**Shri Shankaracharya Institute of Professional Management & Technology, Raipur**

**February-2022- Class Test-2**

**Date:** 05/02/2022

**Student Name:** V OM SAI NAGESHWAR SHARMA

**Roll No.:**

| 3 | 0 | 3 | 3 | 0 | 2 | 2 | 2 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Enrollment No.:**

| B | J | 4 | 5 | 9 | 9 |
|---|---|---|---|---|---|

**Course:** B.Tech **Semester:** 3rd

**Branch:** Computer Science And Engineering

**Subject Name:** Data Structures And algorithms

**Subject Code:**

| B | 0 | 2 | 2 | 3 | 1 | 2 | ( | 0 | 2 | 2 | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Mobile No.:** 8602727389

**Email id:** om.sharma@ssipmt.com

**Signature:**

# PART - I

Q1>

Ans => **Merge Sort**: The elements are split into 2 sub-arrays $n/2$ therefore $\log n$, at last all elements are merged to make it 'n' element size.

$\therefore$ n combining these two we get complexity $O(n \times \log n)$

Worst case → $O(N * \log N)$ Average case → $O(N * \log N)$

Best case → $O(N * \log N)$

**Insertion sort**: It uses 2 insertions to keep track of array and key

$\therefore N \times N$

=> worst case $O(N^2)$

Average case → $O(N^2)$

Best case → $O(N)$

Q4>

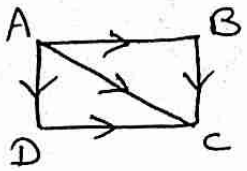
Ans =>

| Binary Tree | Binary Search Tree |
| --- | --- |
| 1.) Binary Tree is a non-linear data structure where each node can have atmost two child nodes. | 1.) Binary Search Tree is a node based binary tree which further has right and left subtree that too are binary search tree. |
| 2.) Binary Tree is unordered hence slower in process of insertion, deletion, and searching. | 2.) Insertion, deletion, searching of an element is faster in Binary Search Tree than Binary Tree due to the ordered characteristics. |
| 3.) In Binary Tree there is no ordering in terms of how the nodes are arranged. | 3.) In Binary Search Tree the left subtree has elements less than the nodes element and the right subtree has elements greater than the nodes element. |

Q2>

Ans => A directed graph, also called a digraph, is a graph in which the edges have a direction. This is usually indicated with an arrow on the edge; more formally, if v and w are vertices, an edge is an unordered pair {v,w}, while a directed edge, called an arc, is an ordered pair (v,w) or (w,v). Each edge of a graph has an associated numerical value, called a weight. Usually, the edge weights are non-negative integers. Weighted graphs may be either directed or undirected.

Example -



The adjacency matrix is a matrix consisting of $n \times n$ where rows and columns are labeled by graph vertices, with a 1 or 0 in position according to whether the vertices are adjacent or not.

Ex:

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |

Q3)

Ans=> The single source shortest paths are based on a technique known as relaxation, a method that repeatedly decreases an upper bound on the actual shortest path weight of each vertex until the upper bound equivalent the shortest path weight.

Relaxation method (u v w)

u ⟶ source vertex

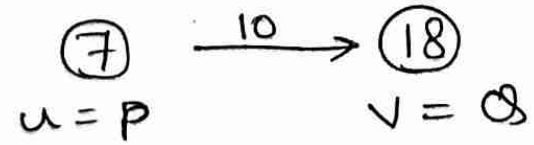
v ⟶ destination vertex

w ⟶ weight.

step① :- If $d[v] > d[u] + w(u,v)$

step② :- then $d[v] \leftarrow d[u] + w(u,v)$

Ex :-




d[P] = key value of vertex P = 7

d[Q] = key value of vertex Q = 18

Applying Relaxation method, (P.S)

$18 > 7 + 10$

$18 > 17$

The above condition is ~~the~~ true then, $d[v] = 17$

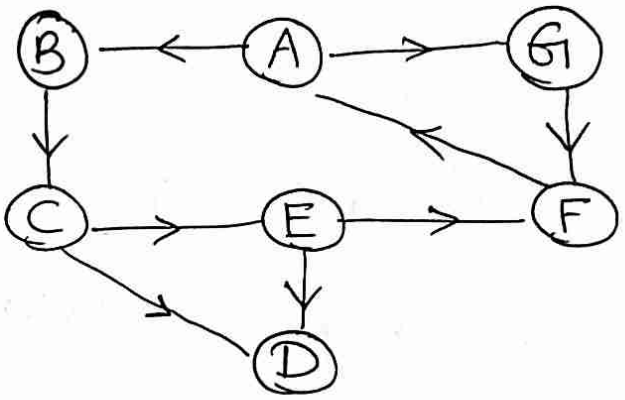
# Part - II

Q3>

Ans => **Graph Traversal Algorithm:**

It is a technique used for a searching vertex in a graph. There are two technique use for vertex traversal :-

| BFS | DFS |
|---|---|
| ① BFS stands for Breadth first Search. | ① DFS stands for Depth first search. |
| ② It uses Queue data structure for finding the shortest path. | ② It uses stack data structure for finding the shortest path. |
| ③ In BFS, we reach a vertex with minimum number of edges from a source vertex. | ③ In DFS, we might traverse through more edges to reach a destination vertex from a source |




Source vertex (assuming) is A

According to the given problem the source matrix is defined as A and we push the vertex A into the stack.

(i) Pop vertex A and push adjacent vertices of A into stack :-

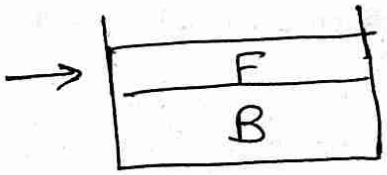
Pop : (A)

Stack : B, G1


(ii) Pop vertex G1 and push adjecnt vertices of G1 into stack:-

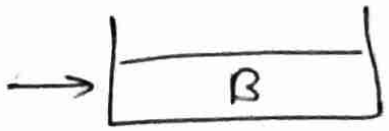
Pop : (A) (G1)

Stack : B, F



(iii) Pop vertex F and push adjacent vertices of F into stack :-

Pop : (A) (G) (F)




Stack: B

(iv) Pop vertex B and push adjacent vertices of E into stack :-

Pop :- (A) (G) (F) (B) (C) (D) (E)

stack :- $\phi$

So, Reachable vertex from SA are

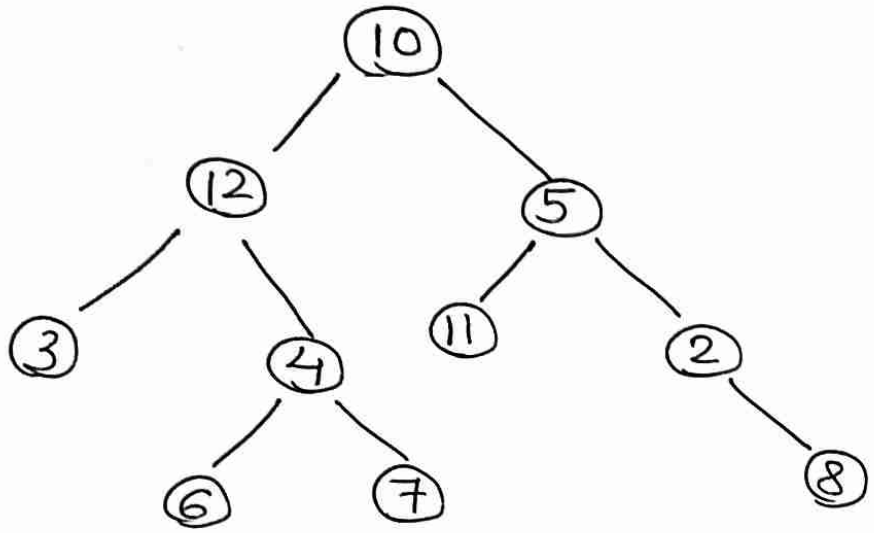
A, G, F, B, C, D, E

Q2>

Ans => Preorder - 10 12 03 04 06 07 | 05 11 02 08

Postorder - 03 06 07 04 12 | 11 08 02 05 10

Left | Right




Algorithm to create a Binary tree from given preorder and postorder.

The first node in the preorder and last node in the post order is Node.

Find out the successor of Root Node in pre-order i.e. B and find out predecessor of root node in post order i.e. "e" and Name them $N_1$ and $N_2$.

(a) if $n_1 == n_2$ then this node can be either be left child or right child both and because this reason we can't create a unique binary tree from given preorder or post order.

(b) if $(n_1 != n_2)$ then set $n_1$ as left child and $N_2$ as Right child.

(c) Find the position of $N_2$ and $N_1$ in preorder and postorder respectively. Now consider the two set of preorder and postorder sequence of left & Right softree of the root.

(d) Repeat these entire stop unit tree is complete.

Q1)

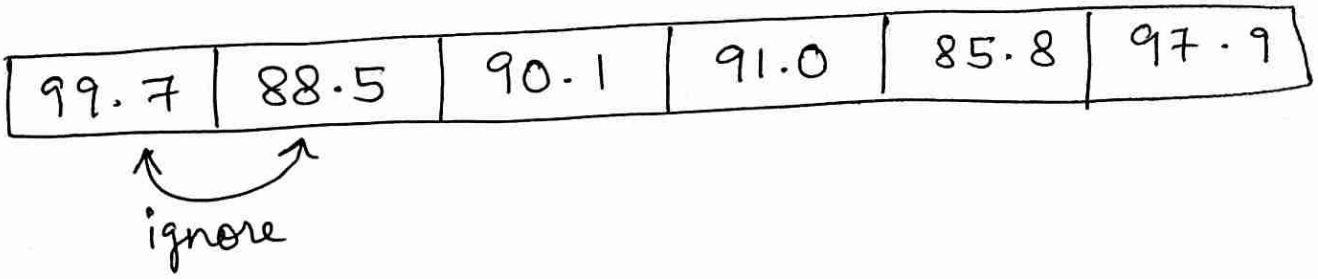
Ans -> ★ Bubble sort is a simple sorting algorithm. This sorting algorithm is comparision based algorithm in which each pair of adjacent elements are compared & the elements are swapped if they are not in correct order.

★ This algorithm is not suitable for large data set.

★ The worst time complexity of this algorithm in $O(n^2)$ where in the no. of items.

| z | y | x | w | v | u |
|---|---|---|---|---|---|
| 99.7 | 88.5 | 90.1 | 91.0 | 85.8 | 97.9 |
| 0 | 1 | 2 | 3 | 4 | 5 |

(1)




99.7 88.5 90.1 91.0 85.8 97.9

Swap

99.7 90.1 88.5 91.0 85.8 97.9

Swap

99.7 90.1 91.0 88.5 85.8 97.9

ignore

99.7 90.1 91.0 88.5 85.8 97.9

swap.

99.7 90.1 91.0 88.5 97.9 85.8

↳ Smallest strike, rate

unsorted

(2) 99.7 90.1 91.0 88.5 97.9 85.8

ignore

99.7 90.1 91.0 88.5 97.9 85.8

swap.

99.7 91.0 90.1 88.5 97.9 85.8

ignore

99.7 91.0 90.1 88.5 97.9 85.8

swap

ignore

99.7 91.0 90.1 97.9 88.5 85.8

unsorted

2 elements are sorted in decreasing order.

(3) 99.7 91.0 90.1 97.9 88.5 85.8

ignore

99.7 91.0 90.1 97.9 88.5 85.8

ignore

99.7 91.0 90.1 97.9 88.5 85.8

swap

99.7 91.0 97.9 90.1 88.5 85.8

unsorted

sorted

Pg. No. → 14

(4) 99.7 91.0 97.9 90.1 88.5 85.8

ignore

99.7 91.0 97.9 90.1 88.5 85.8

Swap

no change

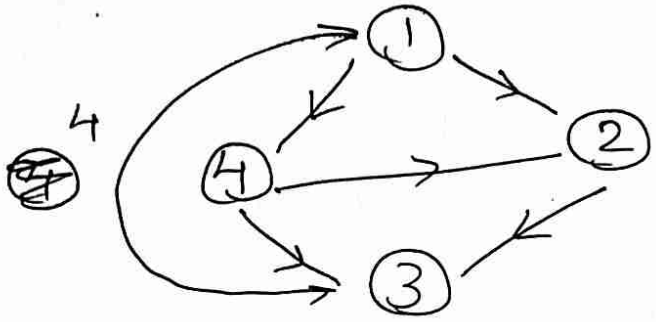
99.7 97.9 91.0 90.1 88.5 85.8

No change NO change

sorted array is decreasing order using bubble sort.

Q4)

Ans => Flogd Washall Algorithm -
It is used to solve all pair shortest path problem.




Step 1 :- ✱ remove all the self loops and parallel edges from the graph.

✱ There are neither self loops or parallel edges on the given graph.

Step 2 :- ✱ Write the initial distance matrix.

✱ for diagnal elements (representing self-loops), distance value = 0

✱ for vertices having a ~~value~~ direct edge in between them, distance value = weight of edge.

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

Step 3 :-

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$D_4$ represents the shortest path distance between every pair of vertices.