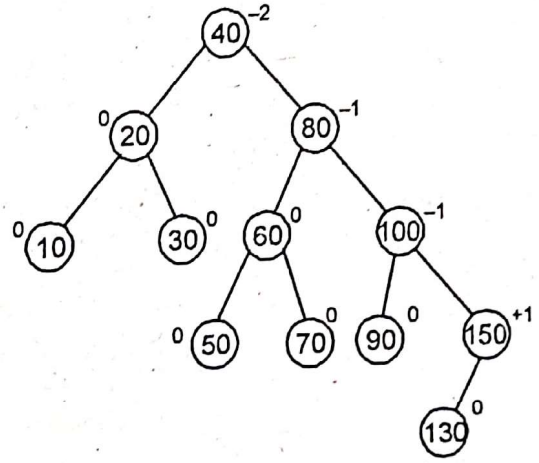
Step 11 Insert 150



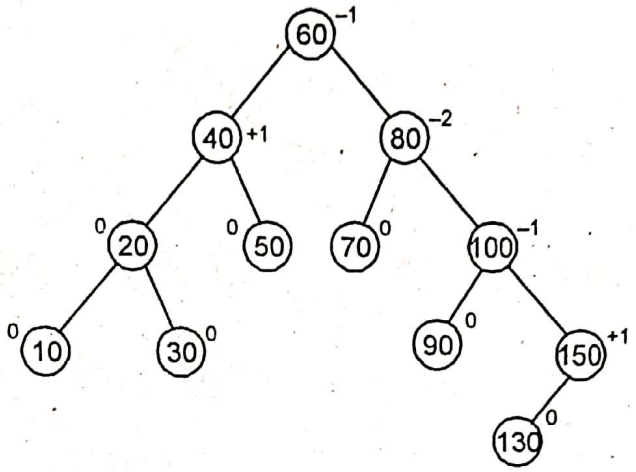
Now the tree is unbalance so after rotation the tree is:



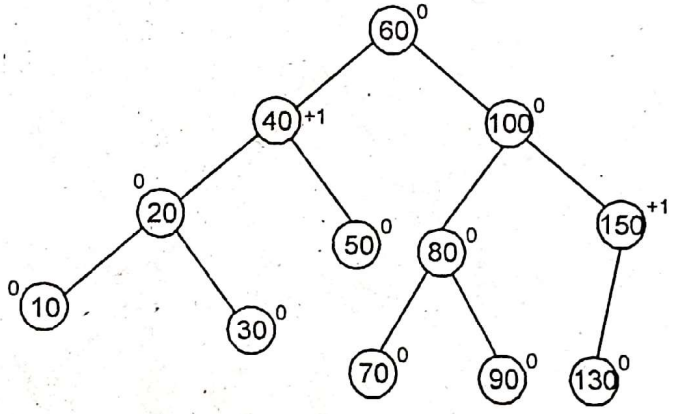
Step 12 Insert 130



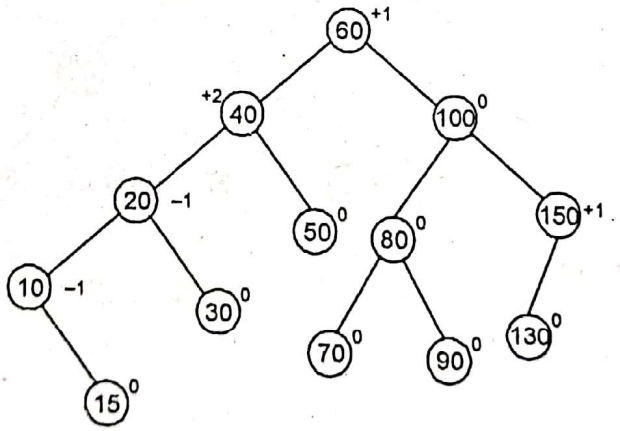
Now the tree is unbalance so after rotation the tree is:



At this time the tree is again unbalance so after another rotation the tree is:

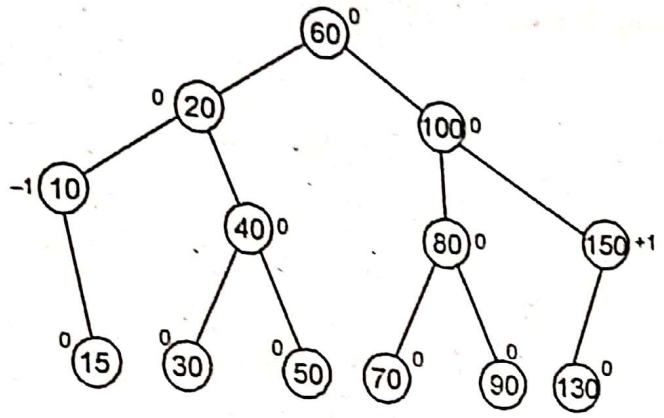


Step 13 Insert 15



Now the tree is unbalance so after rotation the tree is:

HARDIK SETHI



This is the final tree.

Q. 6. What is the role of Binary Search Tree? Give the algorithm for insertion and deletion of a node in binary search tree. Give an example. **(12.5)**

Ans. Binary Search Tree: A Binary Search Tree (BST) is a tree in which all the nodes follows the below-mentioned properties:

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-tree into two segments; the left sub-tree and the right sub-tree and can be defined as -

Left_subtree (keys) $\leq$ node (key) $\leq$ right_subtree (keys).

Insertion in Binary Search Tree:

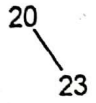
- Check whether root node is present or not (tree available or not). If root is NULL, create root node.
- If the element to be inserted is less than the element present in the root node, traverse the left sub-tree recursively until we reach T->left/ t->right is NULL and place the new node at T->left (key in new node<key in T)/T-> right (key in new node>key in T).
- If the element to be inserted is greater than the element present in root node, traverse the right sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T-> left/ T->right.

Example: Insert 20 into the Binary Search Tree.

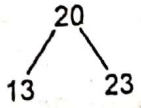
Tree is not available. So, create root node and place 10 into it.

20

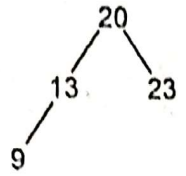
Insert 23 into the given Binary Search Tree. 23>20 (data in root). So, 23 needs to be inserted in the right sub-tree of 20.



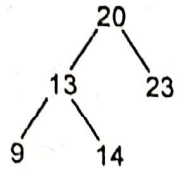
Insert 13 into the given Binary Search Tree. 13<20 (data in root). So, 13 needs to be inserted in left sub-tree of 20.



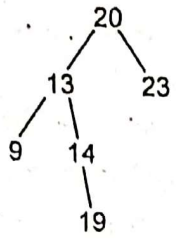
Insert 9 into the Binary Search Tree:



Inserting 14.



Inserting 19.



Deletion in Binary search tree:

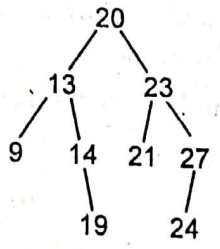
How to delete a node from binary search tree?

There are three different cases that needs to be considered for deleting a node fror binary search tree.

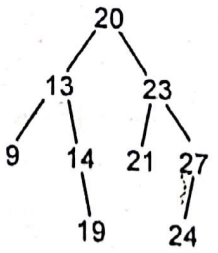
Case 1: Node with no chlidren (or) leaf node

Case 2: Node with one child

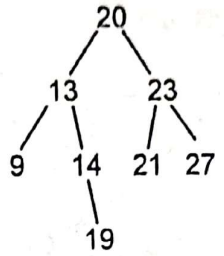
Case 3: Node with two children.



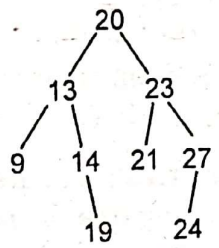
Case 1: Delete a leaf node node/with no children.



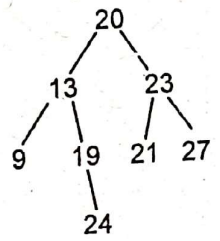
Delete 24 from the above binary search tree.



Case 2: Delete a node with one child.

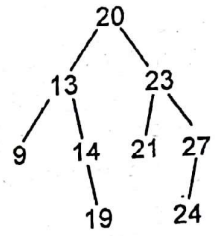


Delete 14 from above binary search tree.

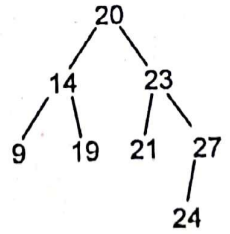


Case 3: Delete a node with two children.

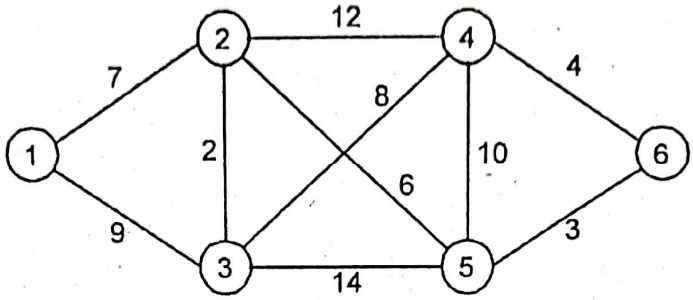
Delete a node whose right child is the smallest node in the right sub-tree. (14 is the smallest node present in the right sub-tree of 13).



Delete 13 from the above binary tree. Find the smallest in the left sub-tree of 13. So, replace 13 with 14.

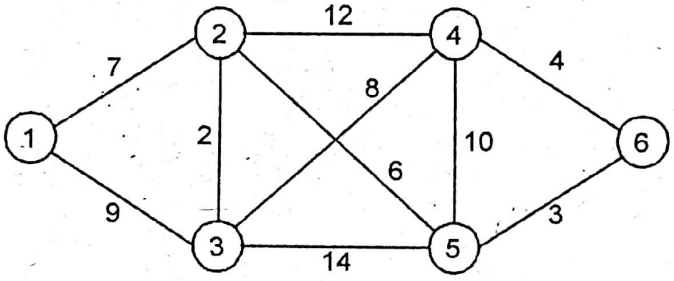


Q. 7. (a) What do you mean by minimum spanning tree? Generate minimum spanning tree for the following graph. (6.5



Ans. Given a connected weighted graph G, it is often desired to create a spanning tree T for G such that the sum of the weights of the tree edges in T is as small as possible. Such a tree is called a minimum spanning tree and represents the cheapest way of connecting all the possible nodes in G.

For create a minimum spanning tree there are 2 most famous methods are Prim and Kruskal algorithm. In any spanning tree the no. of edges will be 1 less than the no. of nodes.

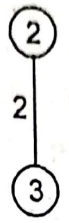


Minimum spanning tree of given above Fig. using Kruskal algorithm:

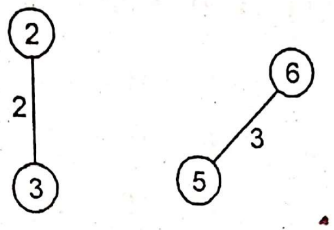
First we create a set of edges into the order by non-decreasing weight, *i.e.*

Edge	Weigth
(2, 3)	2
(5, 6)	3
(6, 4)	4
(2, 5)	6
(1, 2)	7
(3, 4)	8
(1, 3)	9
(5, 4)	10
(2, 4)	12
(3, 5)	14

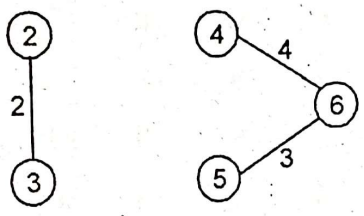
Now we create the minimum spanning tree according to given table. min. weight of Edge is Edge (2, 3) so now we connect the node



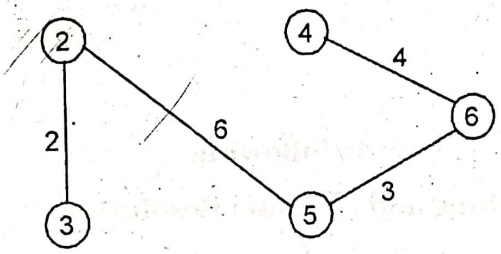
- Next edge is the (5, 6).



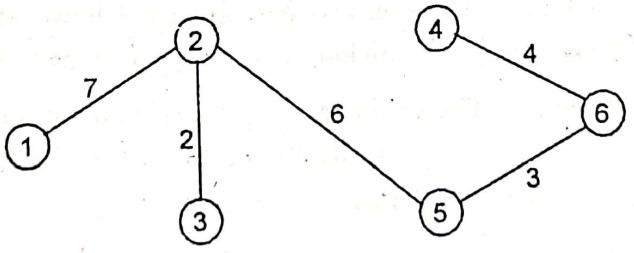
Next edge is the (6, 4).



Next edge is the (2, 5).



Next edge is the (1, 2).



This is the final minimum spanning tree.

Q. 7. (b) Explain selection sort and sort the following item using selection sort. 8, 22, 7, 9, 31, 19, 5, 13, 58, 93, 1, 45. (6)

Ans. Selection sort: The idea of selection sort is rather simple: we repeatedly find the next largest (smallest) element in the array and move it to its final position in the sorted array. This is the simple method of sorting. In this method, if n elements have to be sorted then $(n-1)$ iterations are required. The complexity of selection sort is $O(n^2)$.

Sorting of elements 8, 22, 7, 9, 31, 19, 5, 13, 58, 93, 1, 45 using

Selection sort:

Pass	LOC	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]
∞	∞	8	22	7	9	31	19	5	13	58	93	1	45
1	10	1	22	7	9	31	19	5	13	58	93	8	45
2	6	1	5	7	9	31	19	22	13	58	93	8	45
3	2	1	5	7	9	31	19	22	13	58	93	8	45
4	10	1	5	7	8	31	19	22	13	58	93	9	45
5	10	1	5	7	8	9	19	22	13	58	93	31	45
6	7	1	5	7	8	9	13	22	19	58	93	31	45
7	7	1	5	7	8	9	13	19	22	58	93	31	45
8	7	1	5	7	8	9	13	19	22	58	93	31	45
9	10	1	5	7	8	9	13	19	22	31	93	58	45
10	11	1	5	7	8	9	13	19	22	31	45	58	93
11	10	1	5	7	8	9	13	19	22	31	45	58	93

Elements are sorted.

Q. 8. Write down short note on following:- **(6.25×2=12.5)**

Q. 8. (a) Bucket hashing and Collision Resolution.

Ans. Closed hashing stores all records directly in the hash table Each record with the key value value K_R has a home position that is h (K_R). The slot computed by hash function.

When a record is inserted, the bucket to which it is mapped has available space to store the record. If the bucket does not have enough space, however it indicates an error condition called 'Bucket Overflow'. Bucket overflow occur done to following reasons.

Insufficient Buckets: The number of bucket which we denote by nb, must be choosen such that nb > nr/fr. Where nr denote total number of record.

fr denote the record that fit in the bucket.

Skewness: If the selection of a bucket, during insertion is more frequent than that of others, the bucket is said to be skewed as against being symmetrical.

• Bucket methods are good for implementing hash table stored on disk, because the bucket size can be set to the size of a disk block.

(b) Collision Resolution.

Suppose we want to add a new record R with the key K to our file F, but suppose the memory location address $H(k)$ is already occupied. This situation is called collision.

(1) Collision resolution by open addressing.

(2) Collision resolution by separate chaining.

Open Addressing: In this, all elements are stored in the hash table itself. That is each table entry contains either an element of the dynamic set or NIL.

Advantage: The advantage of open addressing is that it avoids pointers together. Instead of following pointers, we compute the sequence of slot to be examined. The extra memory freed by not storing pointers provides the hash table with a large number of slots for the same amount of memory.

Hashing with Changing: This method maintain a chain of elements which has the same hash address. We can take the hash table as an array of pointers. Size of hash table can be number of records. Here each pointer will point to one linked list and the elements which have the same hash address will be maintained in the linked list. We can maintained the linked list in sorted order and each elements of linked list will contain the whole record with key.

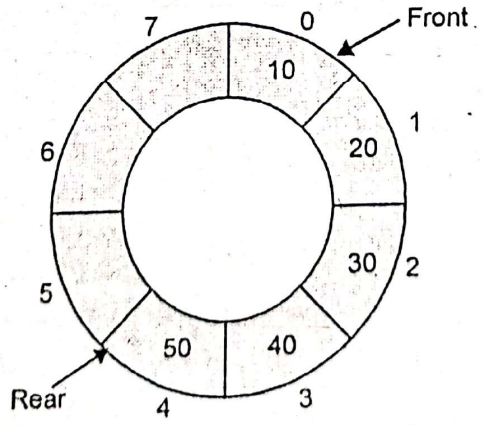
Advantage: The main advantage of this hashing is saving of memory space and perfect collision resolution.

• Here hash table contain only pointer which point to linked list-containing records.

Disadvantage: The main disadvantage is also wastage of memory space. In case where records are very small or contains only key.

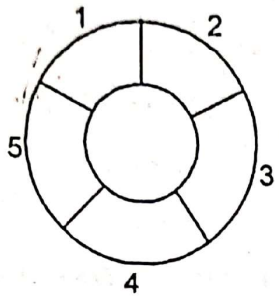
Q. 8. (b) Circular queue.

Ans. Circular Queue: A circular queue is an abstract data type that contains a collection of data which allows addition of data at the end of the queue and removal of data at the beginning of the queue. Circular queues have a fixed size. Circular queue follows FIFO principle. Queue items are added at the rear end and the items are delected at front end fo the circular queue.

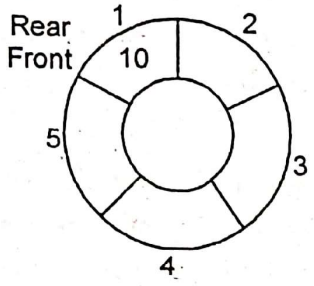


Example: Consider the following circular queue with N = 5.

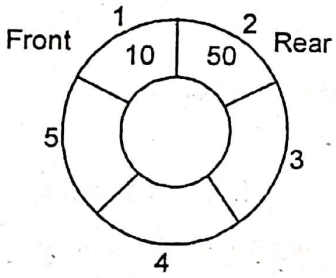
1. Initially, Rear = 0, Front = 0.



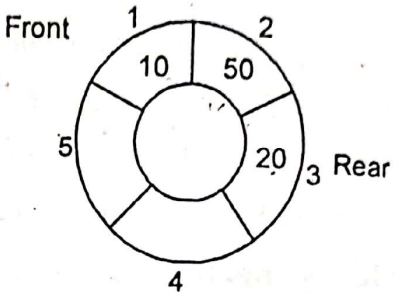
2. Insert 10, Rear = 1, Front = 1.



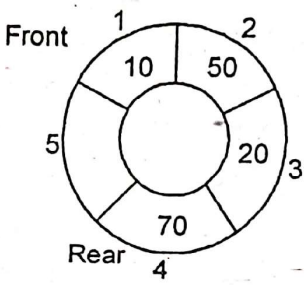
3. Insert 50, Rear = 2, Front = 1.



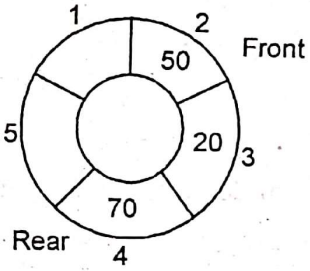
4. Insert 20, Rear = 3, Front = 0.



5. Insert 70, Rear = 4, Front = 1.



6. Delete front, Rear = 4, Front = 2.



FIRST TERM EXAMINATION [SEPT. 2017]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 1½ hrs. **M.M. : 30**

Note: *Q.1 is compulsory. Attmept any two questions from the remaining:*

Q.1. Attempt all parts:

Q.1. (*a*) Convert the given infix into post-fix expression. Demonstrate the steps of conversion. **(2)**

a + b* c– (d/e^f)*g)*h

Ans. Infix into Postfix expression of a + (b* c – (d/e^f)*g)*h:

Character Scanned	Stack	Postfix Notation
a		a
+	+	a
(	+(	a
b	+(	ab
*	+(*	ab
c	+(*	abc
–	+(–	abc*
(	+(–(	abc*
d	+(–(	abc*d
/	+(–(/	abc*d
e	+(–(/	abc*de
^	+(–(/^	abc*de
f	+(–(/^	abc*def
)	+(–	abc*def^/
*	+(–*	abc*def^/
g	+(–*	abc*def^/g
)	+	abc*def^/g*-
*	+*	abc*def^/g*-
h	+*	abc*def^g*-h
)		abc*def^/g*-h*+

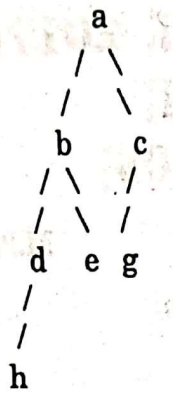
Q.1. (*b*) What is the maximum height of any AVL trees with 7 nodes? Assuming height is 0 for a tree with single node. **(2)**

Ans. The maximum height of any AVL-tree with 7 nodes is 3.

AVL trees are binary trees with the following restrictions.

1) the height difference of the children is at most 1.

2) both children are AVL trees



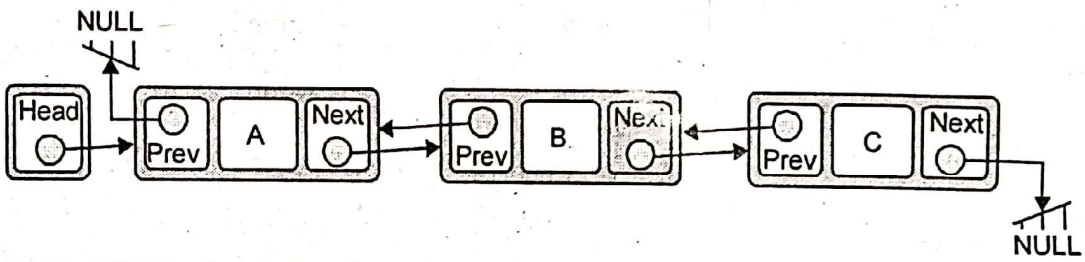
Q.1. (*c*) What is a doubly linked list? (2

Ans. Doubly Linked List :

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- **Link** - Each link of a linked list can store a data called an element.
- **Next** - Each link of a linked list contains a link to the next link called Next.
- **Prev** - Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** - A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation



Q.1. (*d*) What pointer type is used to implement a heterogneous list in C? Explain in brief? (2

Ans. The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

Q.1. (*e*) Explain the following terms used in tree: (2

(i) Terminal Node **(ii) Siblings**

(iii) Degree of Node **(iv) Height of tree**

Ans. (i) Terminal Node : In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

(ii) Siblings : In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.

(iii) Degree of Node : In a tree data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has.

(iv) Height of tree : In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a

tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'**.

Q.2. Attempt all parts:

Q.2. (*a*) Calculate the address of element at X[4, 3] in a 2D array X[1.5][1.4] stored in a row major order. Assume the base address to be 1000 and each element requires 4 words of storage. **(5)**

Ans. Row Major Order

LOC [A[i] [j]] = Base (A) + w[n (i – Lower bound for row index) + (j – lower bound for column index)]

Base (X) = 1000, w = 4, n = 4

So the address of X [4.3] = 1000 + 4 [4 (4 – 1) + (3 – 1)]

= 1000 + 4[4 × 3 + 2] = 1000 + 4 × 14 = 1056.

Q.2. (*b*) Write an algorithm to delete the last node of the linked list. **(5)**

Ans. Step 1: If linked list is empty (START == NULL)

Write "Sorry Buddy there is no node

Step 2: If linked list have only one node (START -> NEXT== NULL)

```
free(START)
START = NULL
```

Step 3: If the list have more than one node

```
PTR = START
while(PTR -> NEXT != NULL)
{
PREPTR = PTR
PTR = PTR -> NEXT
}
PREPTR -> NEXT = NULL
free( PTR )
```

Step 4 : EXIT

Q.3. Attempt all parts: **(3 + 7=10)**

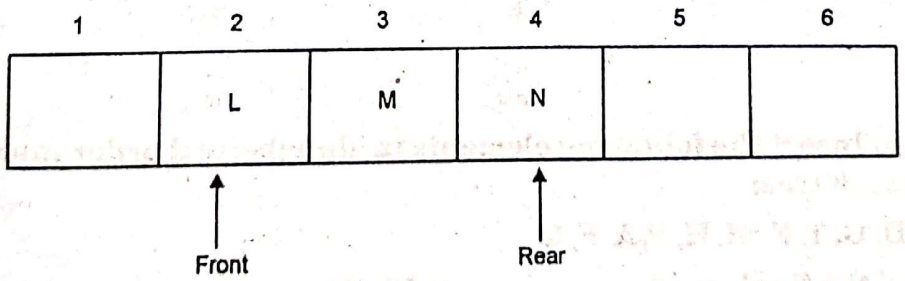
Q.3. (*a*) Consider the following circular queue with a storage size 6.

FRONT = 2, REAR = 4, QUEUE: _, L,M,N _,_

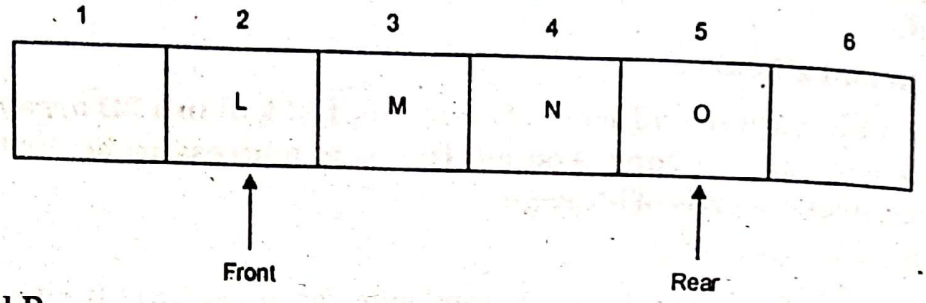
Describe the queue as following operations takes place:

1. Add O 2. Add P 3. Delete two elements 4. Add Q,R,S 5. Delete one. **(3)**

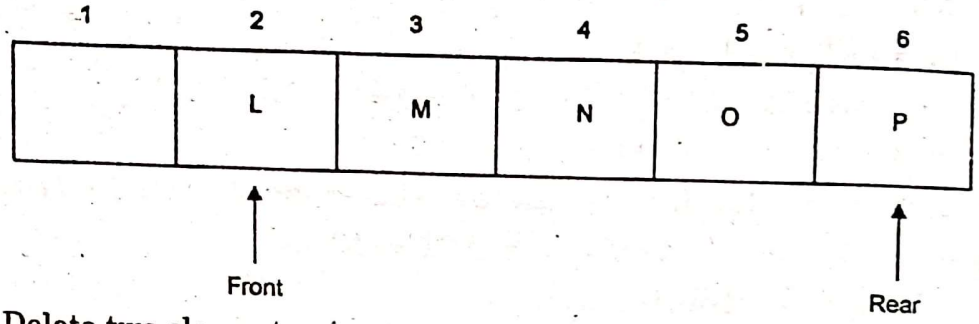
Ans. Initially



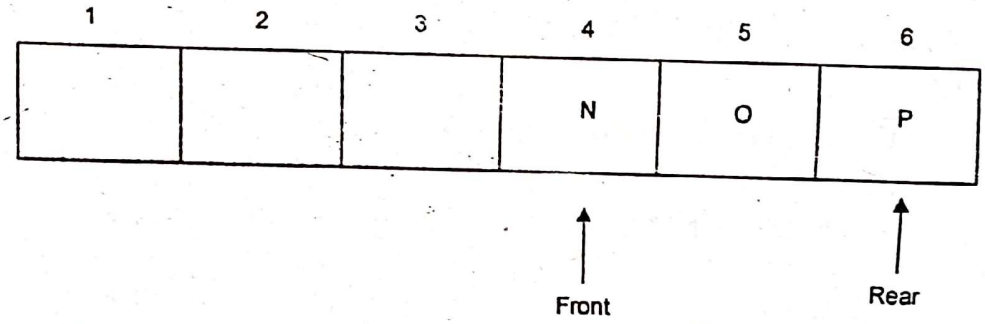
(i) Add 0



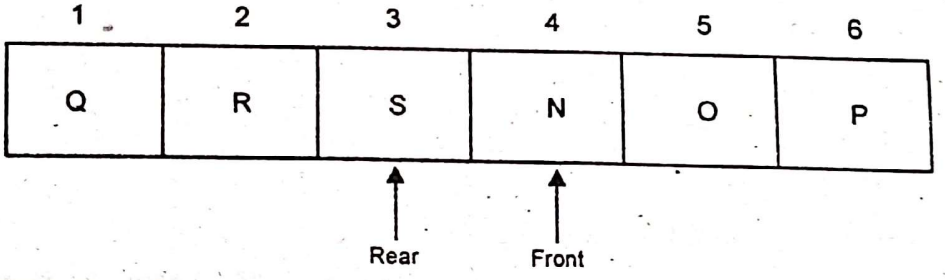
(ii) Add P



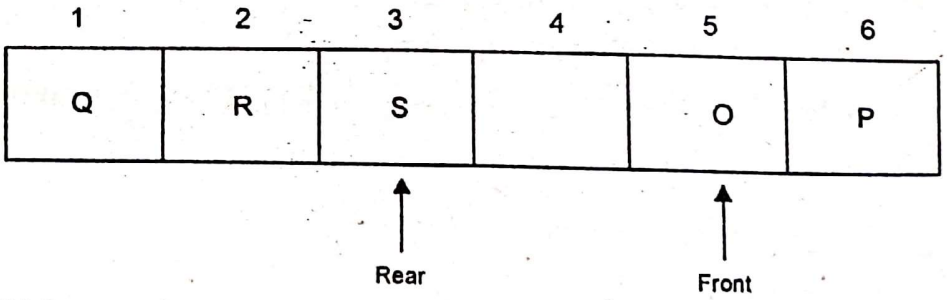
(iii) Delete two elements



(iv) Add Q, R, S



(v) Delete one letter



Q.3. (*b*) Insert the following elements in alphabetical order into an empty binary search tree: **(7)**

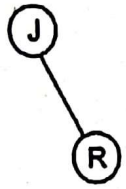
J, R, D, G, T, E, M, H, P, A, F, Q

1. Find the final tree T **2. Find the post-order traversal of T.**

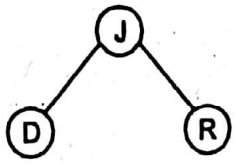
Send Spotify Playlist to Anjali today

Ans. Insert J

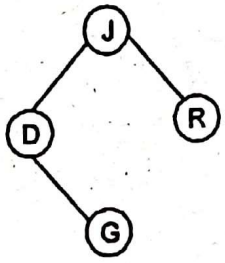
Insert R



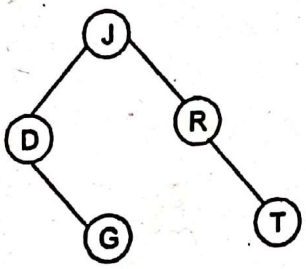
Insert D



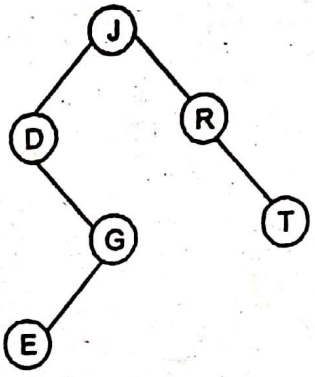
Insert G



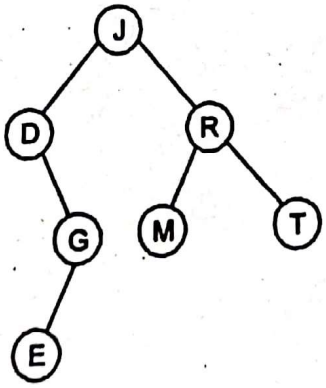
Insert T



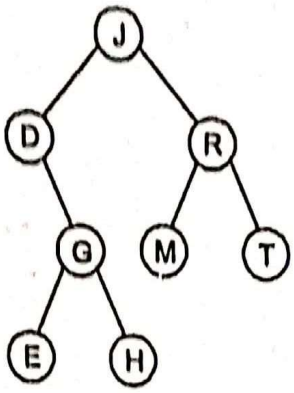
Insert E



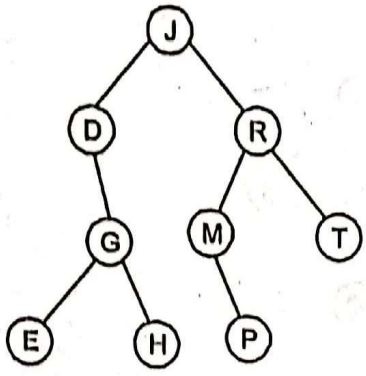
Insert M



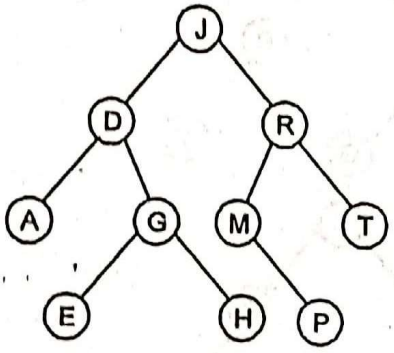
Insert H



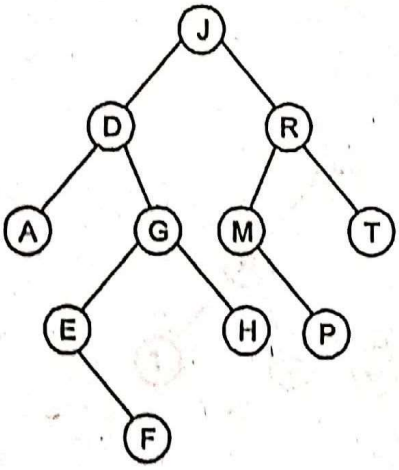
Insert P



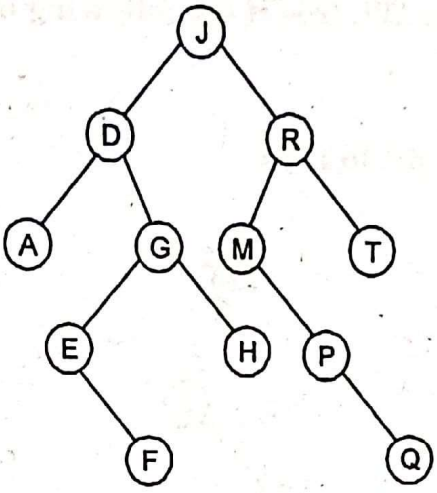
Insert A



Insert F



Insert Q



This is final tree T.

(ii) Post order traversal of tree T is :

$\boxed{\text{AFEHGDQPMTRJ}}$

Q.4. Attempt all parts:

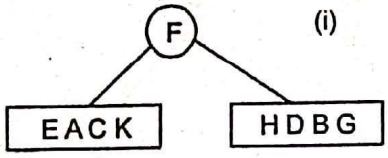
Q.4. (*a*) A binary tree has 9 nodes. The inorder and preorder traversals are of T yield the following sequences of nodes: **(5)**

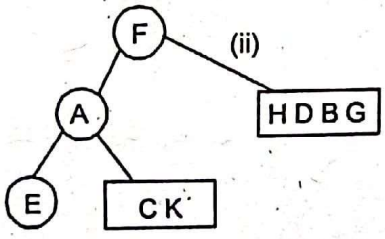
INORDER: E A C K F H D B G

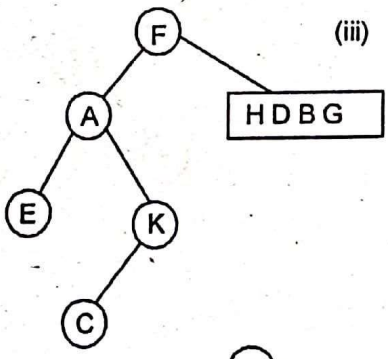
PREORDER: F A E K C D H G B. Draw the tree T.

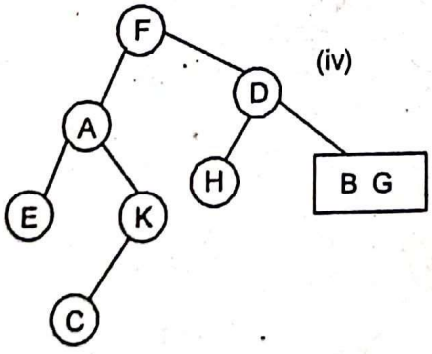
Ans. INORDER: E A C K F H D B G

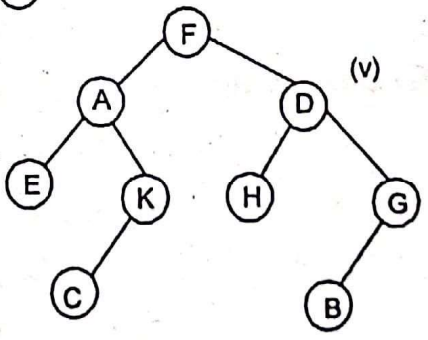
PREORDER: F A E K C D H G B











This is final tree T.

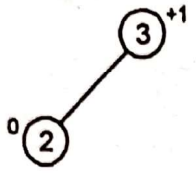
Q.4. (*b*) Construct the AVL tree if the following elements are inserted in order:

3,2,1,4,5,6,16,15,14 **(5)**

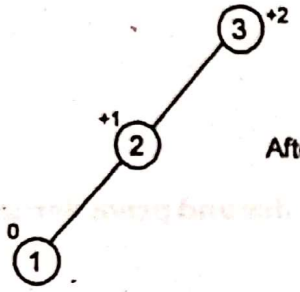
Ans. AVL tree of 3,2,1,4,5,6,16,15,14

Insert 3

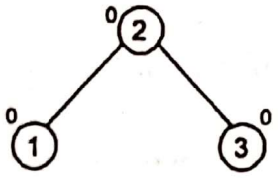
Insert 2



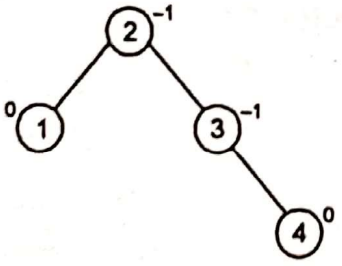
Insert 1



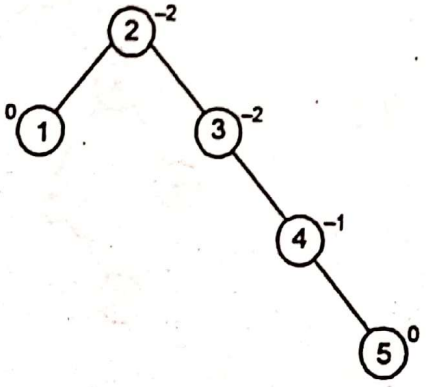
After Rebalancing



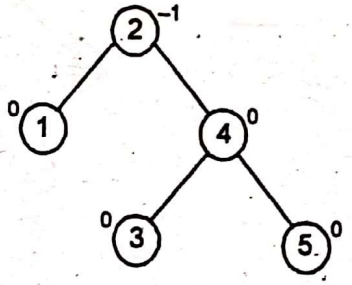
Insert 4



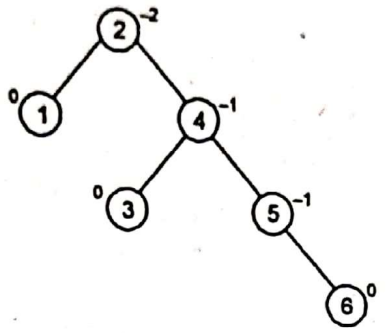
Insert 5



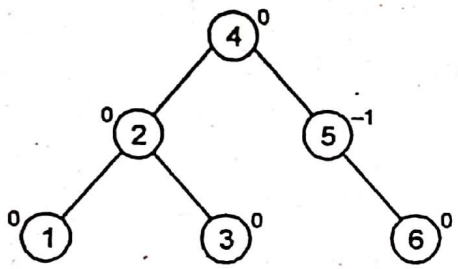
After Rebalancing



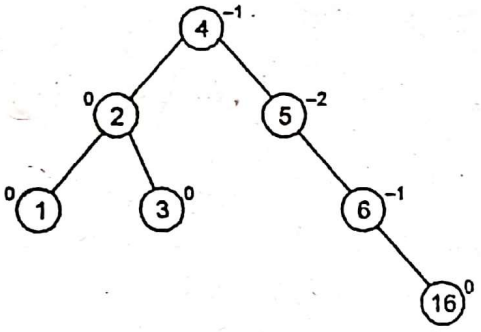
Insert 6



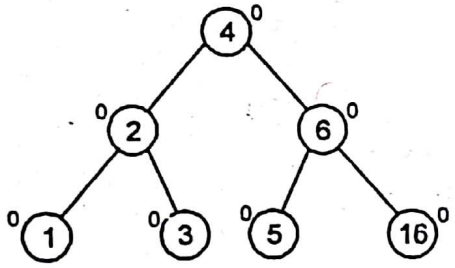
After Rebalancing



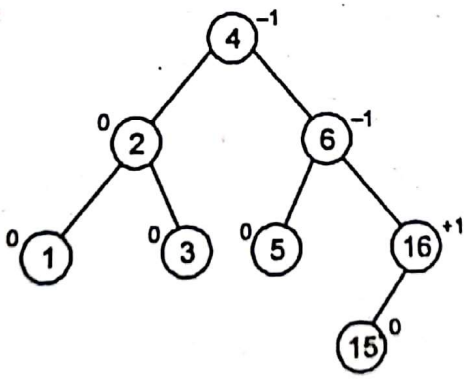
Insert 16



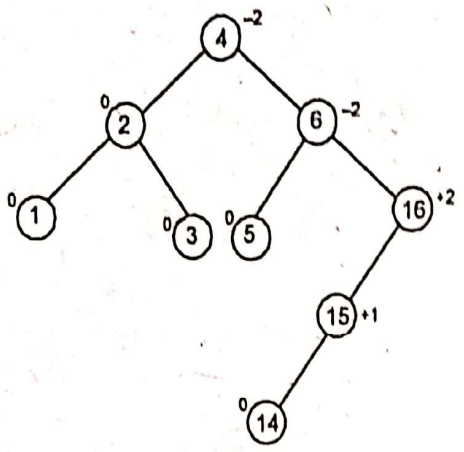
After Rebalancing



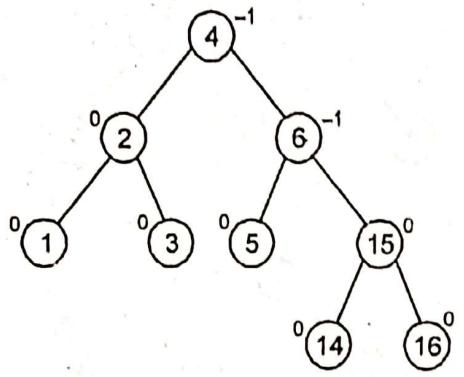
Insert 15



Insert 14



After Rebalancing the final tree T is:



END TERM EXAMINATION [DEC. 2017]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions including Q.No. 1. Which is compulsory. Select one question from each unit.*

Q.1. (*a*) What is an Abstract Data Type (ADT)? **(2.5)**

Ans. Abstract Data Types

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction.

The user of data type need not know that data type is implemented, for example, we have been using int, float, char data types only with the knowledge with values that can take and operations that can be performed on them without any idea of how these types are implemented. So a user only needs to know what a data type can do but not how it will do it.

We can think of ADT as a black box which hides the inner structure and design of the data type. Now we'll define three ADTs namely List ADT, Stack ADT, Queue ADT.

Q.1. (*b*) What is a self-referential structure? **(2.5)**

Ans. A self referential structure is used to create data structures like linked lists, stacks, etc. Following is an example of this kind of structure:

```
struct struct_name
{
datatype datatypename;
struct_name * pointer_name;
};
```

A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type. For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type. For example,

```
typedef struct listnode {
void *data;
struct listnode *next;
} linked_list;
```

In the above example, the listnode is a self-referential structure – because the *next is of the type struct listnode.

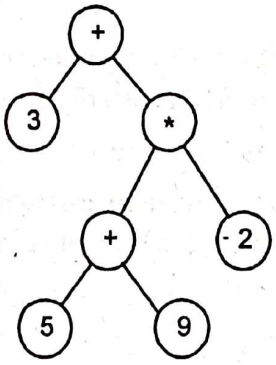
Q.1. (c) Write at least two differences in between a singly linked list and doubly linked list. **(2.5)**

Ans.

Singly linked list	Doubly linked list
A singly linked list is a linked list where the node contains some data and a pointer to the next node in the list	A doubly linked list is complex type of linked list where the node contains some data and a pointer to the next as well as the previous node in the list
It allows traversal only in one way	It allows a two way traversal
It uses less memory per node (single pointer)	It uses more memory per node (two pointers)
If we need to save memory and searching is not required, we use singly linked list	If we need better performance while searching and memory is not a limitation, we go for doubly linked list
Singly linked list can mostly be used for stacks	They can be used to implement stacks, heaps, binary trees.

Q.1. (d) Define expression tree and its usage. **(2.5)**

Ans. Expression Tree: Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be:



Inorder traversal of expression tree produces infix version of given postfix expression (same with preorder traversal it gives prefix expression)

Construction of Expression Tree:

Now For constructing expression tree we use a stack.

(1) If character is operand push that into stack

(2) If character is operator pop two values from stack make them its child and push current node again.

At the end only element of stack will be root of expression tree.

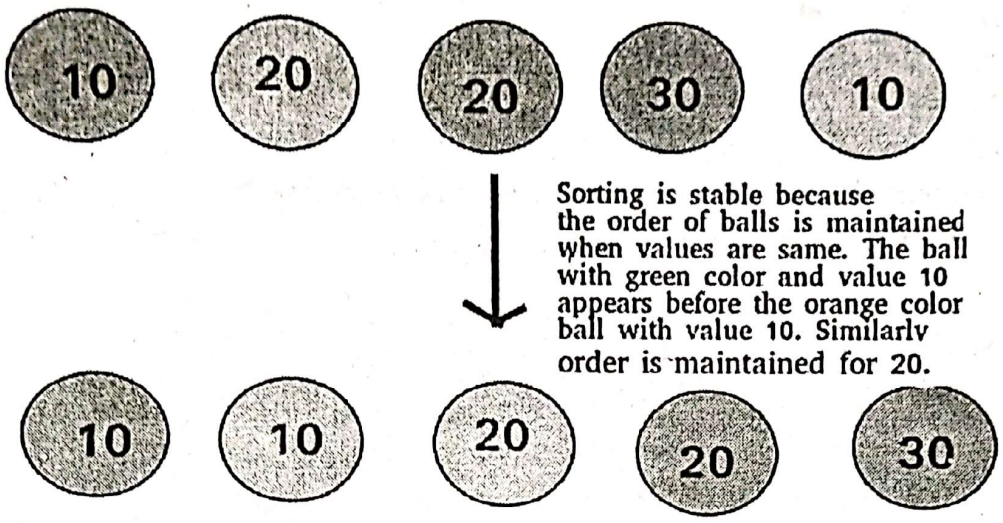
Q.1. (e) When a sorting procedure is considered to be stable? **(2.5)**

Ans. If two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array then the algorithm used for sorting is a stable sort algorithm.

Let's say there are some documents with IDs and you want to sort them according to their document IDs. The IDs needn't be distinct. Two documents, doc1 and doc2 might

have the same ID. If in the unsorted array doc1 comes before doc2 then after sorting also doc1 has to come before doc2. If the algorithm you use for sorting has this property then it is stable.

Stability means that equivalent elements retain their relative positions, after sorting.



Mergesort, bubble sort, insertion sort can be easily made stable just by taking care of this property while comparing itself, i.e, swap only if one is strictly less than the other, otherwise let them be as they are.

Radix sort can be performed only because counting sort is stable, and this is a good example as to why a sorting algorithm should be stable.

Q.1. (*f*) Differentiate in between B tree and B+ tree. (2.5)

Ans. B+ Trees are different from B Trees with following two properties:

1.B+ trees don't store data pointer in interior nodes, they are ONLY stored in leaf nodes. This is not optional as in B-Tree. This means that interior nodes can fit more keys on block of memory.

2.The leaf nodes of B+ trees are linked, so doing a linear scan of all keys will requires just one pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree. This property can be utilized for efficient search as well since data is stored only in leafs.

Q.1. (*g*) Show that the inorder traversal of a BST always produces the sorted output. (2.5)

Ans. In BST all the values in the left subtree is smaller than the parent node, which is smaller than all the values in the right subtree.

In order traversal will at first visit all elements of the left subtree of the root. Then it will print the root element, and lastly it will print all values from the right subtree. Since left and right subtrees have smaller amount of nodes than the whole tree, by induction we know that the traversal will print the numbers in increasing order.

Eventually we get sorted left subtree values, then root value, and then sorted right subtree values. After applying BST definition here, we prove that this list is also sorted, thus proving the algorithm is correct.

Q.1. (*h*) A B-tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place? (2.5)

Ans. Maximum number of node splitting operations that may take place is 5.

Explanation:

Insertion of 3 keys

10 20 30

Insertion of 4th key (1st split)

```
   30
  / \
10*20  40
```

Insertion of 5th key no split

To maximize splits, let us insert a value in a node that has key in access. Let us insert 5

```
    30
  /   \
5*10*20  40
```

Insertion of 6th key (2nd Split)

To maximize splits, let us insert a value in a node that has key in access. Let us insert 6

```
    8*30
  / | \
5  10*20  40
```

Insertion of 7th key

To maximize splits, let us insert a value in a node that has key in access. Let us insert 15

```
    8*30
  /  |  \
5  10*15*20  40
```

Insertion of 8th key (3rd Split)

To maximize splits, let us insert a value in a node that has key in access. Let us insert 12

```
    8*12*30
  / / \ \
5  10 15*20  40
```

Insertion of 9th key

To maximize splits, let us insert a value in a node that has key in access. Let us insert 17

```
    8*12*30
  / / \ \
5  10 15*17*20 40
```

Insertion of 10th key (4th and 5th Splits)

To maximize splits, let us insert a value in a node that has key in access. Let us insert 13

```
      12
    /   \
   8     15*30
  / \   / | \
5  10 13 17*20  40
```

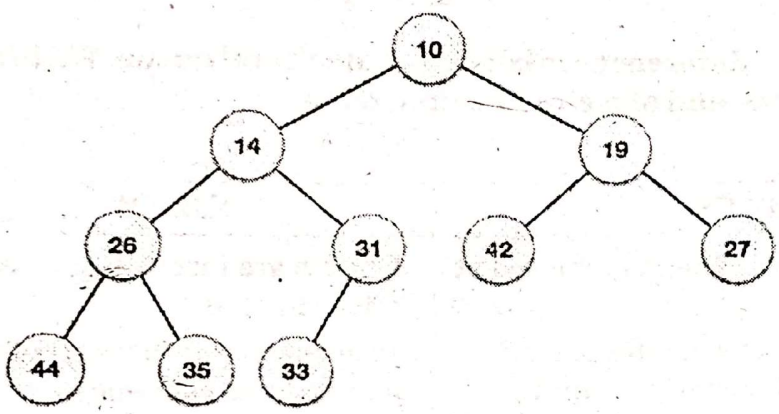
Q.1. (*i*) What is the heap property? Illustrate the insertion of an element in a heap through an example. **(2.5)**

Ans. Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If α has child node β then - key(α) ≥ key(β)

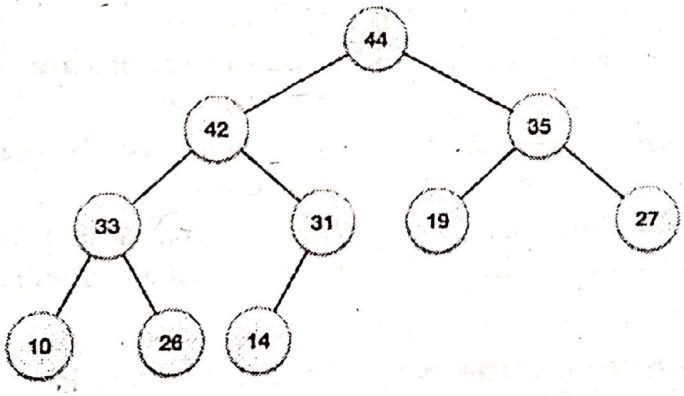
As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types -

For Input → 35 33 42 10 14 19 27 44 26 31

Min-Heap - Where the value of the root node is less than or equal to either of its children.



Max-Heap - Where the value of the root node is greater than or equal to either of its children.



Both trees are constructed using the same input and order of arrival.

Q.1. (*j*) Define perfect binary tree. How many different binary trees are possible with four nodes? **(2.5)**

Ans. Perfect Binary Tree: A Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at same level.

Following are examples of Perfect Binary Trees.

```
        18
      /    \
    15      30
   / \     / \
  40  50  100  40
```

```
        18
      /    \
    15      30
```

A Perfect Binary Tree of height h (where height is number of nodes on path from root to leaf) has $2^h - 1$ node.

For a binary tree with n nodes, the number of labelled binary trees possible is given by:-

C(n)= (2n)! / (n + 1)!

= (2 * 4)! / (4 + 1)!

= 8! / 5!

= 336

336 Binary Trees are possible with 4 nodes.

UNIT–I

Q.2. List major differences in between a stack and queue. Write an algorithm to add a node at the end of a circular linked list. **(12.5)**

Ans.

#	STACK	QUEUE
1.	Objects are inserted and removed at the same end.	Objects are inserted and removed from different ends.
2.	In stacks only one pointer is used. It points to the top of the stack.	In queues, two different pointers are used for front and rear ends.
3.	In stacks, the last inserted object is first to come out.	In queues, the object inserted first is first deleted.
4.	Stacks follow Last In First Out (LIFO) order.	Queues following First In First Out (FIFO) order.
5.	Stack operations are called push and pop.	Queue operations are called enqueue and dequeue.
6.	Stacks are visualized as vertical collections.	Queues are visualized as horizontal collections.
7.	Collection of dinner plates at a wedding reception is an example of stack.	People standing in a file to board a bus is an example of queue.

Algorithm for Insertion at the last of Circular linked list

```
1.node* AddEnd(node* tail, int num)
2. {
3.node *temp = (node*)malloc(sizeof(node));
4. temp->data = num;
5. temp->next = temp;
6. if (tail == NULL)
7.  return temp;
8. temp->next = tail->next;
9. tail->next = temp;
10. return temp;
11. }
```

Q.3. How to convert infix into a postfix expression. Write a program to carry out this conversion and illustrate the conversion of AB * C–D + E/F into a postfix expression.

Ans. Procedure for Postfix Conversion **(12.5)**

1. Scan the Infix string from left to right.
2. Initialize an empty stack.
3. If the scanned character is an operand, add it to the Postfix string.
4. If the scanned character is an operator and if the stack is empty push the character stack.
5. If the scanned character is an Operator and the stack is not empty, compare the precedence of the character with the element on top of the stack.
6. If top Stack has higher precedence over the scanned character pop the stack else push the scanned character to stack. Repeat this step until the stack is not empty and Stack has precedence over the character.
7. Repeat 4 and 5 steps till all the characters are scanned.
8. After all characters are scanned, we have to add any character that the stack may have to the Postfix string.
9. If stack is not empty add top Stack to Postfix string and Pop the stack.
10. Repeat this step as long as stack is not empty.

Infix into Postfix expression of a+ b*c- d + e / f :

Character Scanned	Stack	Postfix Notation
a		a
+	+	a
b	+	a b
*	+ *	a b
c	+ *	a b c
–	–	a b c * +
d	–	a b c * + d
+	+	a b c * + d –
e	+	a b c * + d – e
/	+/	a b c * + d – e
f	+/	a b c * + d – e f
		a b c * + d – e f / +

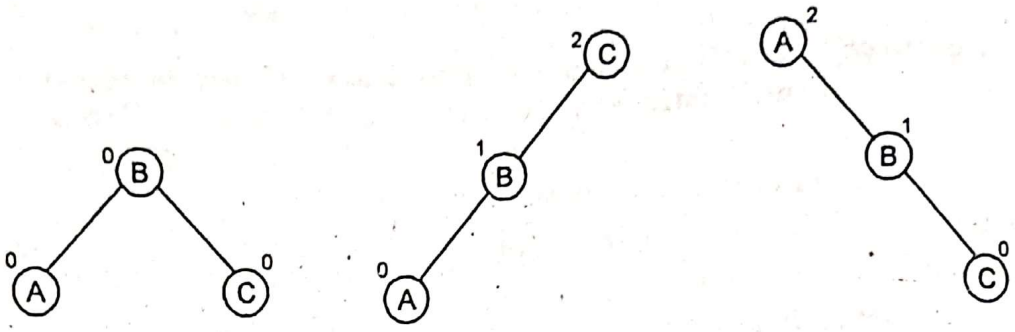
UNIT–II

Q.4. Define an AVL tree. What are the different AVL tree rotations? Construct an AVL tree for the following list of numbers:

10 5 8 12 18 22 1 4 6 30 **(12.5)**

Ans. AVL trees are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor.**

Here we see that the first tree is balanced and the next two trees are not balanced -



Balanced Not balanced Not balanced

In the second tree, the left subtree of **C** has height 2 and the right subtree has heigh 0, so the difference is 2. In the third tree, the right subtree of **A** has height 2 and the lef is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balanc factor) to be only 1.

BalanceFactor = height(left-sutree) – height(right-sutree)

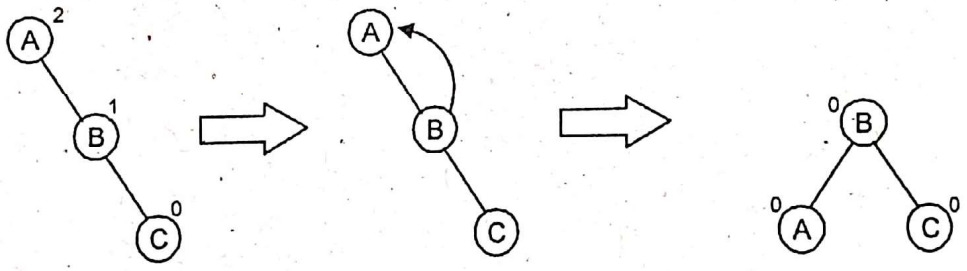
If the difference in the height of left and right sub-trees is more than 1, the tree i balanced using some rotation techniques.

AVL Rotations

To balance itself, an AVL tree may perform the following four kinds of rotations

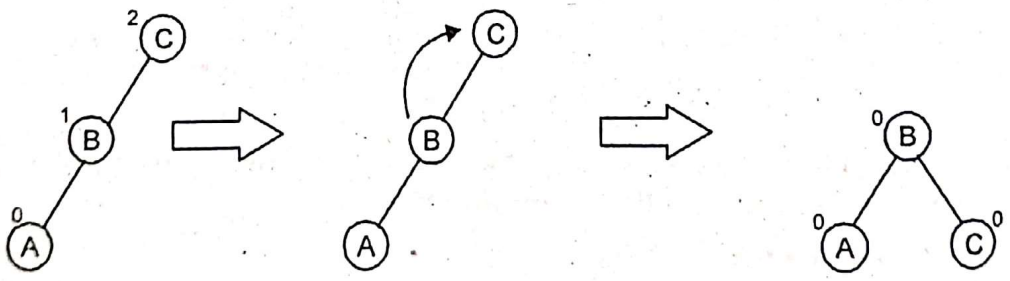
- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

Left Rotation: If a tree becomes unbalanced, when a node is inserted into the righ subtree of the right subtree, then we perform a single left rotation –



Right unbalanced tree Left Rotation Balanced

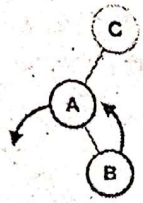
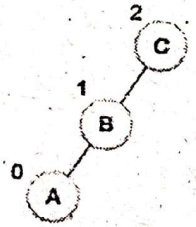
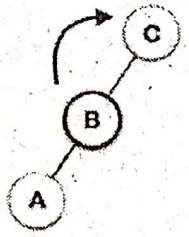
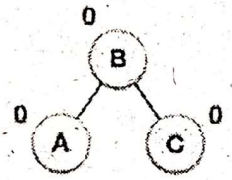
Right Rotation: AVL tree may become unbalanced, if a node is inserted in the le subtree of the left subtree. The tree then needs a right rotation.



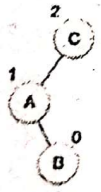
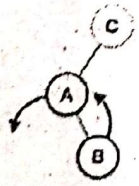
Left unbalanced tree Right Rotation Balanced Tree

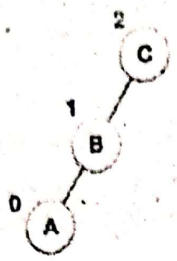
Left-Right Rotation: Double rotations are slightly complex version of alread explained versions of rotations. To understand them better, we should take note of eac

action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

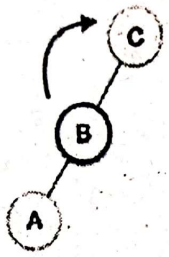
State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.
	The tree is now balanced.

Right-Left Rotation: The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

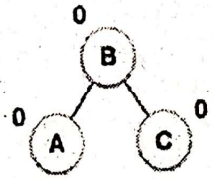
State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.



Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.



A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.

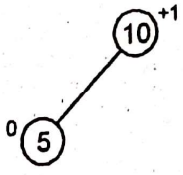


The tree is now balanced.

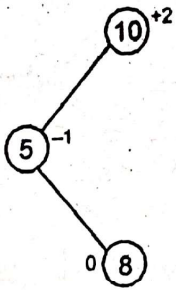
AVL tree of 10, 5, 8, 12, 18, 22, 1, 4, 6, 30

Insert 10

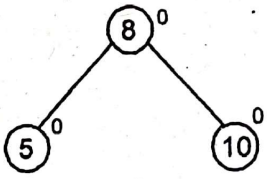
Insert 5



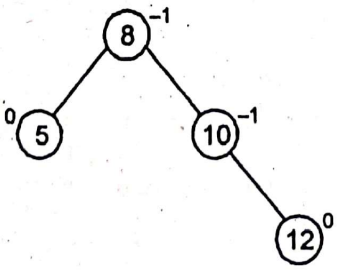
Insert 8



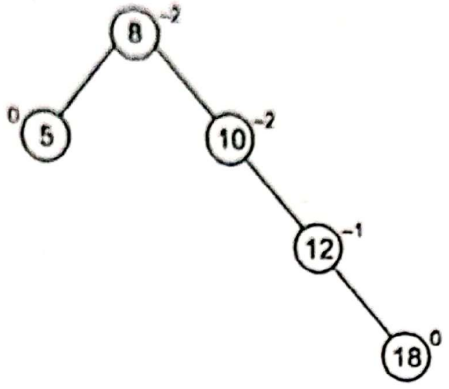
After Rebalancing



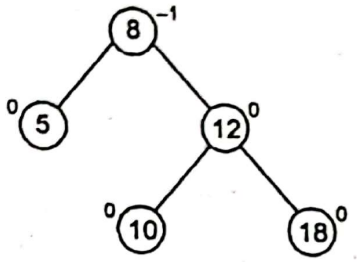
Insert 12



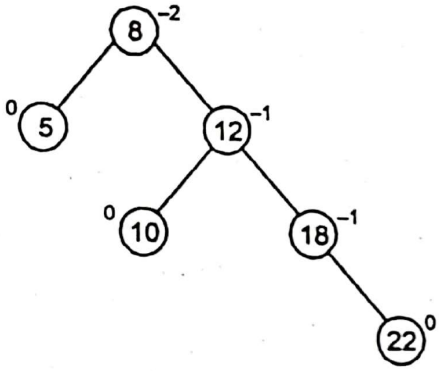
Insert 18



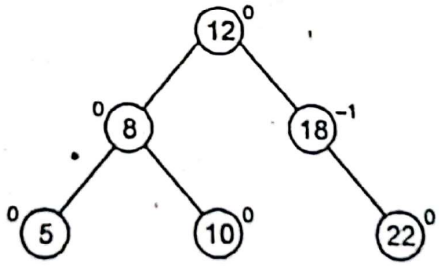
After Rebalancing



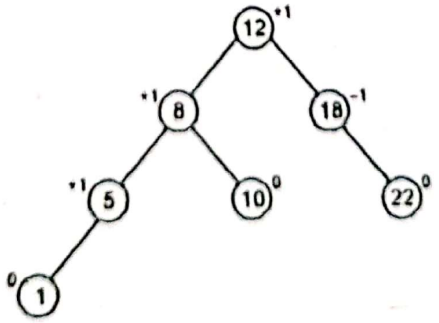
Insert 22



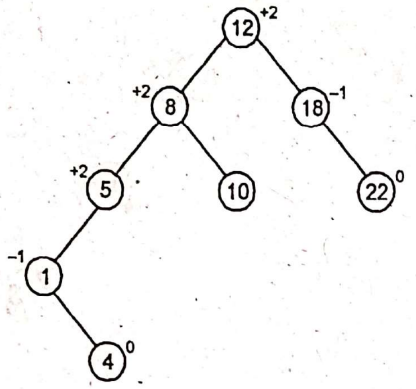
After Rebalancing



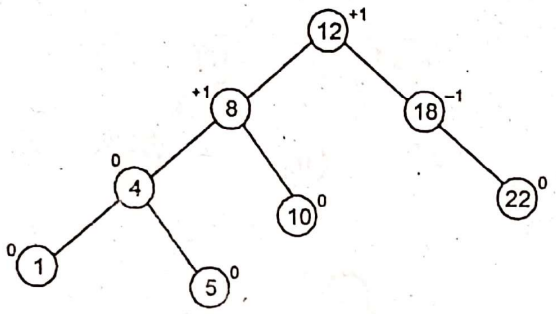
Insert 1



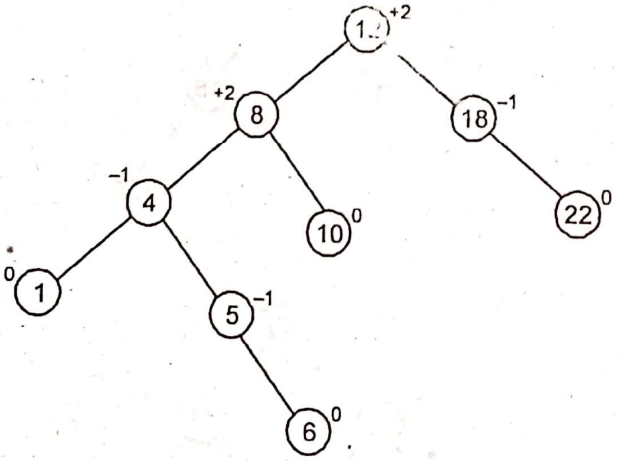
Insert 4



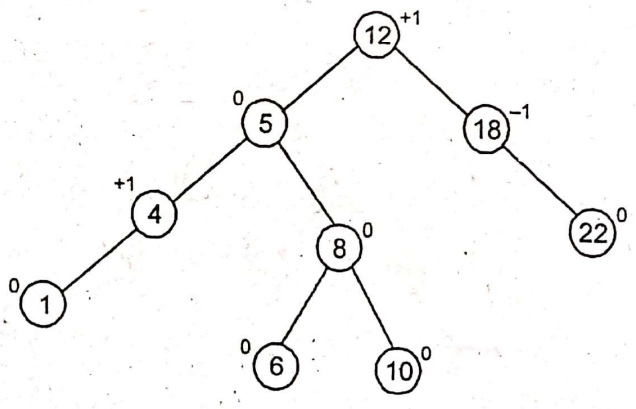
After Rebalancing



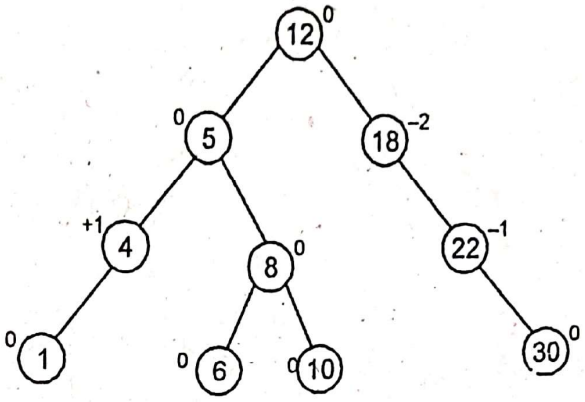
Insert 6



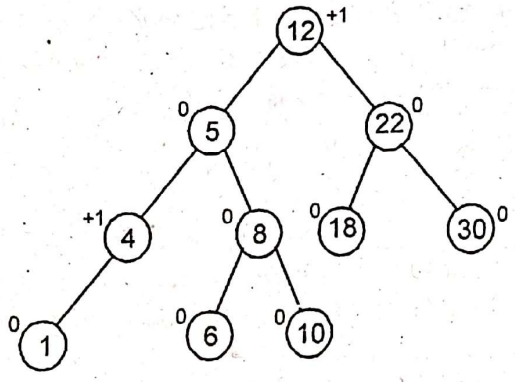
After Rebalancing



Insert 30



After Rebalancing Final tree T is:



Q.5. Define inorder and postorder travesals of a tree using suitable example. Construct a Binary tree using the following data:

INORDER: BGDKHAEICJF

POSTORDER: GKHDBEIJFCA **(12.5)**

Ans. Traversal is a process to visit all the nodes of a tree and may print their values ıo. Because, all nodes are connected via edges (links) we always start from the root ıead) node. That is, we cannot randomly access a node in a tree. There are three ways hich we use to traverse a tree -

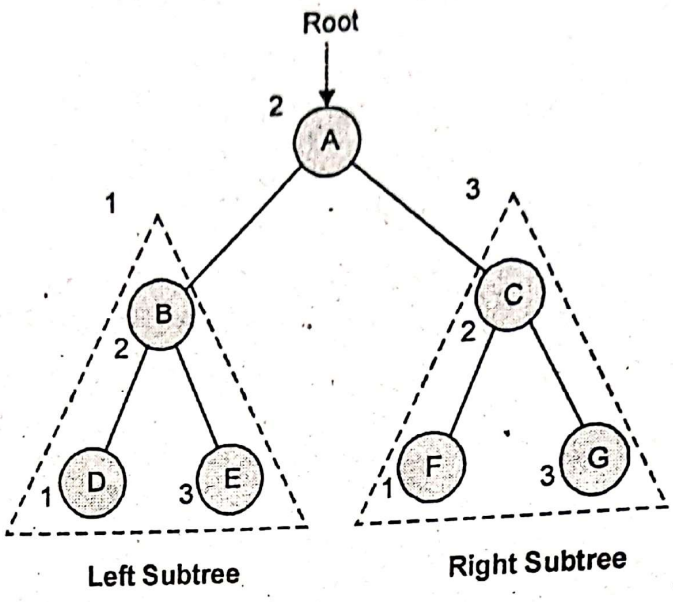
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to rint all the values it contains.

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the ight sub-tree. We should always remember that every node may represent a subtree tself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.

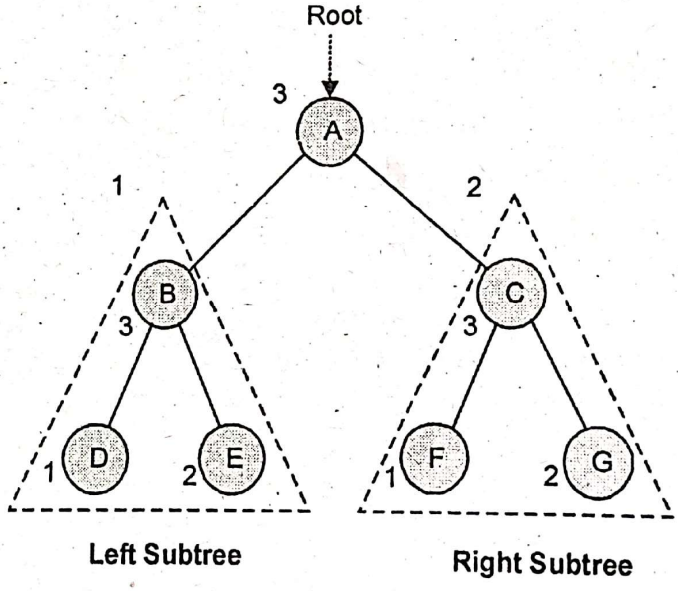


We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** also traversed in-order. The process goes on until all the nodes are visited. The output inorder traversal of this tree will be -

$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

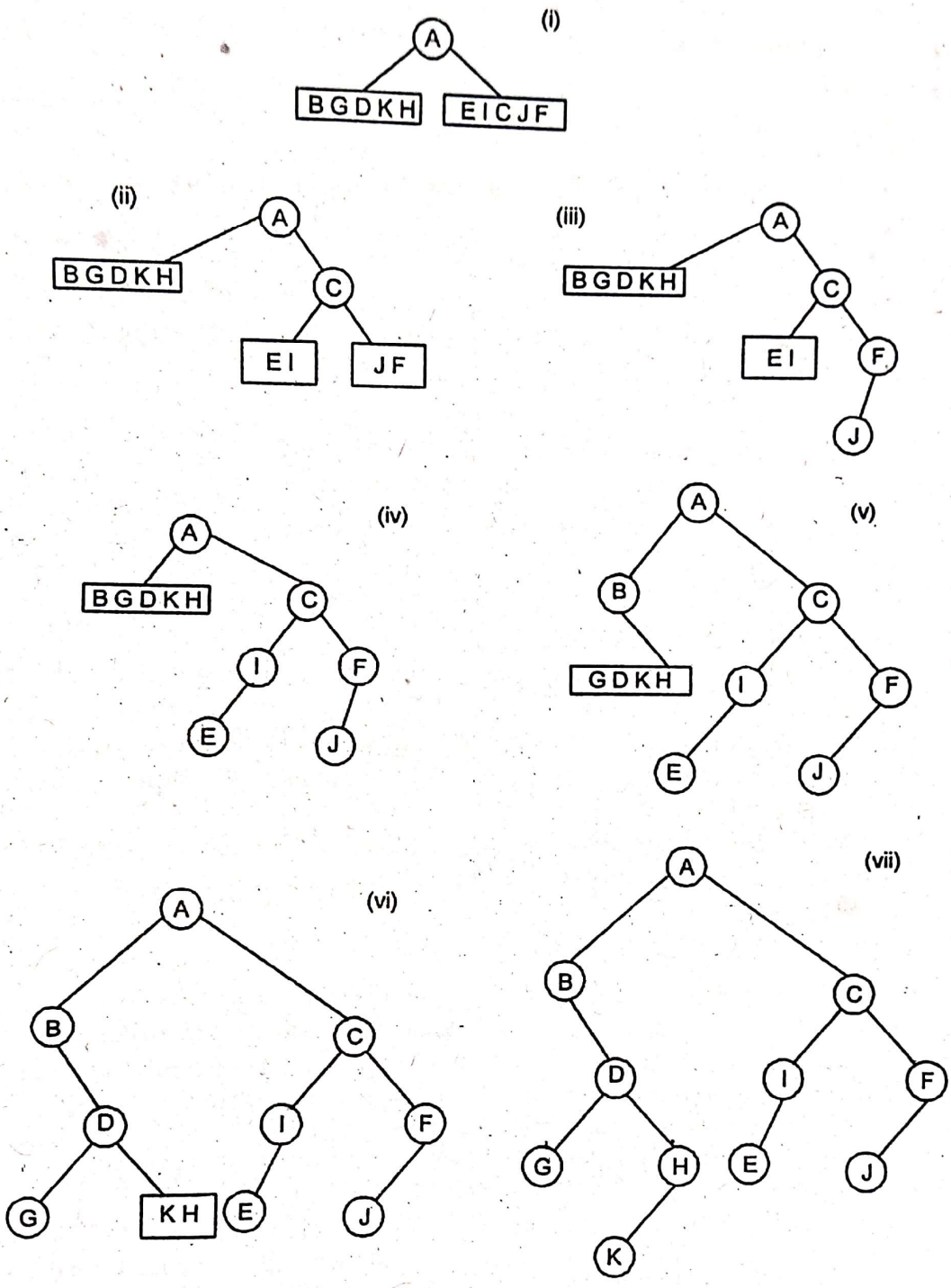


We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be -

$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

INORDER: B G D K H A E I C J F

POSTORDER: G K H D B E I J F C A



This is final tree T.

UNIT–III

Q.6. Define B tree structure properties. Consider a B+ tree in which the maximum number of keys in a node is 7. What is the minimum number of keys in any non-root node? Explain insertion and delections process in a B tree. **(12.5)**

Ans. B-Tree can be defined as follows...

B-Tree is a self-balanced search tree with multiple keys in every node and more than two children for every node.

Here, number of keys in a node and number of children for a node is depend on the order of the B-Tree. Every B-Tree has order.

B-Tree of Order m has the following properties...

Property #1 - All the leaf nodes must be at same level.

Property #2 - All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys.

Property #3 - All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.

Property #4 - If the root node is a non leaf node, then it must have at least 2 children.

Property #5 - A non leaf node with n-1 keys must have n number of children.

Property #6 - All the key values within a node must be in Ascending Order.

In B+ Tree the maximum no. of keys in a node is 7.

That means the order of B+ tree is = 7+1 = 8.

Minimum children in a non root node = (Order/ 2) = 8 / 2 = 4.

Keys = Minimum children in a non root node – 1 = 4 – 1 = 3.

Insertion and Deletion Process in a B tree:

Insertion algorithm

(1) Initialize x as root.

(2) While x is not leaf, do following

(a) Find the child of x that is going to to be traversed next. Let the child be y.

(b) If y is not full, change x to point to y.

(c) If y is full, split it and change x to point to one of the two parts of y. If k is smaller than mid key in y, thenset x as first part of y. Else second part of y. When we split y, we move a key from y to its parent x.

(3) The loop in step 2 stops when x is leaf. x must have space for 1 extra key as we have been splitting allnodes in advance. So simply insert k to x.

Deletion algorithm

Descend to the leaf where the key exists.

1.Remove the required key and associated reference from the node.

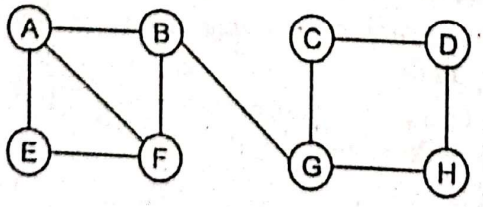
2.If the node still has enough keys and references to satisfy the invariants, stop.

3.If the node has too few keys to satisfy the invariants, but its next oldest or next youngest sibling at the same level has more than necessary, distribute the keys between this node and the neighbor. Repair the keys in the level above to represent that these nodes now have a different "split point" between them; this involves simply changing a key in the levels above, without deletion or insertion.

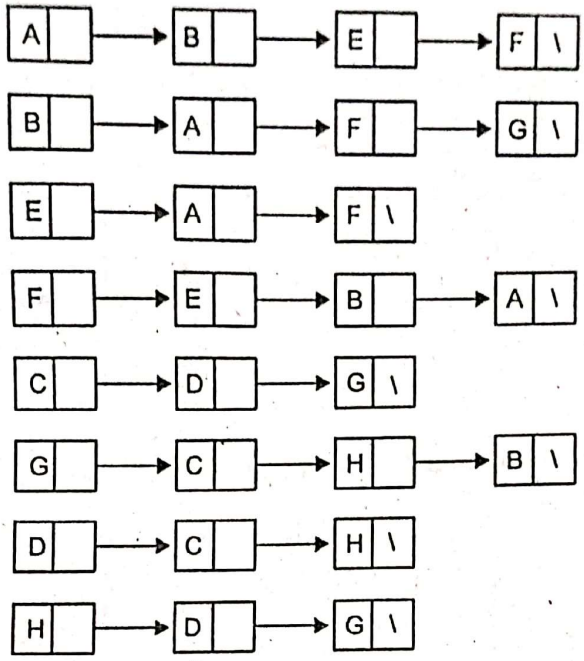
4. If the node has too few keys to satisfy the invariant, and the next oldest or next youngest sibling is at the minimum for the invariant, then merge the node with its sibling; if the node is a non-leaf, we will need to incorporate the "split key" from the parent into our merging. In either case, we will need to repeat the removal algorithm on the parent node to remove the "split key" that previously separated these merged nodes — unless the parent is the root and we are removing the final key from the root, in which case the merged node becomes the new root (and the tree has become one level shorter than before).

Q.7. How to represent a graph using adjacency linked list? Execute the Breadth first search (BFS) algorithm on the following graph starting from the vertex "A".

(12.5)



Ans. List representation of graph :



BFS Algorithm of given graph:

The adjacency list representation of given graph is:

Vertex	Adjacency List
A	B, E, F
B	A, F, G
E	A, F
F	E, B, A
C	D, G
D	C, H
G	C, H, B
H	D, G

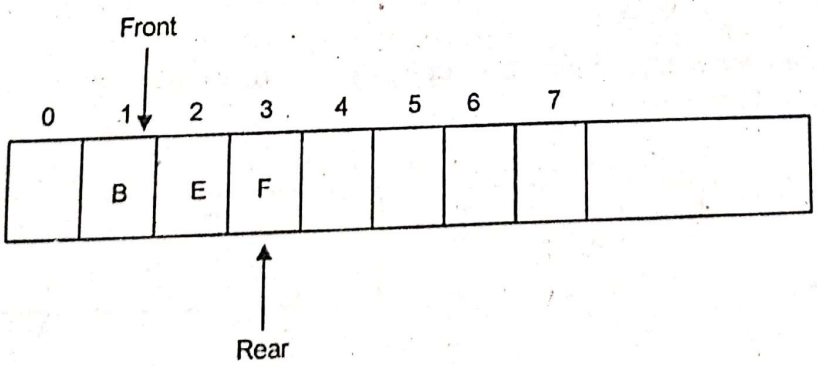
Step 1 Push the source vertex A to the queue:

0	1	2	3	4	5	6	7	
A								

Front = 0

Rear = 0

Step 2 Remove the front element, and push all the neighbouring vertices of A to the queue. If it is not in queue.



Front = 1

Rear = 3

Traversed node A.

Step 3 Remove the front element and push all the neighbouring vertices of B to the queue, If it is not in queue.

0	1	2	3	4	5	6	7	
		E	F	G				

Front = 2

Rear = 4

Traversed nodes A, B

Step 4 Remove the front element and push all the neighbouring vertices of E to the queue, It is not in queue.

0	1	2	3	4	5	6	7	
			F	G				

Front = 3

Rear = 4

Traversed nodes A, B, E

Step 5 Remove the front element and push all the neighbouring vertices of F into queue, If it is not in queue.

0	1	2	3	4	5	6	7	
				G				

Front = 4

Rear = 4

Traversed nodes A, B, E, F

Step 6 Remove the front element and push all the neighbouring vertices of G into queue, If it is not in queue.

0	1	2	3	4	5	6	7	
					C	H		

Front = 5

Rear = 6

Traversed nodes A, B, E, F, G

Step 7 Remove the front element and push all the neighbouring vertices of C into queue, If it is not in queue.

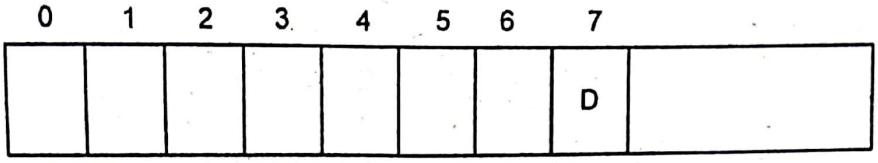
0	1	2	3	4	5	6	7	
						H	D	

Front = 6

Rear = 7

Traversed nodes A, B, E, F, G, C

Step 8 Remove the front element and push all the neighbouring vertices of H into ueue, If it is not in queue.

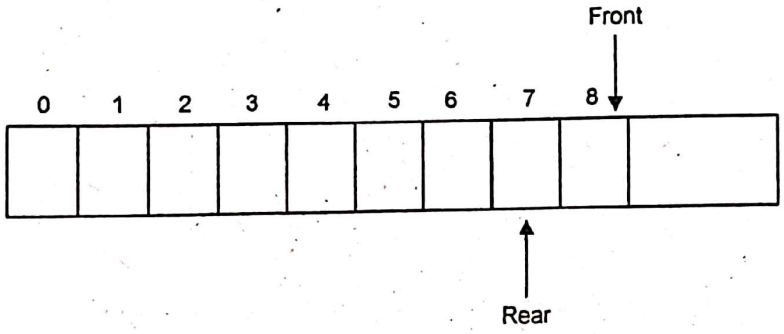


Front = 7

Rear = 7

Traversed nodes A, B, E, F, G, C, H

Step 9 Again remove the front element and push all the neighbouring vertices of D ıto queue, If it is not in queue.



Front = 8

Rear = 7

Here the front > rear that means BFS traversal is complete.

So the final traversed nodes are:

A, B, E, F, G, C, H, D.

Q.8. Why quicksort is generally considered to be better than mergesort? ;xplain Pseudocode of quicksort and mergesort procedures and compare their erformances. **(12.5)**

Ans. There are certain reasons due to which quicksort is better:

Why quicksort is better than mergesort ?

There are certain reasons due to which quicksort is better:

1. Auxiliary Space: Mergesort uses extra space, quicksort requires little space and xhibits good cache locality.

2. Worst Cases: The worst case of quicksort $O(n^2)$ can be avoided by using andomized quicksort. It can be easily avoided with high probability by choosing the ight pivot. Obtaining an average case behavior by choosing right pivot element makes t improvise the performance and becoming as efficient as Merge sort.

3. Locality of reference: Quicksort in particular exhibits good cache locality and his makes it faster than merge sort in many cases like in virtual memory environment.

4. Merge sort is better for large data structures: Mergesort is a stable sort, unlike quicksort and heapsort, and can be easily adapted to operate on linked lists and very large lists stored on slow-to-access media such as disk storage or network attached storage.

Pseudocode for merge sort:

To sort the entire sequence A[1 .. n], make the initial call to the procedure MERGE-SORT (A, 1, n).

```
MERGE-SORT (A, p, r)
1. IF p < r                          // Check for base case
2. THEN q = FLOOR[(p + r)/2]         // Divide step
3. MERGE (A, p, q)                   // Conquer step.
4. MERGE (A, q + 1, r)               // Conquer step.
5. MERGE (A, p, q, r)                // Conquer step.
```

The pseudocode of the MERGE procedure is as follow:

```
MERGE (A, p, q, r )
1. n1 → q − p + 1
2. n2 → r − q
3. Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
4. FOR i → 1 TO n1
5. DO L[i] → A[p + i − 1]
6. FOR j → 1 TO n2
7. DO R[j] → A[q + j]
8. L[n1 + 1] → ∞
9. R[n2 + 1] → ∞
10. i → 1
11. j → 1
12. FOR k → p TO r
13. DO IF L[i ] d" R[ j]
14. THEN A[k] → L[i]
15. i → i + 1
16. ELSE A[k] → R[j]
17. j → j + 1
```

Pseudocode for Quick Sort:

```
Quicksort(A,p,r) {
  if (p < r) {
    q <- Partition(A,p,r)
    Quicksort(A,p,q)
    Quicksort(A,q+1,r)
  }
}

Partition(A,p,r)
  x <- A[p]
  i <- p-1
  j <- r+1
  while (True) {
    repeat
      j <- j-1
```

```
    until (A[j] <= x)
    repeat
      i <- i+1
    until (A[i] >= x)
    if (i<-= “”> A[j]
    else
      return(j)
  }
}
```

Performance comparison of quick sort and merge sort:

Merge sort has the following performance characteristics:

- Best case: O(n log n)
- Average case: O(n log n)
- Worst case: O(n log n)

Quicksort has the following performance characteristics:

- Best case: O(n log n)
- Average case: O(n log n)
- Worst case: O(n^2)

Q.9. Define separate chaining, linear probing and quadratic probing. Given the values {2341, 4234, 2839, 430, 22, 397, 3920}, a hash table of size 7, and hash function h(x) = x mod 7, show the resulting tables after inserting the values in the given order with each of these coillision strategy. (12.5)

Ans. Values are: {2341, 4234, 2839, 430, 22, 397, 3920}

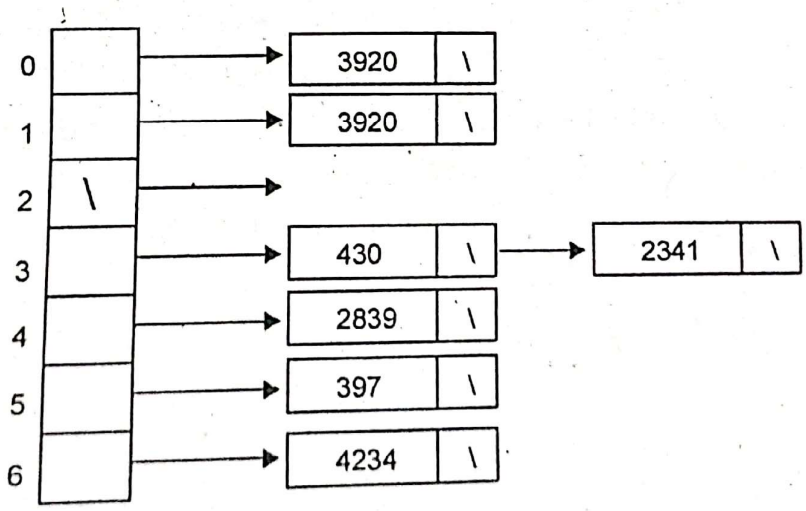
Table size M = 7

Hash function H(X) = X Mod 7

so

2341 % 7 = 3
4234 % 7 = 6
2839 % 7 = 4
430 % 7 = 3
22 % 7 = 1
397 % 7 = 5
3920%7 = 0

(1) Separate chaining



(2) Linear Probing Final Table

0	1	2	3	4	5	6
397	22	3920	2341	2839	430	4234

Value (430) is collide at 3

so the next probe is combuted as:

$$H(430, 1) = [430 \bmod 7 + 1] \bmod 7$$
$$= [3 + 1] \bmod 7 = 4 \bmod 7 = 4$$

But the T [4] is not free so again:

$$H(430, 2) = [430 \bmod 7 + 2] \bmod 7 = 5 \bmod 7 = 5$$

Now the value (397) is collide at 5.

so the next probe is:

$$H(397, 1) = 6$$

But six is not free so next probe is:

$$H(397, 2) = 0$$

Now the value (3920) is collide at 0.

So the next probe is:

$$H(3920, 1) = 1$$

But T[1] is not free so next probe is:

$$H(3920, 2) = 2.$$

(3) Quadratic probing **Final Table**

0	1	2	3	4	5	6
430	22	3920	2341	2839	397	4234

Now the (430) is collide at 3. So the next probe is computed as:

$$H(430, 1) = [430 \% 7 + 1^2] \bmod 7$$
$$= (3 + 1) \bmod 7 = 4$$

But the T[4] is not free so next is:

$$H(430, 2) = [430\%7 + 2^2] \bmod 7$$
$$= (3 + 4) \% 7 = 0$$

Now (3920) is collide at 0. So the next probe is:

$H(3920, 1) = 1$ Here T [1] is not free.

$H(3920, 2) = 4$ again T [4] is not free

$H(3920, 3) = 2$ It is free.

FIRST TERM EXAMINATION [SEP. 2018]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

Time : 1.5 hrs. M.M. : 30

Note: Q. No. 1 is compulsory. Attempt any two more Questions from the rest.

Q.1. Attempt all parts:

(a) Calculate the postfix expression using stack also demonstrate the steps. (2)

5, 3, + 8, 2, –,*

Ans.

Symbol Scanned	Stack	Operation (B op A)
5	5	
3	5,3	
+	8	[5 + 3] (A = 3, B = 5)
8	8,8	
2	8, 8, 2	
–	8, 6	[8 – 2] (A = 2, B = 8)
*	48	[8 * 6] (A = 6, B = 8]

Q. 1. (b) What do you understand by the term space-time tradeoff? (2)

Ans. Time-Space tradeoff: Is a way of solving a problem calculation in less time by using more space (or memory), or by solving a problem in very little space by spending a long time. Most computers have a large amount of space, but not infinite space. Also, most people are willing to wait a little while for a big calculation, but not forever. So if your problem is taking a long time but not much memory, a space-time tradeoff would let you use more memory and solve the problem more quickly or it could be solved very quickly but requires more memory than you have, you can try to spend more time solving the problem in the limited memory.

Q. 1. (c) Differentiate between arrays and linked lists. (2)

Ans.

ARRAY	LINKED LIST
Array is a collection of elements of similar data type.	Linked List is an ordered collection of elements of same type, which are connected to each other using pointers.
Array supports **Random Access.**	Linked List supports **Sequential Access.**
In an array, elements are stored in **contiguous memory location** or consecutive manner in the memory.	In a linked list, new elements can be stored anywhere in the memory.
In array, **Insertion and Deletion** operation takes more time, as the memory locations are consecutive and fixed.	Insertion and Deletion operations are **fast** in linked list.
Memory is allocated as soon as the array is declared, at **compile time.** It's also known as **Static Memory Allocation.**	Memory is allocated at **runtime**, as and when a new node is added. It's also known as **Dynamic Memory Allocation.**

In array, each element is independent and can be accessed using it's index value.	In case of a linked list, each node/element points to the next, previous, or maybe both nodes.
Array can **single dimensional, two dimensional** or **multidimensional** .	Linked list can be **Linear(Singly), Doubly** or **Circular** linked list.

Q. 1. (d) Briefly explain different types of queues. (2)

Ans. Types of queue:

Simple queue: In a simple FIFO queue, the order is retained and data leaves in the same order in which it comes in.

Priority queue: A queue in which the elements are given a predefined priority.

Circular queue: Similar to a simple queue, except that the back of the queue is followed by the front of the queue.

Double ended queue (Dequeue): Similar to the simple queue but can add or remove elements from either the front or the back of the queue.

Q. 1. (e) Define the following terms used in tree: (2)

(i) Terminal Node

(ii) Siblings

(iii) Balance factor

(iv) Height of tree

Ans. (i) Terminal Node: In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

(ii) Siblings: In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.

(iii) Balance Factor: We define balance factor for each node as :

balanceFactor = height(left subtree) - height(right subtree)

The balance factor of any node of an AVL tree is in the integer range [-1,+1]. If *after* any modification in the tree, the balance factor becomes less than –1 or greater than +1, the subtree rooted at this node is unbalanced, and a rotation is needed.

(iv) Height of tree: In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'.**

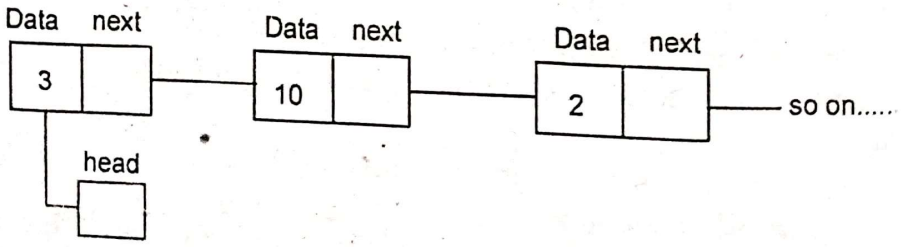
Q. 2. Attempt all parts:

(a) Explain different types of linked lists. (5)

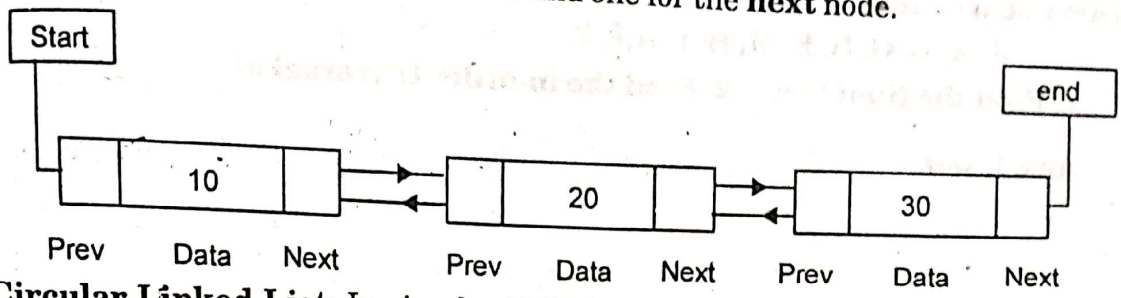
Ans. Types of Linked Lists: There are 3 different implementations of Linked List available, they are;

(1) Singly Linked List: Singly linked lists contain nodes which have a **data** part as well as an **address part** i.e. next, which points to the next node in the sequence of nodes.

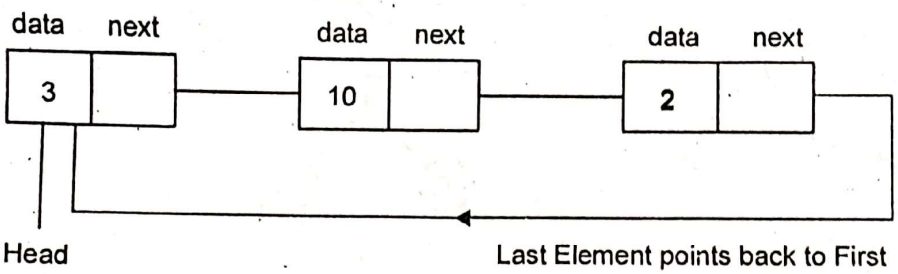
The operations we can perform on singly linked lists are **insertion, deletion** and **traversal.**



Doubly Linked List: In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



Circular Linked List: In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.



Q. 2. (b) Write an algorithm to delete the second-last node of a doubly linked list. **(5)**

Ans.
```
ptr=first;
while(ptr->rpt->rpt! = NULL)
{
pre=ptr:
ptr=ptr->rpt;
}
pre->rpt=ptr->rpt;
ptr->rpt->lpt=pre;
free(ptr);
```

Q. 3. Attempt all parts

(a) Write the algorithms for PUSH and POP operations in stack. **(4)**

Ans. Push Operation: The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps

- **Step 1** – Checks if the stack is full.
- **Step 2** – If the stack is full, produces an error and exit.
- **Step 3** – If the stack is not full, increments **top** to point next empty space.
- **Step 4** – Adds data element to the stack location, where top is pointing.
- **Step 5** – Returns success.

Pop Operation: Accessing the content while removing it from the stack, is known as a Pop Operation. A Pop operation may involve the following steps–

- **Step 1** – Checks if the stack is empty.
- **Step 2** – If the stack is empty, produces an error and exit.
- **Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** – Decreases the value of top by 1.
- **Step 5** – Returns success.

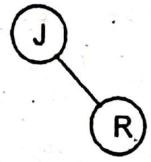
Q. 3. (b) Insert the following elements in alphabetical order into an empty binary search tree: (5)

J, R, D, G, B, E, M, H, P, A, F, Z

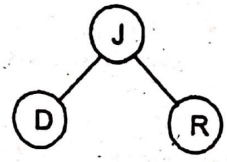
1. Find the final tree 2. Find the in-order traversal of.

Ans. Insert J

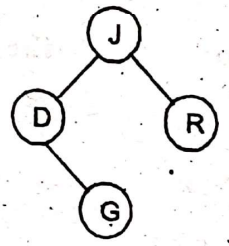
Insert R



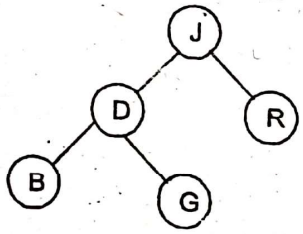
Insert D



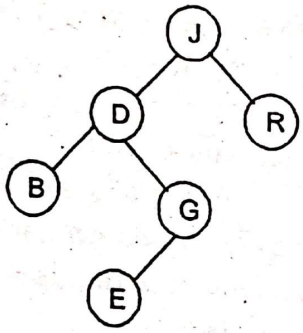
Insert G



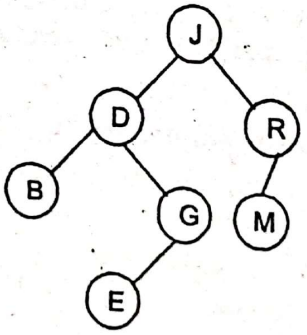
Insert B



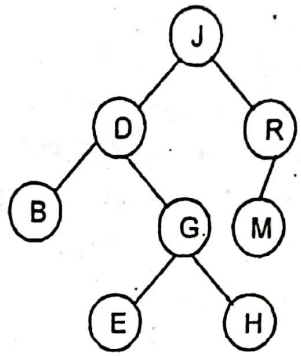
Insert E



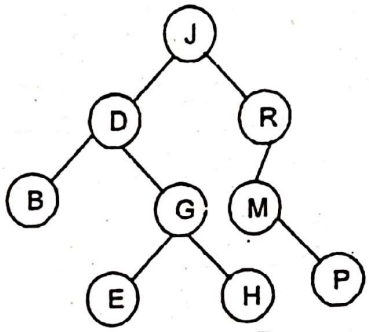
Insert M



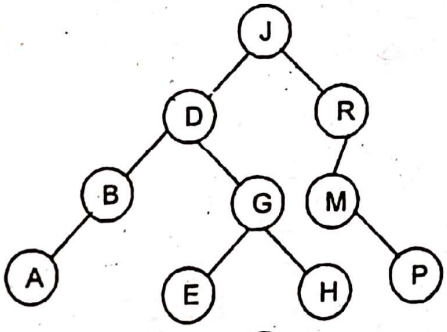
Insert H



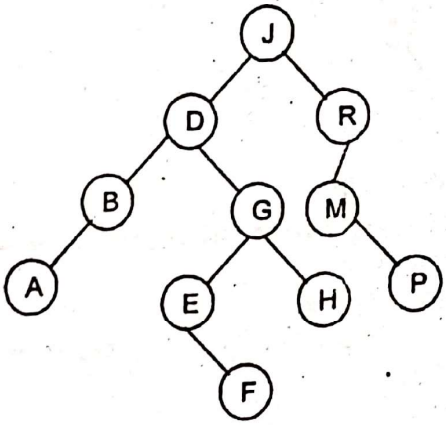
Insert P



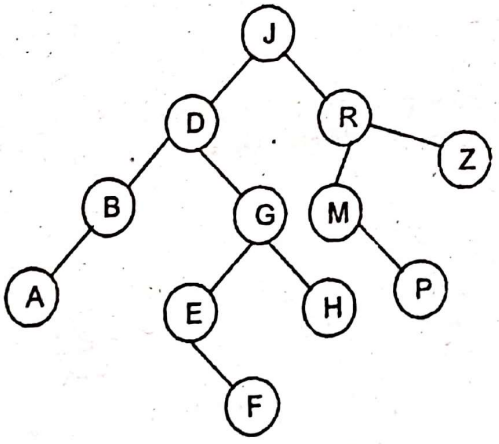
Insert A



Insert F



Insert Z



In order Transversal of above tree is:
A B E F G H D J M P R Z

Q. 4. Attempt all parts:

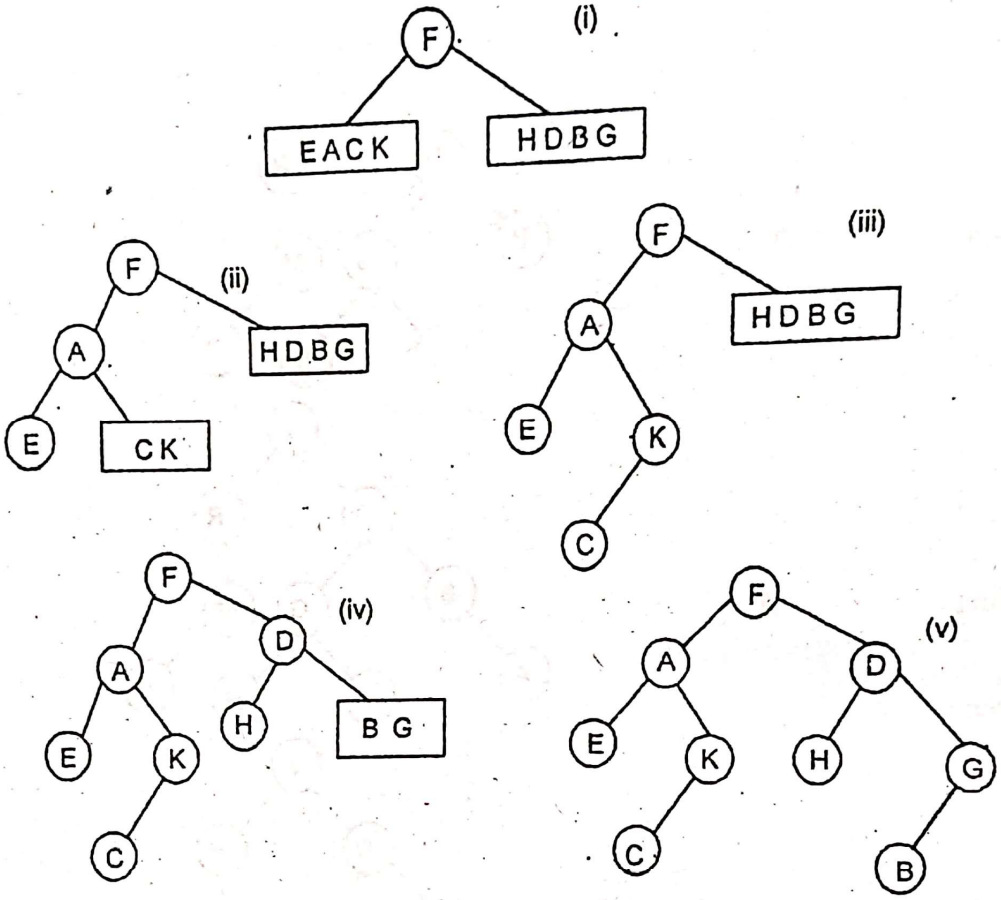
(a) A binary tree has 9 nodes. The inorder and preorder traversals of the tree yield the following sequences of nodes: (5)

INORDER: E A C K F H D B G

PREORDER: F A E K C D H G B . **Draw the tree.**

Ans. INORDER: E A C K F H D B G

PREORDER: F A E K C D H G B



This is final tree T

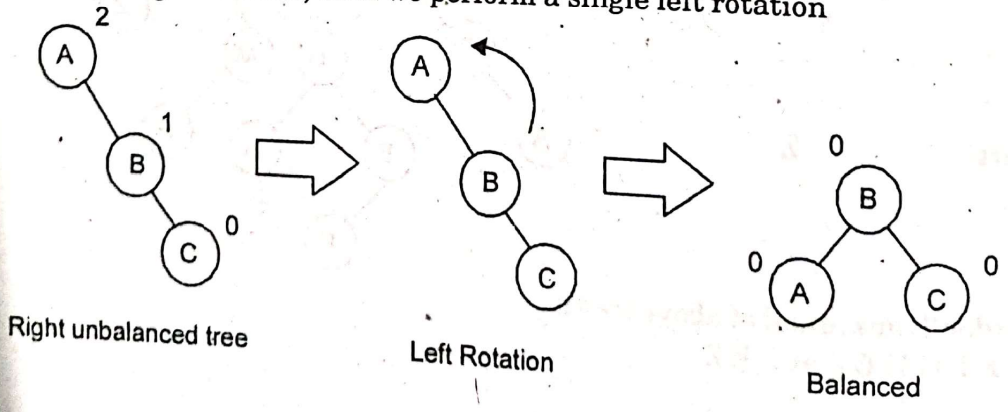
Q. 4. (b) Explain different types of rotations in AVL tree with the help of suitable examples. (5)

Ans. AVL Rotations

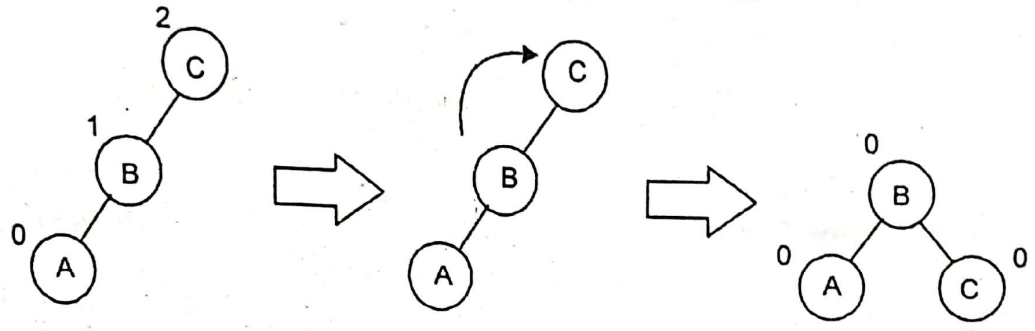
To balance itself, an AVL tree may perform the following four kinds of rotations–

- R-R rotation
- L-L rotation
- L-R rotation
- R-L rotation

R-R Rotation: If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation

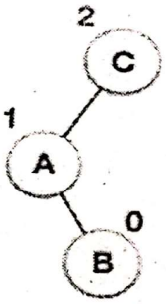
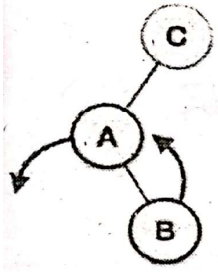
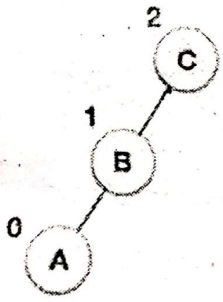
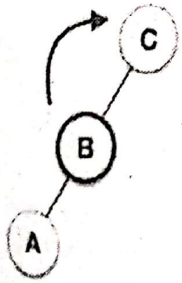


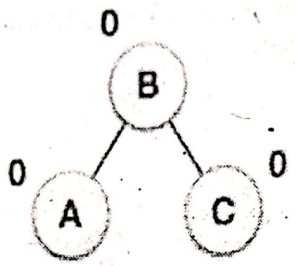
L-L Rotation: AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



Left unbalanced Tree Right Rotation Balanced Tree

L-R Rotation: Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

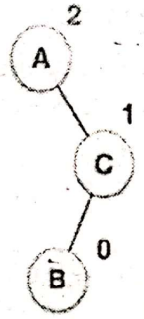
State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.



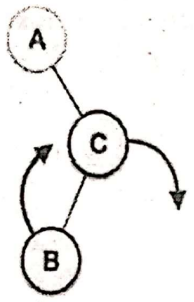
The tree is now balanced.

R-L Rotation: The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

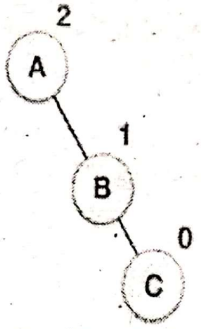
State **Action**



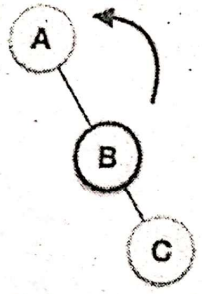
A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.



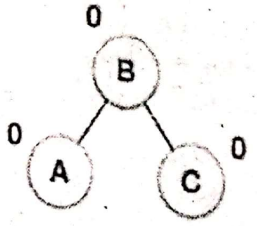
First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.



Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.



A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.



The tree is now balanced.

END TERM EXAMINATION [NOV-DEC 2018]
THIRD SEMESTER [B.TECH]
DATA STRUCTURES [ETCS–209]

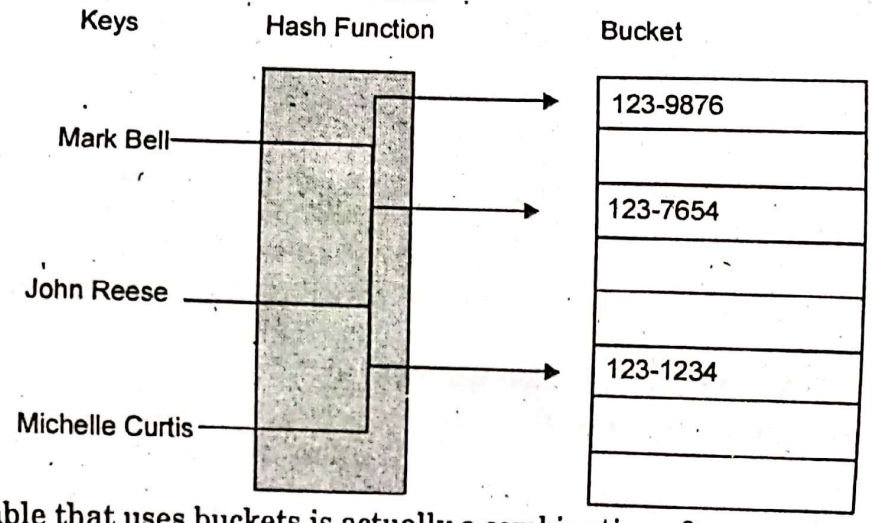
Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions including Q. No 1 which is compulsory. Select one question from each unit.*

Q.1. Attempt all:

(a) What is bucket Hashing? **(2.5)**

Ans. Hash buckets are used to apportion data items for sorting or lookup purposes The aim of this work is to weaken the linked lists so that searching for a specific item can be accessed within a shorter timeframe.



A hash table that uses buckets is actually a combination of an array and a linked list. Each element in the array (the hash table) is a header for a linked list. All elements that hash into the same location will be stored in the list.

The hash function assigns each record to the first slot within one of the buckets. In case the slot is occupied, then the bucket slots will be searched sequentially until an open slot is found.

In case a bucket is completely full, the record will get stored in an overflow bucket of infinite capacity at the end of the table. All buckets share the same overflow bucket. However, a good implementation will use a hash function that distributes the records evenly among the buckets so that as few records as possible go into the overflow bucket.

Q. 1. (b) What is the condition to check overflow and underflow condition in circular queue? **(2.5)**

Ans. If value of REAR variable is greater then or equal to SIZE – 1 and value of FRONT variable is equal to 0 then we can not insert an element into Circular Queue. This condition is known as **"Overflow"**.

If Circular Queue is empty then we can not delete an element from Circular Queue. This condition is known as **"Underflow"**.

Q. 1. (c) What is Header Linked list? **(2.5)**

Ans. A Header linked list is one more variant of linked list. In Header linked list, we have a special node present at the beginning of the linked list. This special node is used to store number of nodes present in the linked list. In other linked list variant, if we want to know the size of the linked list we use traversal method. But in Header linked list, the size of the linked list is stored in its header itself.

Types of Header linked list:

(a) Grounded header linked list.

(b) Circular header linked list.

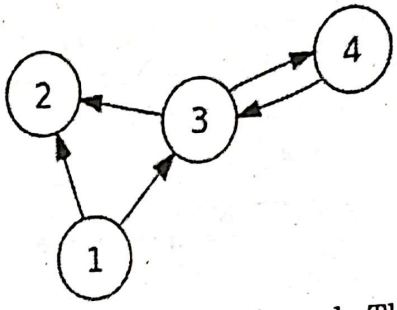
Q. 1. (d) What is the maximum possible Number of nodes in binary tree at level 7. (2.5)

Ans. It the level of root is 1 and left and right children of root is on level 2. Then maximum no. of nodes at level 7 in binary tree are

$$2^7 - 1 = 128 - 1 = 127.$$

Q. 1. (e) Define In-degree and Out-degree of a node in the graph. (2.5)

Ans.

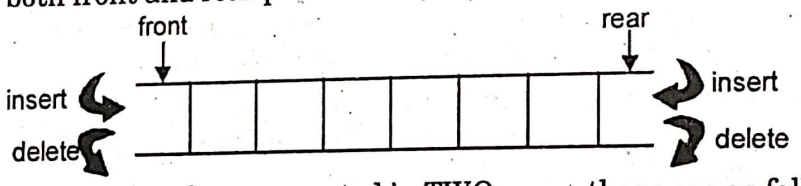


In Degree: This is applicable only for directed graph. This represents the number of edges incoming to a vertex. In above directed graph, degree of 1 is 0 and degree of 2 is 2.

Out degree: This is also applicable only for directed graph. This represents the number of edges outgoing from a vertex. In above directed graph, degree of 1 is 2 and degree of 2 is 0.

Q. 1. (f) Explain Doubly ended Queue. (2.5)

Ans. Double Ended Queue Data structure: Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (**front** and **rear**). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

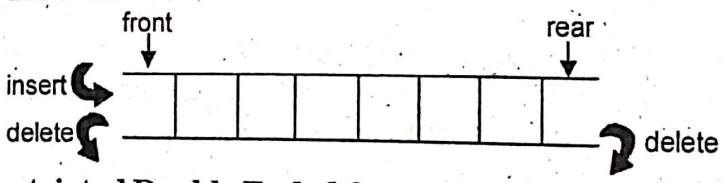


Double Ended Queue can be represented in TWO ways, those are as follows...

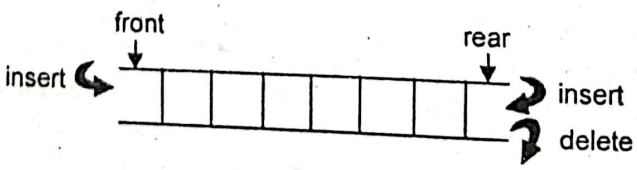
(a) Input Restricted Double Ended Queue

(b) Output Restricted Double Ended Queue

(a) Input Restricted Double Ended Queue: In input restricted double-ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



(b) Output Restricted Double Ended Queue: In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



Q. 1. (g) For 1-D array if base address is 2000, find 6th index element address if a data stored in this array needs only 3 byte.

Ans. Base (A) = 2000 **(2.5)**

$$w = 3 \text{ byte}$$
$$k = 6$$
$$\text{Loc}(A[k]) = \text{Base}(A) + w * k$$
$$\text{Loc}(A[6]) = 2000 + 3 * 6$$
$$= 2018$$

Q. 1. (h) Explain application of DFS. **(2.5)**

Ans. Applications of Depth First Search

Depth-first search (DFS) is an algorithm (or technique) for traversing a graph.

Following are the problems that use DFS as a building block.

(1) For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.

(2) Detecting cycle in a graph: A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges

(3) Topological: Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when re computing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers.

(4) Finding Strongly Connected Components of a graph A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.

(5) Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

Q. 1. (i) Explain threaded Binary tree. **(2.5)**

Ans. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

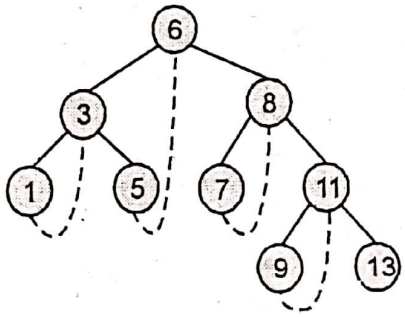
There are two types of threaded binary trees.

(1) Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)

(2) Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

The threads are also useful for fast accessing ancestors of a node.

Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.

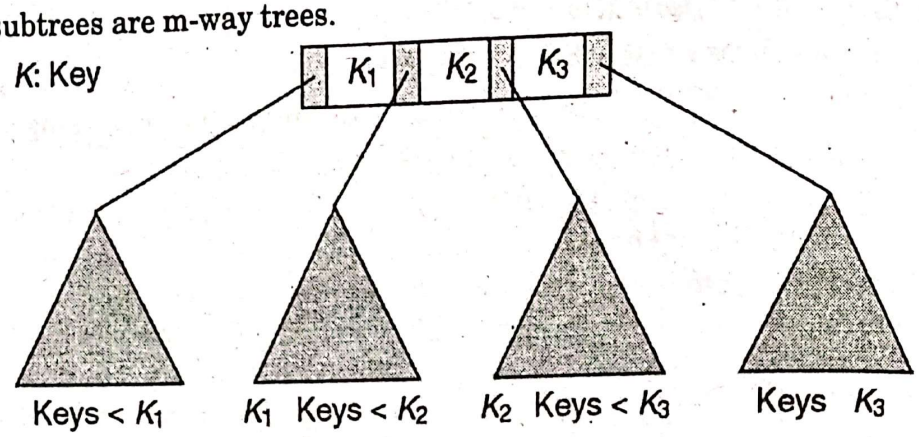


~~**Q. 1. (j) What is M-way Tree?**~~ ~~**(2.5)**~~

Q. 1. (j) What is M-way Tree? (2.5)

Ans. Multi-Way Tree (m-way trees): A main problem with binary trees is that the search path may become deep. Main uses of m-way trees are external file systems (ISAM, VSAM) and spell checkers. M-way trees have the following properties:

1. Each node has 0 .. m subtrees
2. A node with k < m subtrees, contains k-1 keys.
3. The key values of the first subtree are all less than the key value.
4. The data entries are ordered.
5. All subtrees are m-way trees.



UNIT-I

Q. 2. (a) A 2-dimensional array X[5] [4] is stored row-wise in the memory. The 1st element of the array is stored at location 80. Find the memory location of X[3] [2] if each element of array required 4 byte memory space. (6)

Ans. Suppose the first element of the given array is x[1] [1] in row major order.

Loc (A [i] [j]) = Base (A) + w[n (i – Lower bound for row index) + (j – lower bound for column index)]

Here

$$w = 4 \text{ byte; Base (A)} = 80$$
$$n = 4, \quad i = 3, j = 2$$
$$\text{Loc (x [3] [2])} = 80 + 4\,[4\,(3 - 1) + (2 - 1)]$$
$$= 80 + 4\,[4 \times 2 + 1]$$
$$= 80 + 36 = 116.$$

Q. 2. (b) Consider the following infix expression. (6.5)

A+(B*C-(D/E↑F)*G)*H

Transform this expression into postfix expression using stack.

Ans.

Character Scanned	Stack	Postfix Notation
A		A
+	+	A
(	+(	A
B	+(	AB
*	+(*	AB
C	+(*	ABC
-	+(-	ABC*
(	+(-(	ABC*
D	+(-(	ABC*D

/	+(-(/	ABC*D
E	+(-(/	ABC*D
↑	+(-(/↑	ABC*DE
F	+(-(/↑	ABC*DEF
)	+(-	ABC*DEF↑/
*	+(-*	ABC*DEF↑ /
G	+(-*	ABC*DEF↑/G
)	+	ABC*DEF↑/G*-
*	+*	ABC*DEF↑/G*-
H	+*	ABC*DEF↑/G*-H
		ABC*DEF↑/G*-H

Q. 3. (a) Write an algorithm/Program to implement Queue using linked list. **(6)**

Ans. Implementation of Queue Data structure using Linked List -

```
#include<stdio.h>
#include<conio.h>
struct Node
{
  int data;
  struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice:");
    scanf("%d",&choice);
    switch(choice){
    case 1: printf("Enter the value to be insert:");
                scanf("%d", &value);
               insert(value);
                break;
            case 2: delete(); break;
           case 3: display(); break;
            case 4: exit(0);
        default: printf("\nWrong selection!!! Please try again!!!\n");
```

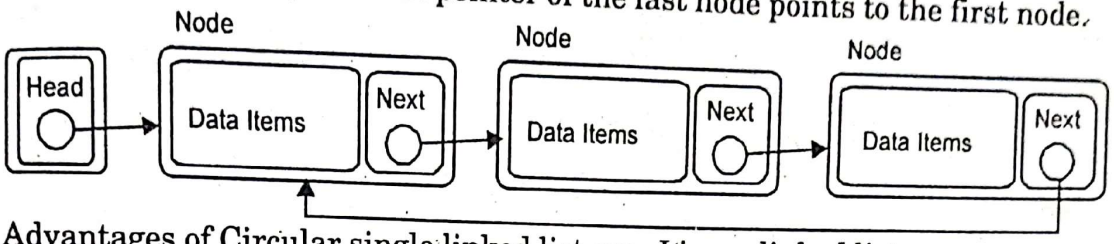
```
    }
  }
}
void insert(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void delete()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
        printf("%d—>",temp->data);
        temp = temp -> next;
    }
    printf("%d—>NULL\n",temp->data);
  }
}
```

Q. 3. (b) What is circular linked list? What are its advantages over linear linked list? Write algorithm to insert a node at desired position in circular linked list. (6.5)

Ans. Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.



Advantages of Circular single linked list over Linear linked list :

In single linked list you can't go back. But using circular linked list you can access previous node but somehow it is expensive.

The last node of single linked list is not used very effectively.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the circular linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.

Step 7 - If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).

Step 8 - If temp is last node then set temp → next = newNode and newNode → next = head.

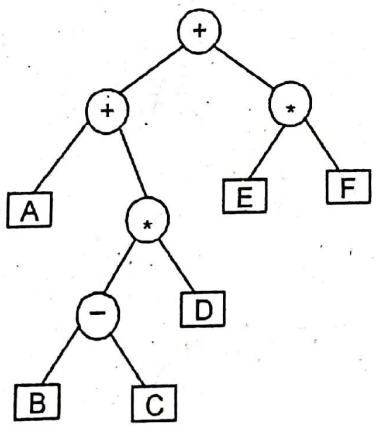
Step 9 - If temp is not last node then set newNode → next = temp → next and temp → next = newNode.

UNIT-II

Q. 4. (a) Construct expression tree for the following expression. (6)

A+(B-C)*D+ (E*F)

Ans.



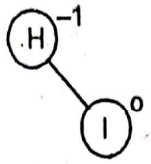
Q. 4. (b) Create AVL search tree from given set of values and explain rotations: (6.5)

H I J B A E C F D G K L

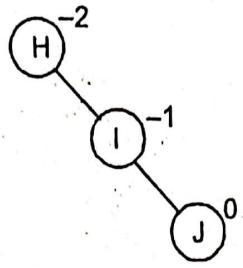
Ans. H I J B A E C F D G K L

Insert H

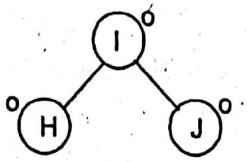
Insert I



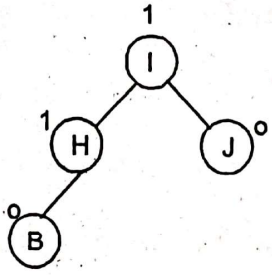
Insert J



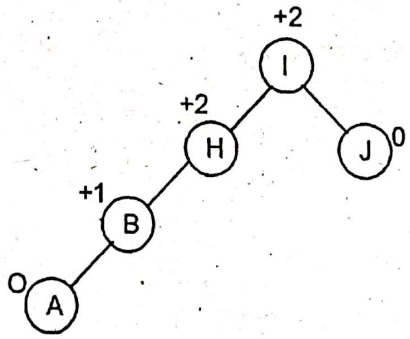
After R–R Rotation



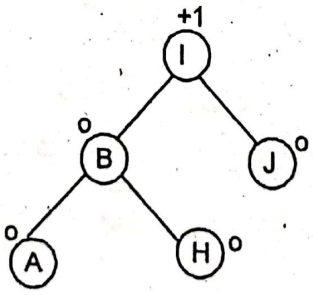
Insert B



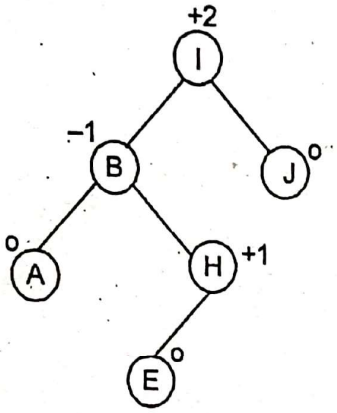
Insert A



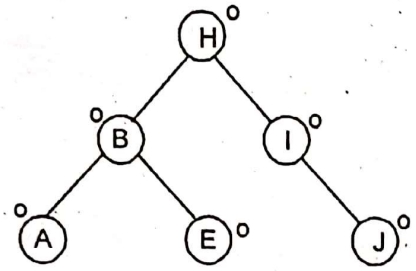
After L – L Rotation



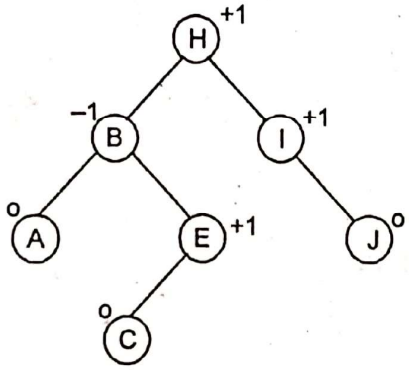
Insert E



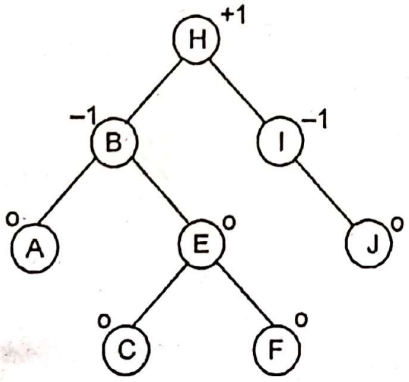
After L–R Rotation



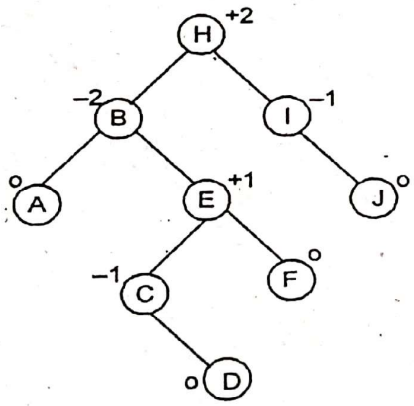
Insert C



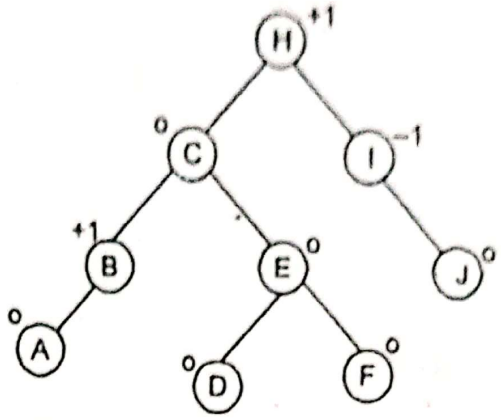
Insert F



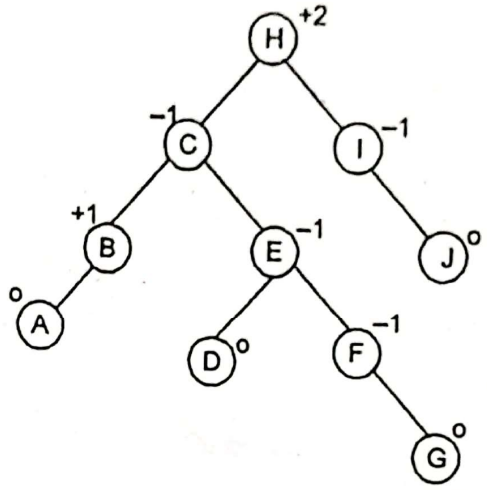
Insert D



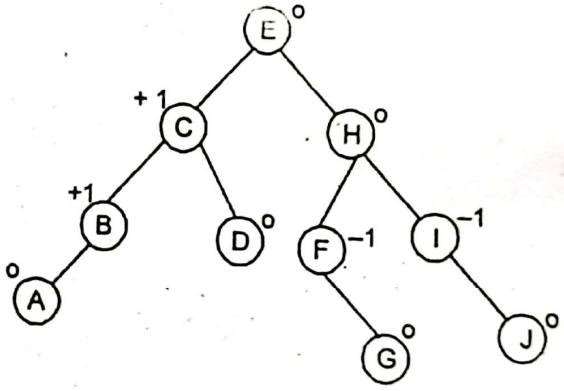
After R–L Rotation



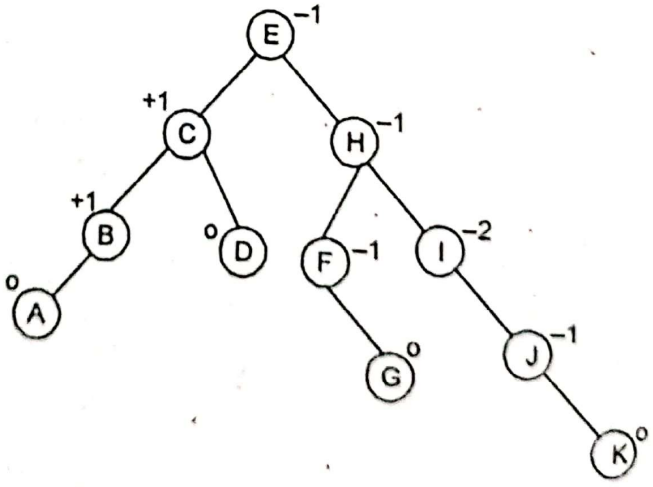
Insert G



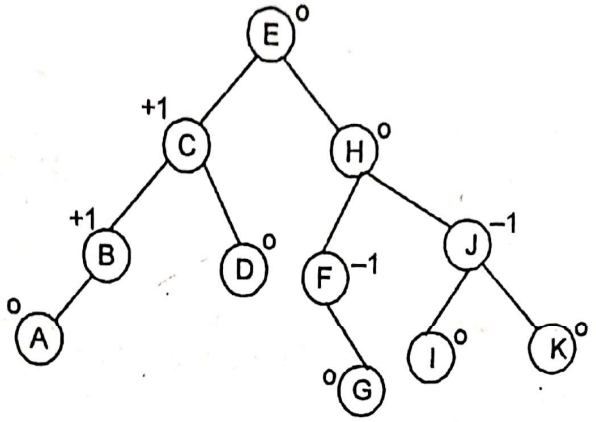
After L – R Rotation



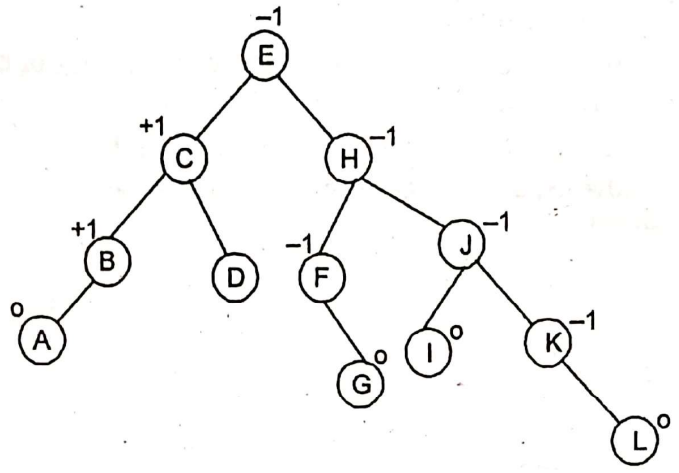
Insert K



After R – R Rotation



Insert L



Q. 5. (a) Explain following (Give one example of each) **(2 × 3 = 6)**

(i) BST **(ii) Complete Binary tree**

(iii) Heap Tree

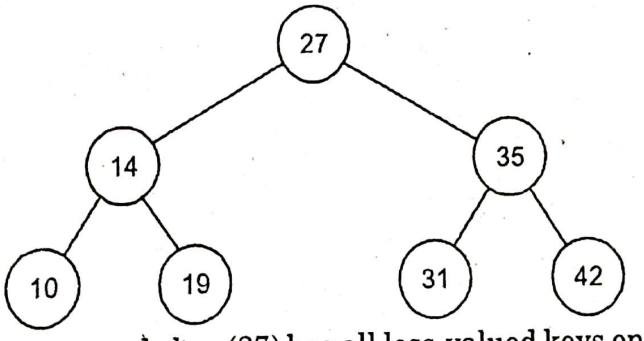
Ans. (i) BST: A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties

The left sub-tree of a node has a key less than or equal to its parent node's key.

The right sub-tree of a node has a key greater than to its parent node's key.

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

(ii) Complete Binary Tree: A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible

Following are examples of Complete Binary Trees

```
      18                          18
     /  \                        /  \
    15   30                     15   30
   / \   / \                   / \   / \
  40 50 100 40                40 50 100 40
                             / \ /
                            8  7 9
```

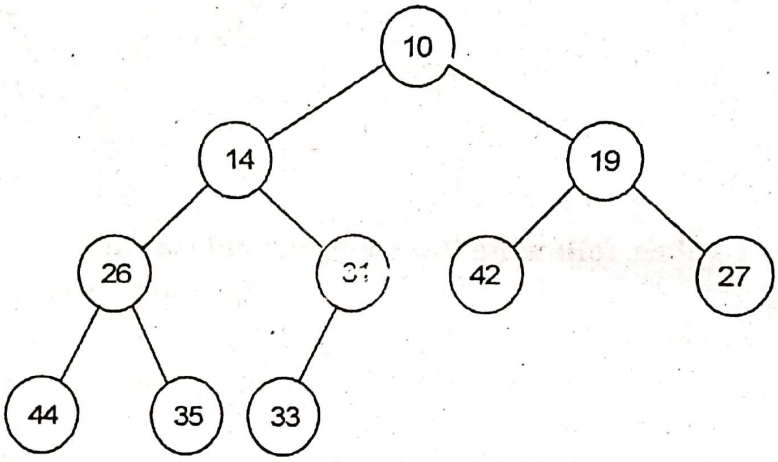
(iii) Heap tree: Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If α has child node β then

key(α) ≥ key(β)

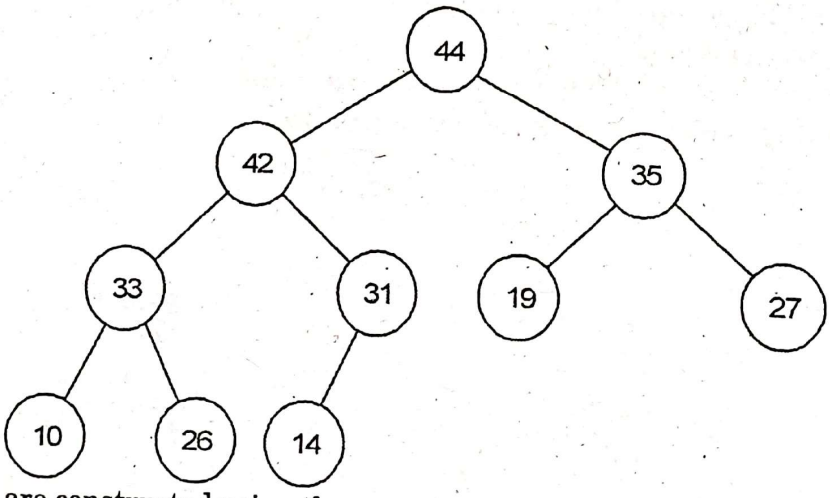
As the value of parent is greater than that of child, this property generates Max Heap. Based on this criteria, a heap can be of two types

For Input → 35 33 42 10 14 19 27 44 26 31

Min-Heap – Where the value of the root node is less than or equal to either of its children.



Max-Heap– Where the value of the root node is greater than or equal to either of its children.

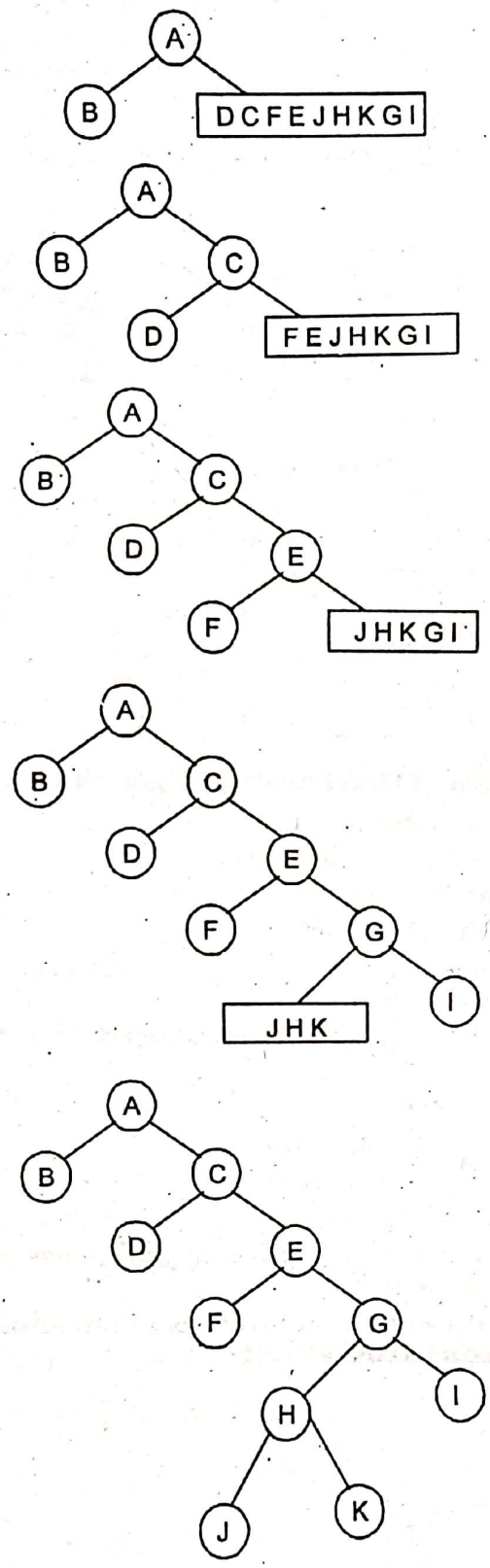


Both trees are constructed using the same input and order of arrival.

Q. 5. (b) Draw Binary tree when Inorder and Preorder traversal is given as follows and Write algorithm for the same. **(6.5)**

Pre: ABCDEFGHIJKL

In: BADCFEJHKGL
Ans. Pre: A B C D E F G H I J K
In: B A D C F E J H K G I

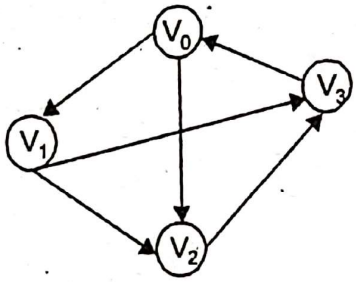


UNIT-III

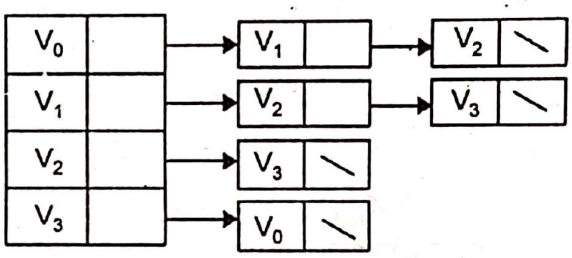
Q. 6. (a) Draw directed graph that corresponds to following adjacency matrix. Also write down adjacency list corresponding to graph. **(6)**

	V_0	V_1	V_2	V_3
V_0	0	1	1	0
V_1	0	0	1	1
V_2	0	0	0	1
V_3	1	0	0	0

Ans. Directed graph of given adjacency matrix.



The adjacency list of above graph is



Q. 6. (b) Write BFS and DFS traversal algorithm for a graph. **(6.5)**

Ans. DFS Algorithmic Steps

Step 1: Push the root node in the Stack.

Step 2: Loop until stack is empty.

Step 3: Peek the node of the stack.

Step 4: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.

Step 5: If the node does not have any unvisited child nodes, pop the node from the stack.

BFS Algorithmic Steps

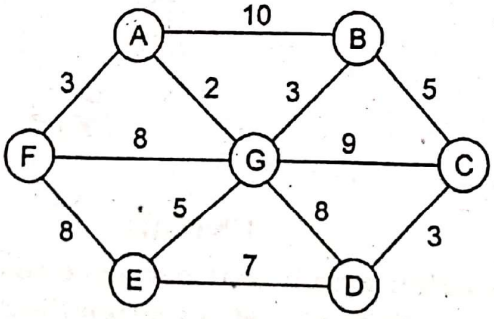
Step 1: Push the root node in the Queue.

Step 2: Loop until the queue is empty.

Step 3: Remove the node from the Queue.

Step 4: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

Q. 7. (a) Explain minimum spanning tree. Find minimum spanning tree for the following graph using kruskal's algo. **(6)**

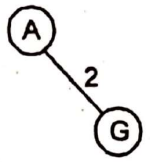


Ans. Minimum Spanning Tree (MST): In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

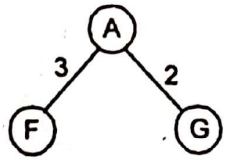
First sort the edges into increasing order of weight

Edge	Weight
(A, G)	2
(A, F)	3
(B, G)	3
(C , D)	3
(G, E)	5
(B, C)	5
(E, D)	7
(G, D)	8
(F, G)	8
(F, E)	8
(G, C)	9
(A, B)	10

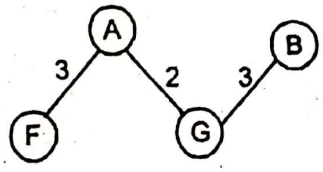
Now we take the first edge (A, G)



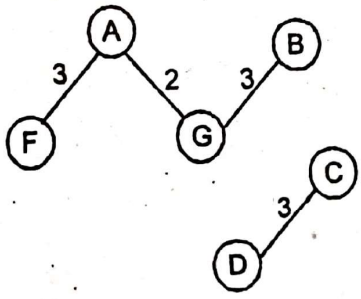
Then next edge (A, F)



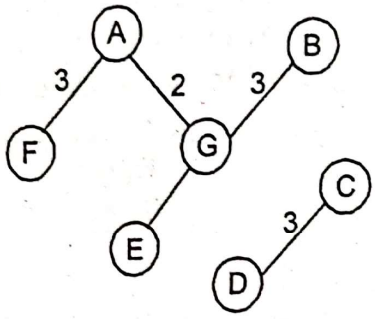
Next is the (B, G)



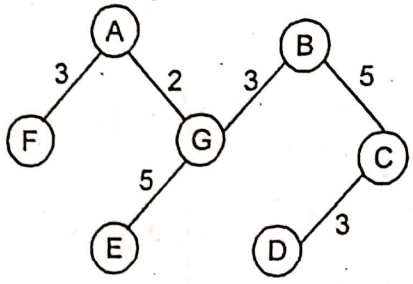
Next (C, D) edge is considered and forest becomes.



Now next edge is the (G, E)



Next is the (B, C)



Next edge is the (E, D) but it makes loop so ignore it.

Next edge is (G, D) again it makes a loop ignore it.

Next edge (F, G) again it makes a loop ignore it.

Next edge (F, E) again loop, ignore it.

Next edge (G, C) ignore it.

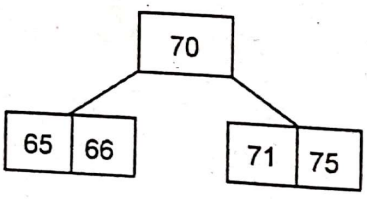
and Last is the (A, B) it makes a loop so ignore it. So above is the minimum spanning tree of given graph, and the total cost is 21.

Q. 7. (b) Create B-tree of order 5 taking the keys in give order into an initially empty tree 65, 71, 70, 66, 75, 68, 72, 77, 74, 69, 83, 73, 82, 88, 67, 76, 78, 84, 85, 80 then delete 75, 69, 73, 88, 67. **(6.5)**

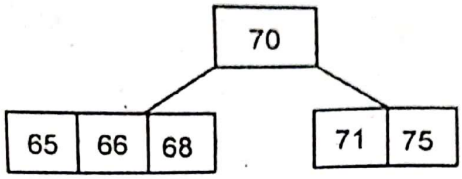
Ans. B-Tree of order 5:

65, 71, 70, 66, 75, 68, 72, 77, 74, 69, 83, 73, 82, 88, 67, 76, 78, 84, 85, 80

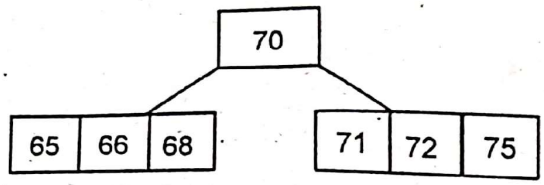
Insert	65	[65]
Insert	71	[65 | 71]
Insert	70	[65 | 70 | 71]
Insert	66	[65 | 66 | 70 | 71]
Insert	75	Here node way already full 80 after insertion of 75 it is splitted into two nodes.



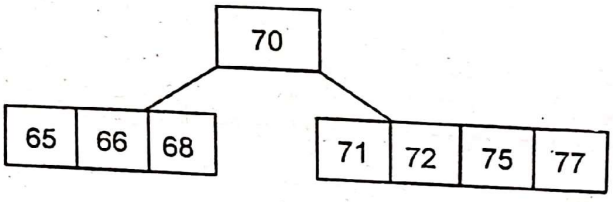
Insert 68



Insert 72

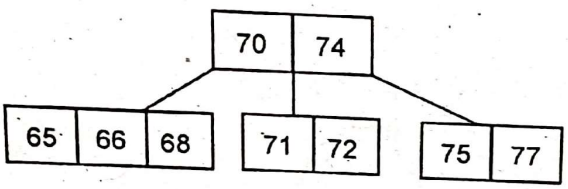


Insert 77

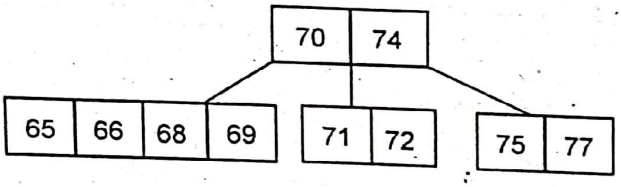


Insert 74

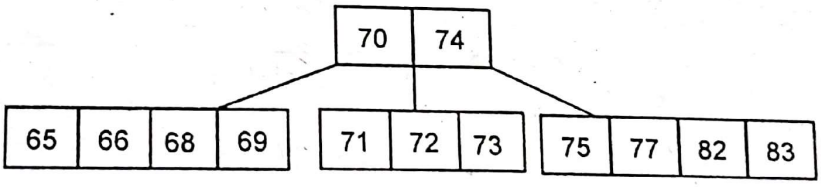
Here node is already full. So after insertion of 74 it is splitted into two nodes.



Insert 69

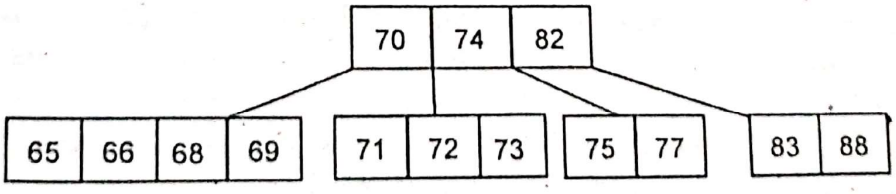


Insert 83, 73, 82



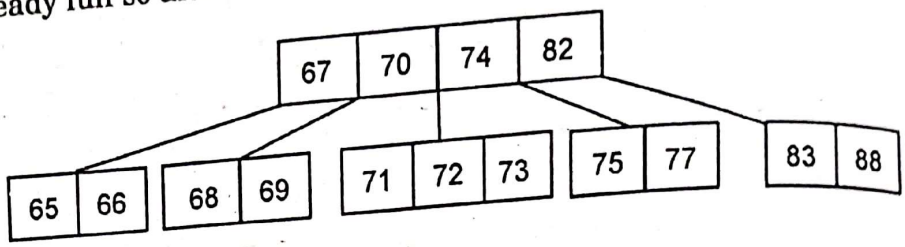
Insert 88

Here node is already full. So after insertion of 88 it is splitted into two nodes.

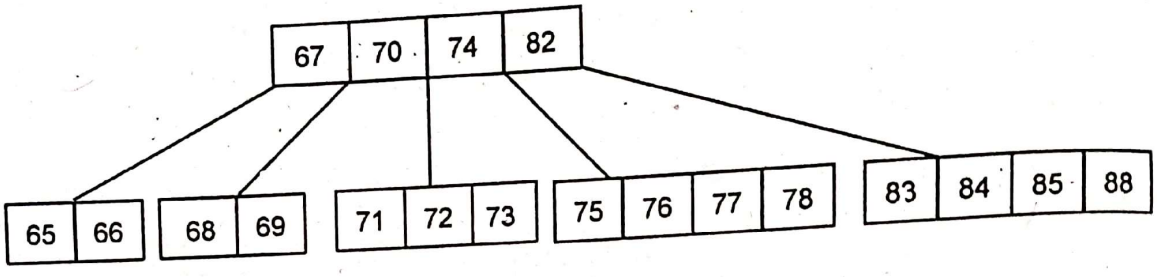


Insert 67

Here node is already full so after insertion of 67 it is splitted into two nodes.

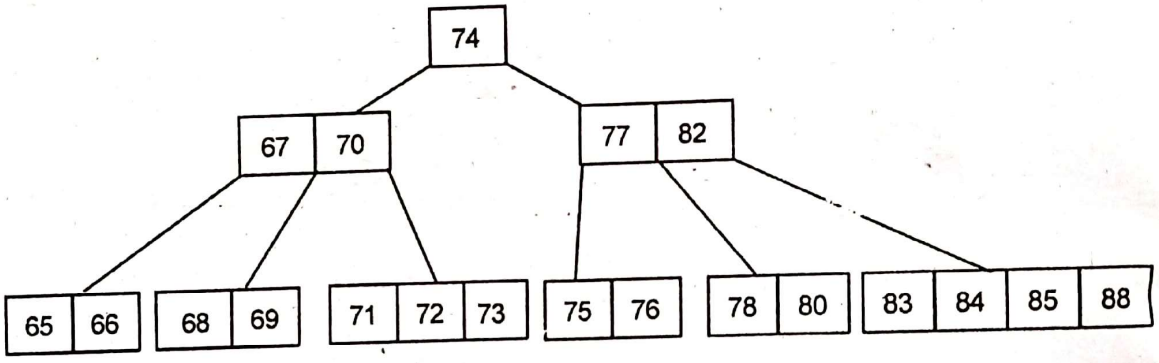


Insert 76, 78, 84, 85

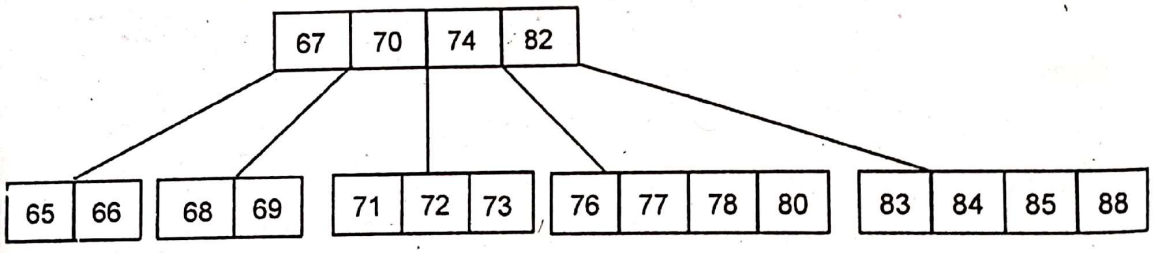


Insert 80

Here node was already full so after insertion 80 it is splitted in two nodes.



After Deletion of 75



UNIT-IV

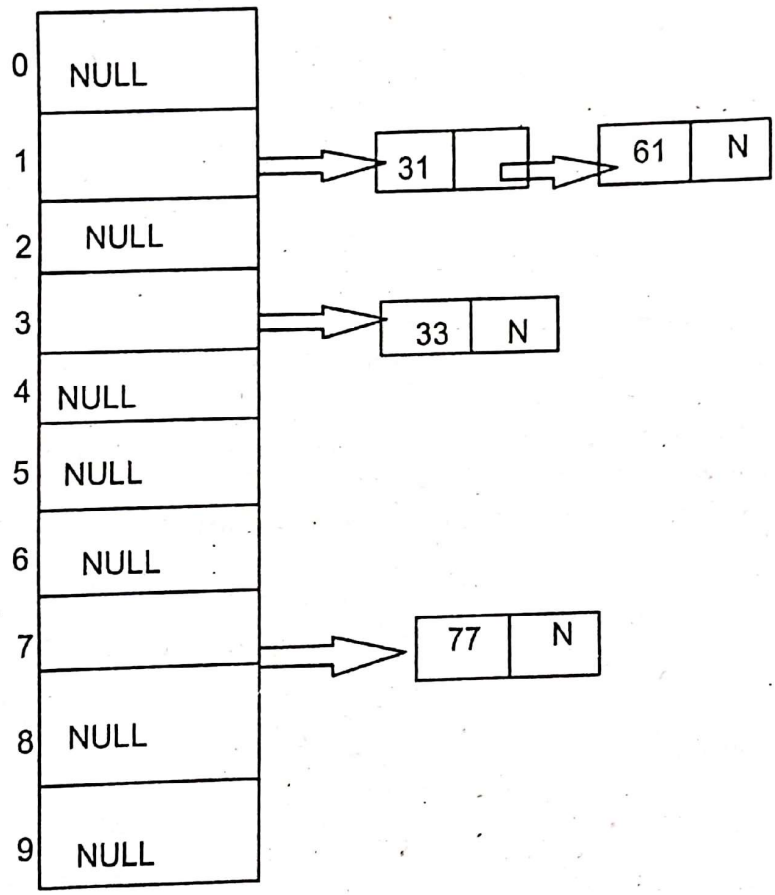
Q. 8. (a) What is Hashing? Explain various hashing algorithm. Discuss various collision resolution techniques. **(6)**

Ans. Hashing: There are many possibilities for representing the dictionary and one of the best methods for representing is hashing. Hashing is a type of a solution which can be used in almost all situations. Hashing is a technique which uses less key comparisons and searches the element in O(n) time in the worst case and in an average case it will be done in O(1) time. This method generally used the hash functions to map the keys into a table, which is called a hash table.

Collision: It is a situation in which the hash function returns the same hash key for more than one record, it is called as collision. Sometimes when we are going to resolve the collision it may lead to a overflow condition and this overflow and collision condition makes the poor hash function.

Collision resolution technique

1. Chaining: It is a method in which additional field with data i.e. chain is introduced. A chain is maintained at the home bucket. In this when a collision occurs then a linked list is maintained for colliding data.



Example: Let us consider a hash table of size 10 and we apply a hash function of H(key)=key % size of table. Let us take the keys to be inserted are 31,33,77,61. In the above diagram we can see at same bucket 1 there are two records which are maintained by linked list or we can say by chaining method.

2. Linear probing: It is very easy and simple method to resolve or to handle the collision. In this collision can be solved by placing the second record linearly down, whenever the empty place is found. In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Example: Let us consider a hash table of size 10 and hash function is defined as H(key)=key % table size. Consider that following keys are to be inserted that are 56,64,36,71.

0	NULL
1	71
2	NULL
3	NULL
4	64
5	NULL
6	56
7	36
8	NULL
9	NULL

In this diagram we can see that 56 and 36 need to be placed at same bucket but by linear probing technique the records linearly placed downward if place is empty i.e. it can be seen 36 is placed at index 7.

3. Quadratic probing: This is a method in which solving of clustering problem is done. In this method the hash function is defined by the H(key) = (H(key)+ x*x)%table size. Let us consider we have to insert following elements that are:-67, 90,55,17,49.

0	90
1	
2	
3	
4	
5	55
6	
7	67
8	17
9	49

In this we can see if we insert 67, 90, and 55 it can be inserted easily but at case of 17 hash function is used in such a manner that :-(17+0*0)%10=17 (when x=0 it provide the index value 7 only) by making the increment in value of x. let x =1 so (17+1*1)%10=8.in this case bucket 8 is empty hence we will place 17 at index 8.

4. Double hashing: It is a technique in which two hash function are used when there is an occurrence of collision. In this method 1 hash function is simple as same as division method. But for the second hash function there are two important rules which are

1. It must never evaluate to zero.
2. Must sure about the buckets, that they are probed.

The hash functions for this technique are:

H1(key)=key % table size

H2(key)=P-(key mod P)

Where, **p** is a prime number which should be taken smaller than the size of a hash table.

Example: Let us consider we have to insert 67, 90,55,17,49.

0	90
1	17
2	
3	
4	
5	55
6	
7	67
8	
9	49

In this we can see 67, 90 and 55 can be inserted in a hash table by using first hash function but in case of 17 again the bucket is full and in this case we have to use the second hash function which is H2(key)=P-(key mode P) here p is a prime number which should be taken smaller than the hash table so value of p will be the 7.

i.e. H2(17)=7-(17%7)=7-3=4 that means we have to take 4 jumps for placing the 17. Therefore 17 will be placed at index 1.

Q. 8. (b) Write a algorithm for Quick Sort and perform the same sorting on following values. **(6.5)**

75, 12, 23, 58, 11, 94, 6, 8, 13.

Ans. Quick Sort Algorithm :

```
1.  algorithm Quick_Sort(list)
2.  Pre: list 6= fi
3.  Post: the list has been sorted in ascending order
4.  if list.Count = 1 // list already sorted
5.  return list
6.  end if
7.  pivot <- Median_Value(list)
8.  for i <- 0 to list.Count - 1
9.  if list[i] = pivot
10. equal.Insert(list[i])
11. end if
```

```
if list[i] < pivot
less.Insert(list[i])
end if
if list[i] > pivot
greater.Insert(list[i])
end if
end for
return Concatenate(Quick_Sort(less), equal, Quick_Sort(greater))
end Quick_sort
```

75, 12, 23, 58, 11, 94, 6, 8, 13

Quick sort

0	1	2	3	4	5	6	7	8
75	12	23	58	11	94	6	8	13

We choose first element of pivot set Loc = 0,

left = 0, Right = 8

Scan from Right to left since

a[Loc] < a [Right]

Now 13 < 75. They are interchange.

13	12	23	58	11	94	6	8	75
Left Loc								Right Loc

No scan from left to right till a [left] > a [loc]

Now 94 is greater than 75. They are interchange

13	12	23	58	11	75	6	8	94
					Left Loc			Right Loc

Now again scan Right to Left till a [right] <a[Loc]

13	12	23	58	11	8	6	75	94
							Left Loc	Right Loc

Now pivot element occupy its position all the preceding element are shorter than 75 at left side and all element are greater than 75 at right side.

Now the left subarray from pivot element will also arranged in this way until the complete array will not sort.

Q. 9. (a) Write algorithm/Program for binary search. Explain why binary search is better than linear search. (6)

Ans. C programming code for binary search

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

```
int main()
{
  int c, first, last, middle, n, search, array[100];
  printf("Enter number of elements\n");
  scanf("%d",&n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d",&array[c]);
  printf("Enter value to find\n");
  scanf("%d", &search);
  first = 0;
  last = n - 1;
  middle = (first+last)/2;
  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;
    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);
  return 0;
}
```

Binary search is more efficient than linear search; it has a time complexity of **O(log n)** while the complexity of linear search is O(n). The list of data must be in a sorted order for it to work.

A binary search works by finding the middle element of a sorted array and comparing it to your target element. Based on what you find, you either use the left or right half of the array and update the start and ending indexes until you either do or do not find the element.

Q. 9. (b) What is difference between internal and external sorting? Write any one algorithm for internal sorting and one for external sorting. (6.5)

Ans. Internal & External Sorting :

Internal Sorting: If all the data that is to be sorted can be adjusted at a time in the main memory, the internal sorting method is being performed.

External Sorting: When the data that is to be sorted cannot be accommodated in the memory at the same time and some has to be kept in auxiliary memory such as hard disk, floppy disk, magnetic tapes etc, then external sorting methods are performed.

Internal Sort Algorithm (Bubble Sort)

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for
  return list
end BubbleSort
```

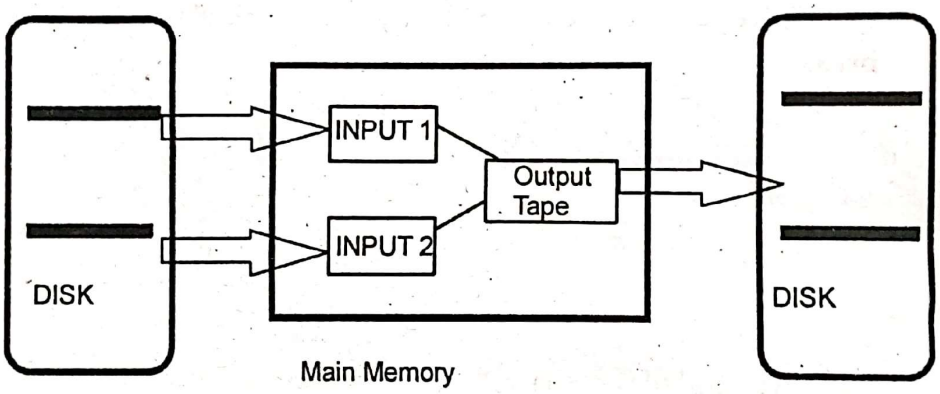
External Sort Algorithm (Two-Way Merge Sort)

Step 1. Divide the elements into the blocks of size M. Sort each block and then write on disk.

Step 2. Merge two runs

1. Read first value on every two runs.
2. Then compare it and sort it.
3. Write the sorted record on the output tape.

Step 3. Repeat the step 2 and get longer and longer runs on alternates tapes. Finally, at last, we will get a single sorted list.



Two-Way Merge