1.a

Ans. 2 marks for this differentiation

Feature	BFS	DFS
Exploration	Level-by-level traversal, guaranteed shortest path (unweighted)	Deepest path first traversal, no guarantee of shortest path
Data Structure	Queue (FIFO)	Stack (LIFO)
Applications	Shortest paths, minimum spanning trees, bipartiteness, level-order tree traversal	Topological sorting, cycle detection, connected components
Visualization	Expands outward like ripples in water	Explores like navigating a maze

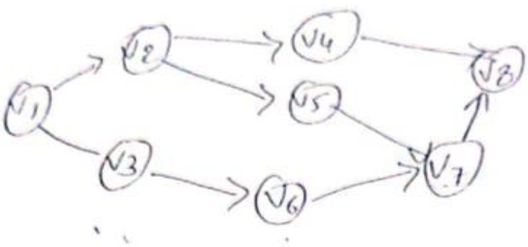
NOTE - Students may take any example of graph and its traversal. For the reference 1 example is below:

1.5 marks for DFS

1.5 marks for BFS

0.5 each for definition and 1 ,mark each for example.

{DFS}:— It is the method to traverse graph by level. It uses stack



Stack	Node traversal
V1	-
V2, V3	V1
V2, V6	V3
V2, V7	V6
V2, V8	V7
V2, V4	V8
V2, V5	V4
V2	V5
	V2

V1, V3, V6, V7, V8, V4, V5, V2

↳ V1, V2, V4, V8, V5, V7, V3, V6

BFS → V1, V2, V3, V4, V5, V6, V7, V8

↳ V1, V3, V2, V6, V5, V4, V7, V8

. 3 states are used in traversal

-> **Ready** => Node doesn't reach to queue.

-> **Waiting** => Node is inserted into the queue but proc
eater.

-> **Processed** => Node is removed from the queue & pr

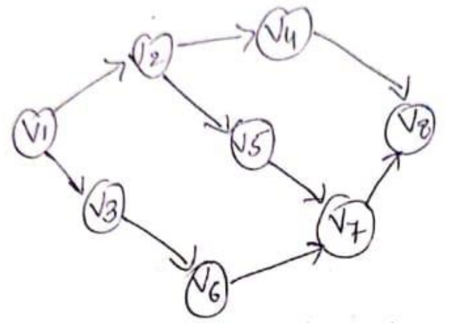
(1) {BFS} :-

-> Start with a node.

-> Progresses by analysing its adjacent node.

-> We use queue in BFS

{Eg}



Traversal Sequence :-

V1, V2, V3, V4, V5, V6, V8, V7 -> 1 less move.

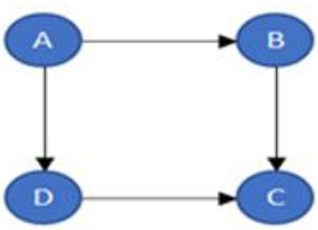
Queue	Node traversal
V_1	
V_2, V_3	V_1
V_3, V_4, V_5	V_2
V_4, V_5, V_6	V_3
V_5, V_6, V_8	V_4
V_6, V_8, V_7	V_5
V_8, V_7	V_6
	V_8
	V_7

1.b

Adjacency Matrix:

- An adjacency matrix is a way to represent a graph as a 2D array (matrix). In an adjacency matrix, rows and columns represent vertices of the graph, and each cell indicates whether there is an edge between the corresponding vertices. If there is an edge between vertex i and vertex j, then the cell at row i and column j will have a value indicating the weight of the edge (if the graph is weighted) or simply 1 (if the graph is unweighted).
- If there is no edge between vertex i and vertex j, the cell will contain a special value (often denoted as 0 or ∞ for unweighted graphs). ....

0.5



$$
\begin{array}{c c}
 & \begin{array}{cccc} A & B & C & D \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\end{array}
$$

1M

ii) Adjacency List

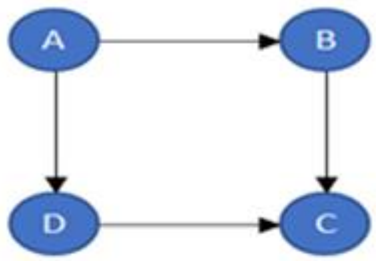
An adjacency list is another way to represent a graph, where each vertex in the graph maintains a list of its adjacent vertices. For each vertex in the graph, there is a list (array, linked list, etc.) containing references or indices to the vertices that are adjacent to it. If the graph is weighted, each entry in the adjacency list can store both the identifier of the adjacent vertex and the weight of the edge.

If the graph is unweighted, the adjacency list may only need to store the identifiers of adjacent vertices.

0.5

Example:

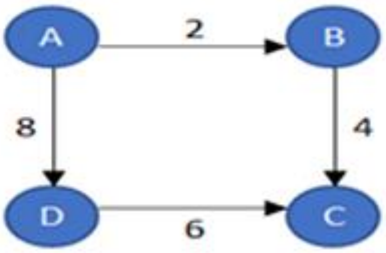
Directed unweighted weigh Graph



$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

1

Example Weighted graph



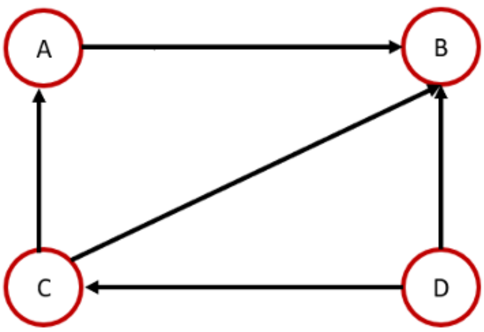
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 2 & 0 & 8 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 \end{bmatrix} \end{array}$$

iii) Transitive Closure

Transitive Closure in a graph shows the path between the nodes. In other words, if there exists a path between node x and node y, then there should be a corresponding edge(s) between these in Graph. The final transitive Closure Ti j (k) of a graph is a Boolean matrix, where if there is a path between two vertices, it is indicated by 1 in the Matrix, otherwise 0.

1

Example



Transitive Closure:

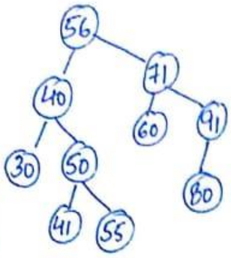
	1	2	3	4
1	0	1	0	0
2	0	0	0	0
3	1	1	0	0
4	1	1	1	0

2

2.a

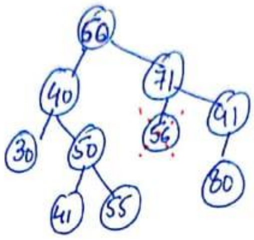
Keys to delete : 56, 71, 40

Inorder Traversal: 30, 41, 50, 55, 40, 56, 60, 80, 91, 71



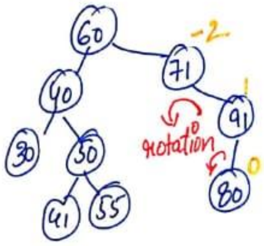
delete 56 →

Replace with its inorder successor i.e. 60.

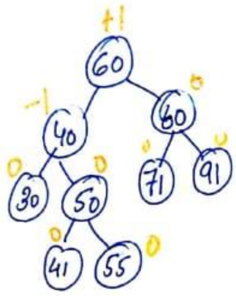


Now, 56 is a leaf node then delete it directly.

⇓ Tree after deletion

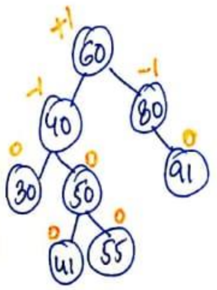


⇐



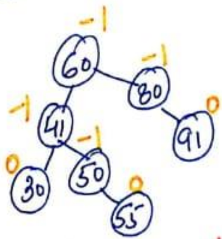
— 1 Mark

⇓ delete 71
(71 is a leaf node so delete it directly)



(Tree after deletion of 71)

delete ⇒ 40
replace it with 41

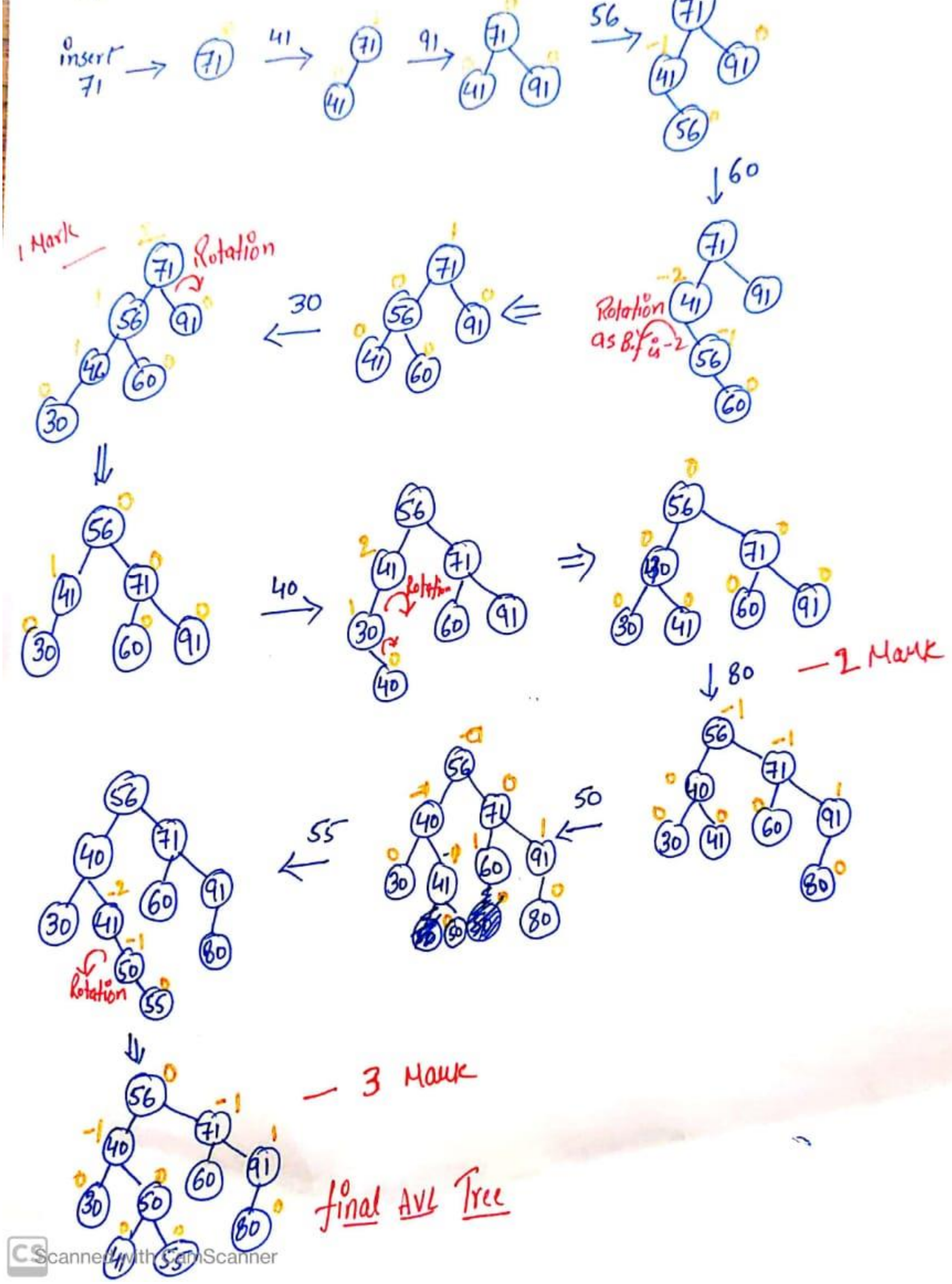


— 2 Mark

final AVL Tree

AVL Tree :-

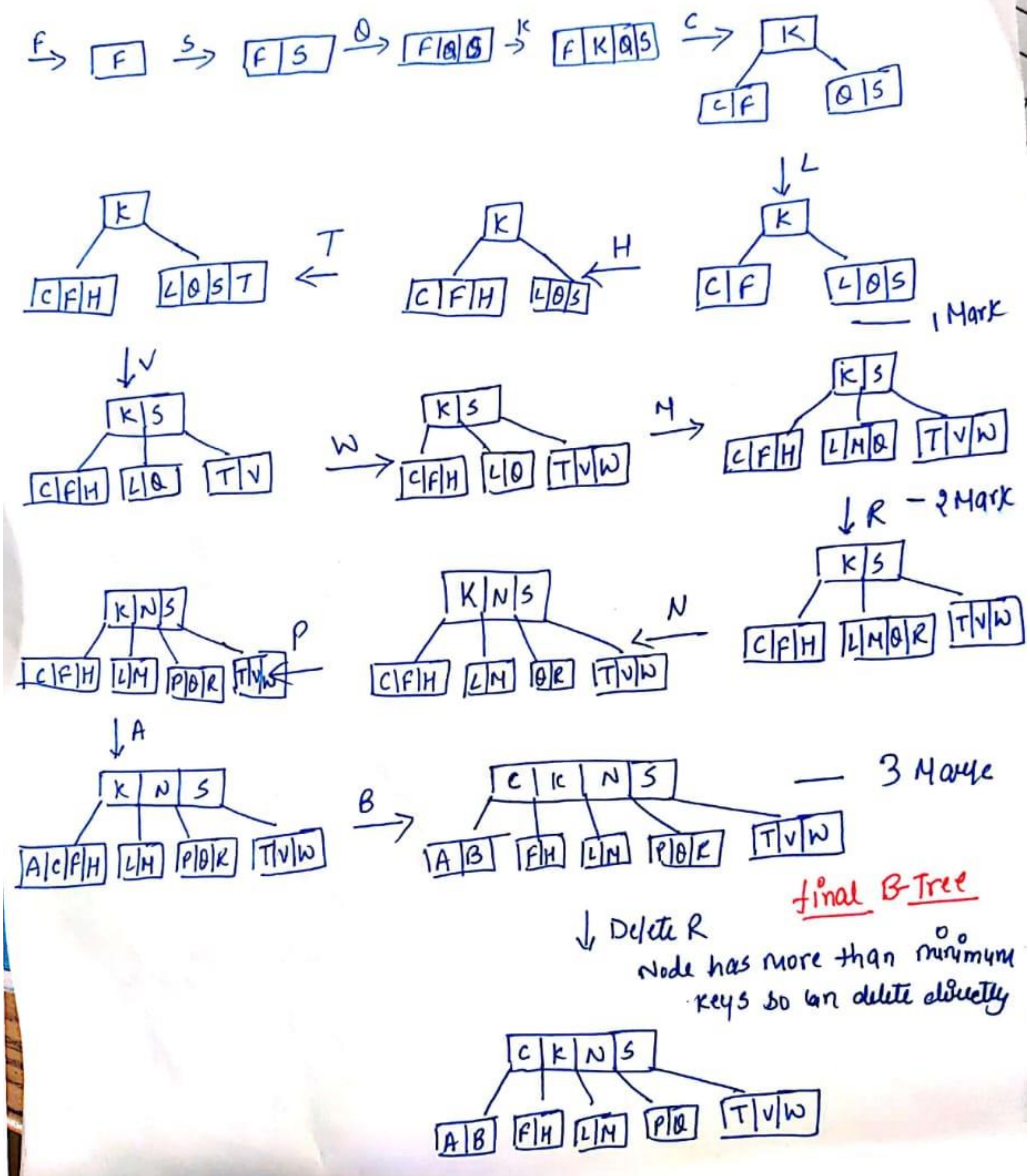
Keys : 71 , 41 , 91 , 56 , 60 , 30 , 40 , 80 , 50 , 55



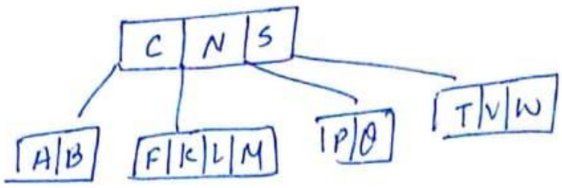
2.b

B-Tree Order-5

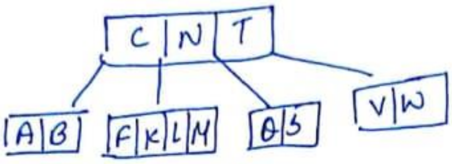
Keys : F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B



Delete H

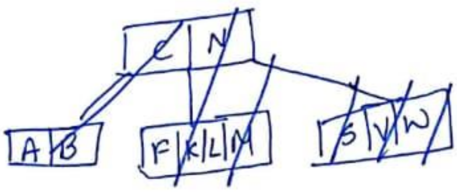


Delete P



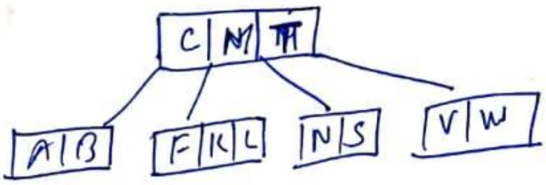
— 1 Mark

~~Delete Q~~



~~Final B.Tree~~

Delete Q



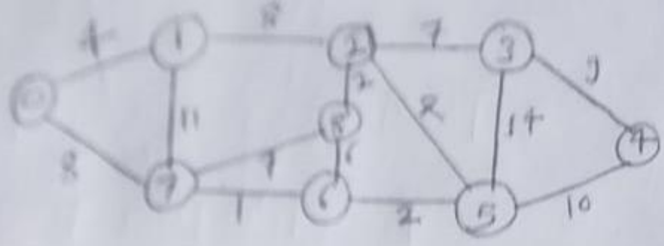
— 2 Mark

3.a

Q.3 (a)

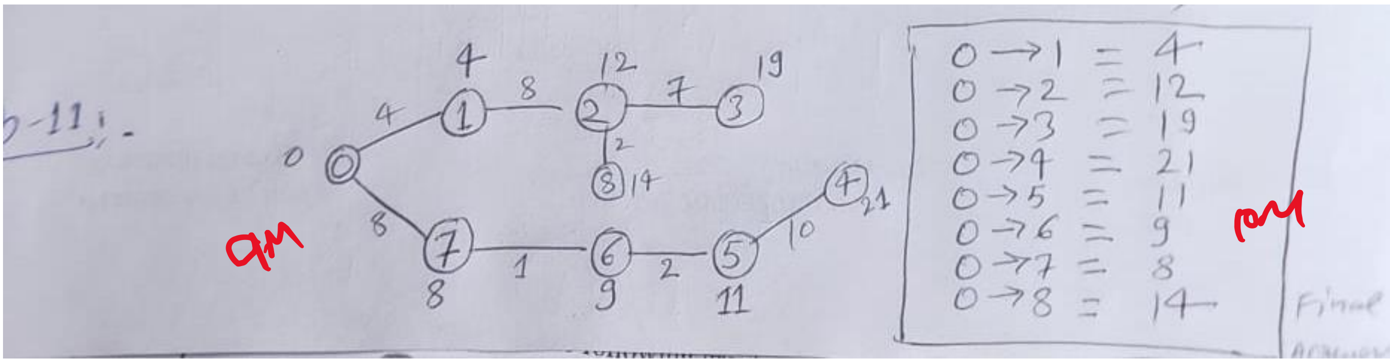
Solution :-

Given graph



Step-1 Create priority queue with all vertex

		Extracted			Priority queue (min-heap)								
		π	vertex	dist	0	1	2	3	4	5	6	7	8
Step-2		-	0	0	0	∞ Nil	∞ Nil	∞ Nil	∞ Nil	∞ Nil	∞ Nil	∞ Nil	∞ Nil
Step-3	1M	0	1	4	X	4 0	∞ Nil	∞ Nil	∞ Nil	∞ Nil	∞ Nil	8 0	∞ Nil
Step-4	2M	0	7	8	X	X	12 1	∞ Nil	∞ Nil	∞ Nil	∞ Nil	8 0	∞ Nil
Step-5 :-	3M	7	6	9	X	X	12 1	∞ Nil	∞ Nil	∞ Nil	9 7	X	15 7
Step-6 :-	4M	6	5	11	X	X	12 1	∞ Nil	∞ Nil	11 6	X	X	15 7
Step-7	5M	1	2	12	X	X	12 1	25 5	21 5	X	X	X	15 7
Step-8	6M	2	8	14	X	X	X	19 2	21 5	X	X	X	14 2
Step-9	7M	5	4	21	X	X	X	19 2	21 5	X	X	X	X
Step-10	8M	2	3	19	X	X	X	19 2	X	X	X	X	X
					X	X	X	X	X	X	X	X	X



3.b

ALGORITHM HeapSort(A[ ], N)

BEGIN:

BuildMaxHeap(A, N)

FOR j = N to 2 STEP – 1 DO

Exchange(A[j], A[1])

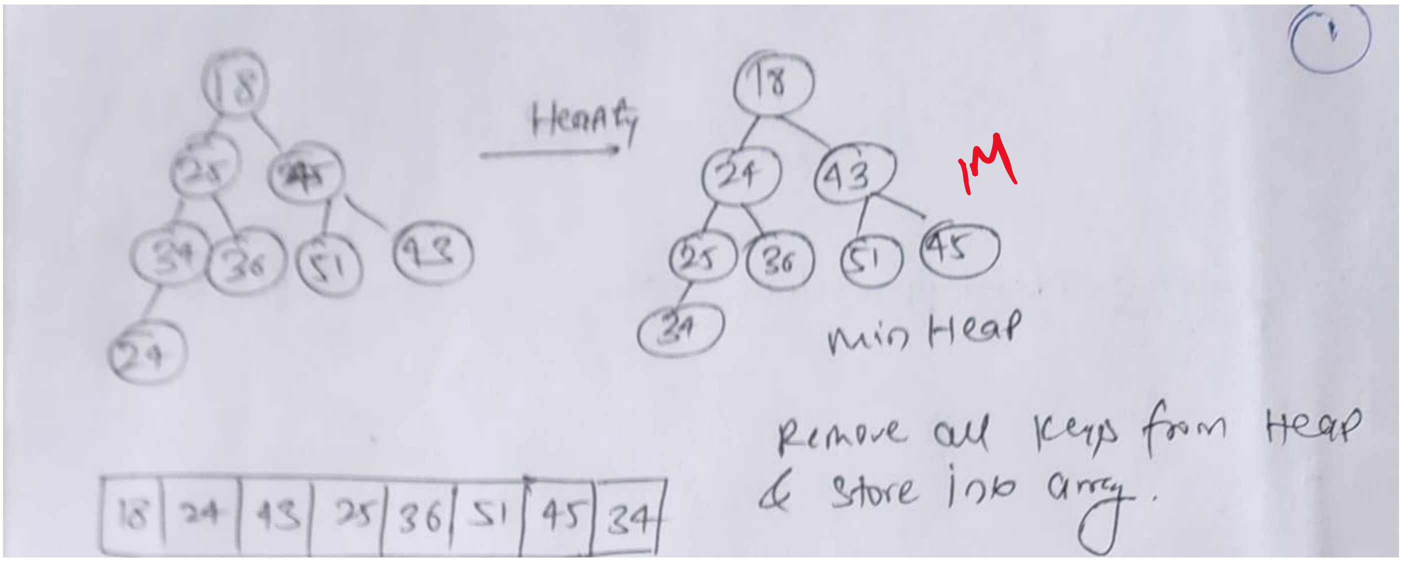
MaxHeapify(A, 1, j-1)

END;

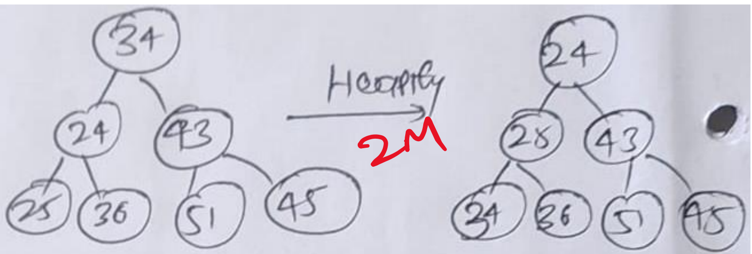
3M

Build min Heap with given keys and remove all keys to sort in descending order

Keys : 18, 25, 45, 34, 36, 51, 43, 24

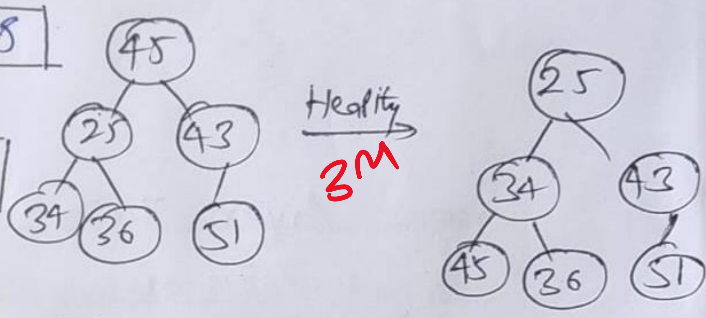


34	24	43	25	36	51	45	18

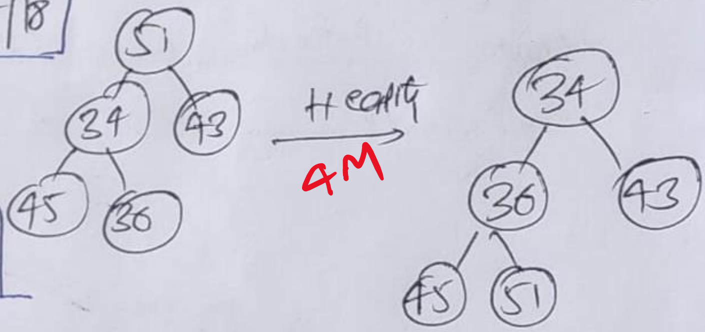


24	25	43	34	36	51	45	18

45	25	43	34	36	51	24	18

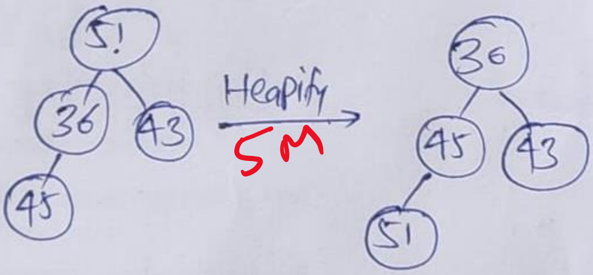


25	34	43	45	36	51	24	18

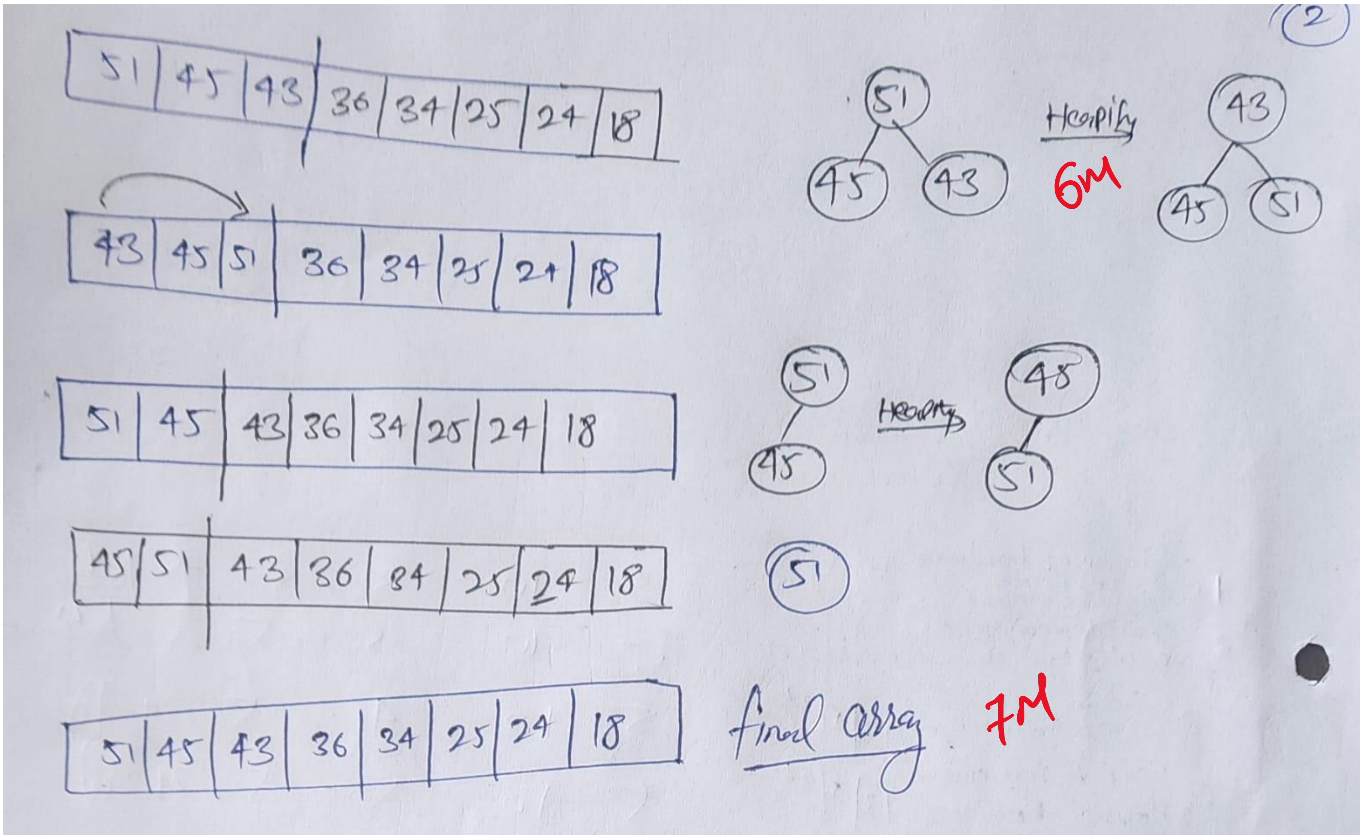


34	36	43	45	51	25	24	18

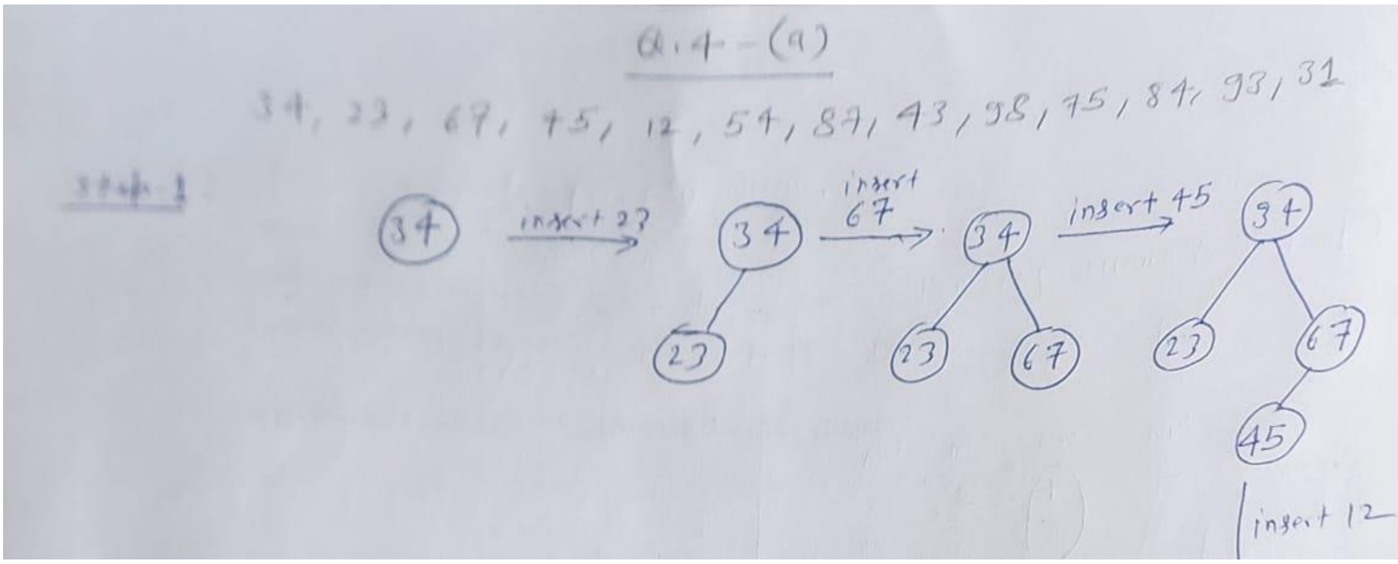
51	36	43	45	34	25	24	18

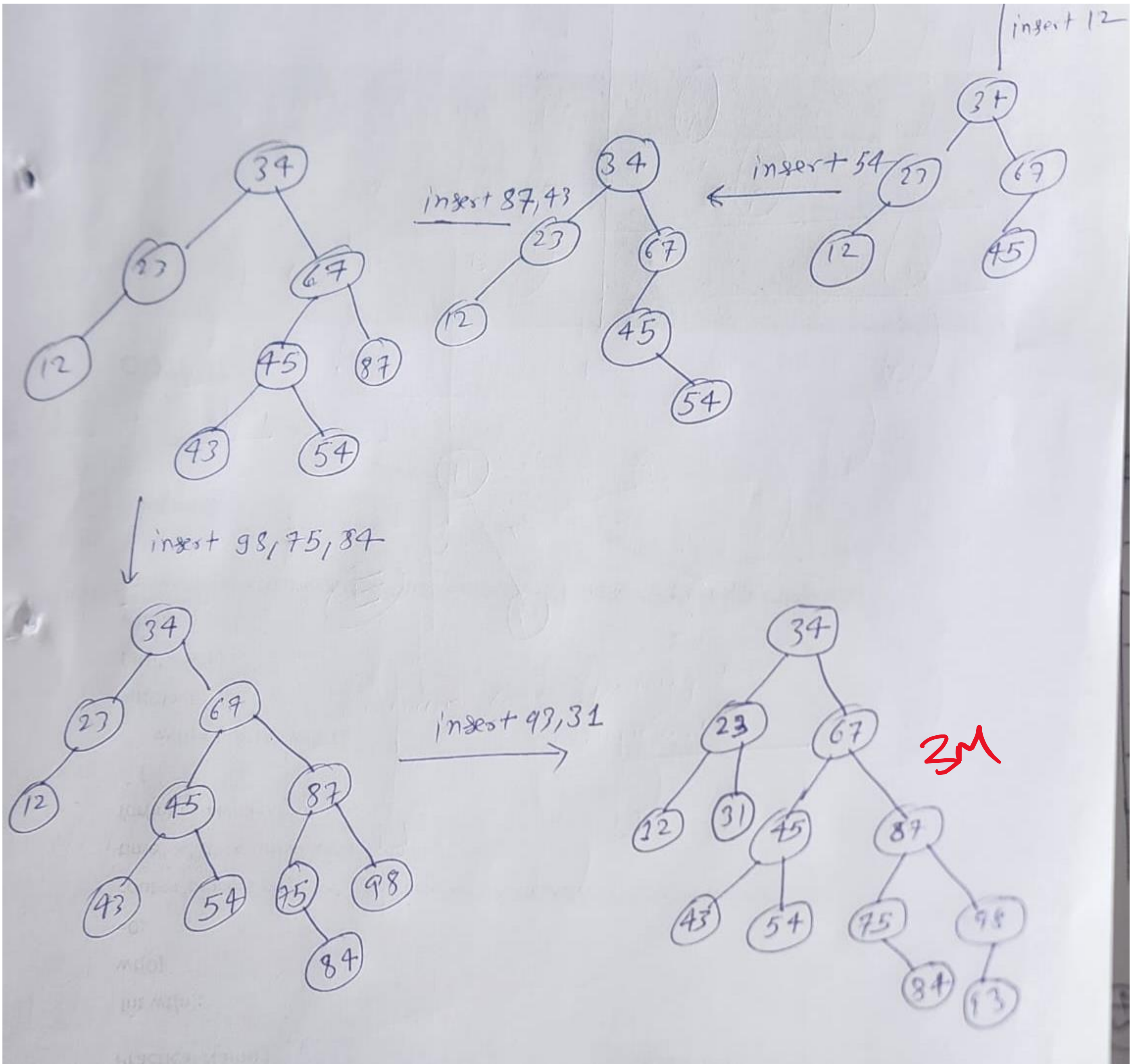


36	45	43	51	34	25	24	18


2
51 | 45 | 43 | 36 | 34 | 25 | 24 | 18
Heapify
6M
43 | 45 | 51 | 36 | 34 | 25 | 24 | 18
51 | 45 | 43 | 36 | 34 | 25 | 24 | 18
Heapify
45 | 51 | 43 | 36 | 34 | 25 | 24 | 18
51 | 45 | 43 | 36 | 34 | 25 | 24 | 18
final array
7M

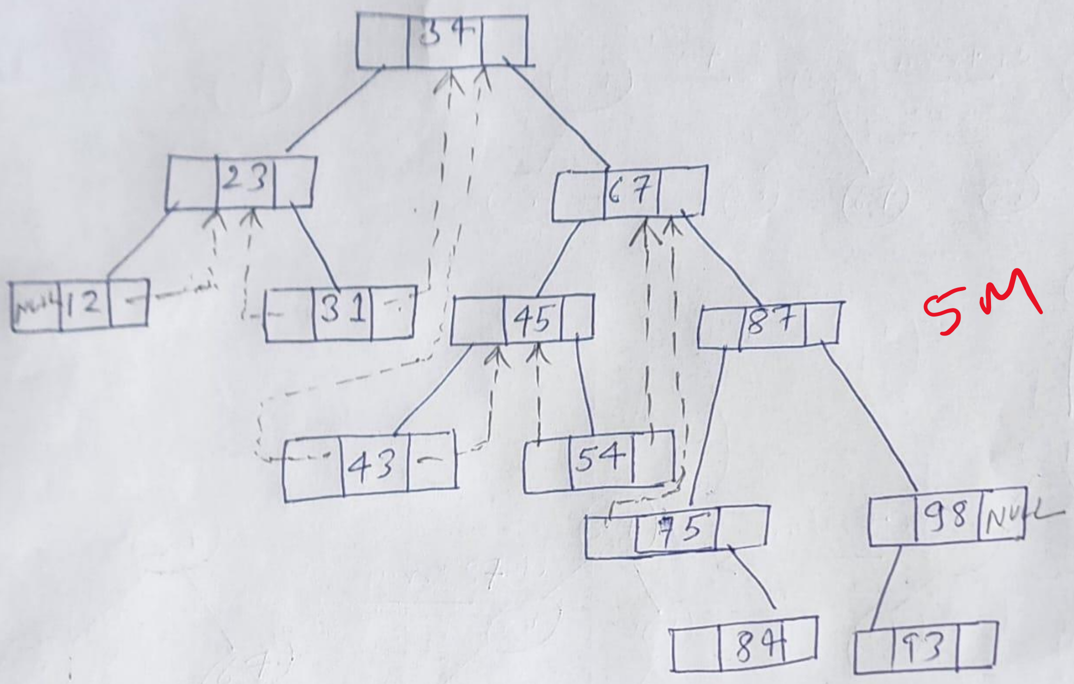
4.a


Q.4-(a)
34, 23, 67, 45, 12, 54, 87, 43, 98, 75, 84, 93, 31
Step 1
insert 23
insert 67
insert 45
insert 12



Find Binary Search tree

Threaded Binary Tree



5M

4.b

Q.4.(b)

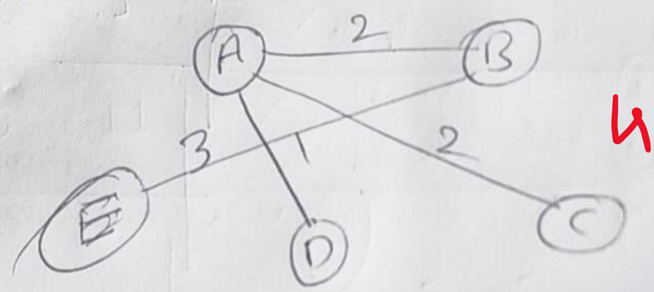
Solution:-

Step 1.:- Create priority queue.

A as Source Node

	Extracted			Priority queue (min-heap)				
	π	Vertex	cost	A	B	C	D	E
Step.2	–	A	0	0	∞ Nil	∞ Nil	∞ Nil	∞ Nil
Step.3	A	D	1		2 A	2 A	1 A	4 A
Step 4	A	1M B	2		2 A	2 A	X	4 A
Step-5	A	C	2	X	X	2 A	X	3 B
Step-6	B	2M E	3	X	X	X	X	3 B
				X	X	X	X	X

Step-7 :-



4M

minimum cost spanning tree
minimum cost = 8

5M

5.a

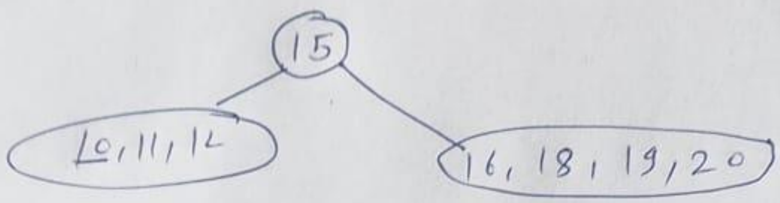
Pre order : 15, 10, 12, 11, 20, 18, 16, 19

As given in question that tree is BST so inorder traversal will produce element in sorted order

Hence inorder : 10, 11, 12, 15, 16, 18, 19, 20

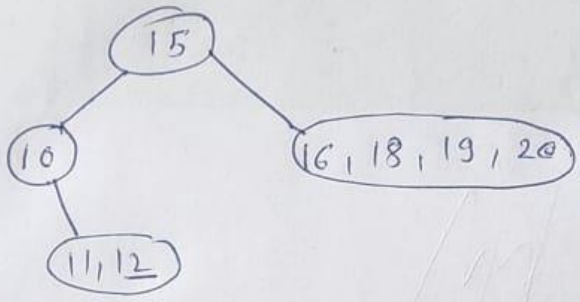
Since preorder follows: Root, left subtree, right subtree
So root is 15

Step-1



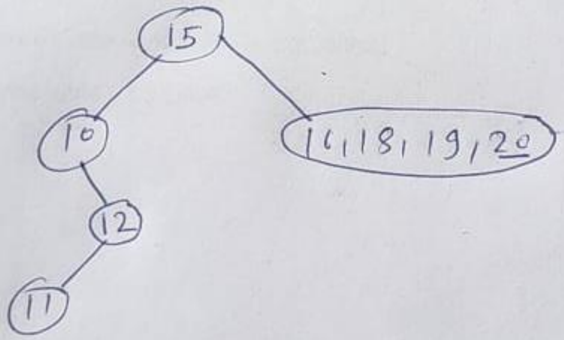
10 is root according to preorder

Step-2 :

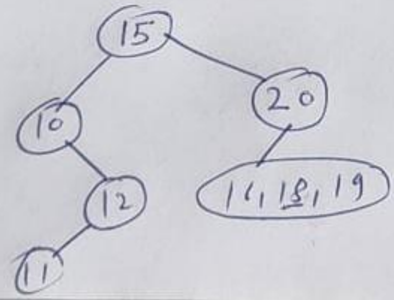


12 is root according to preorder

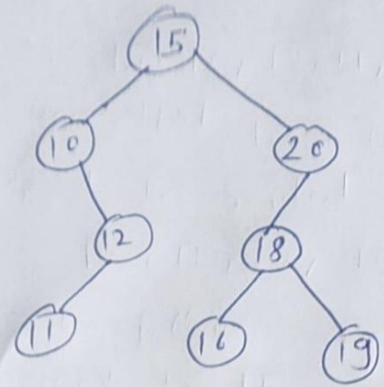
Step-3:



Step-4: 20 is root according to pre.order



Step-5 18 is root according to preorder



— 3

Final Binary Search Tree

Postorder Traversal : left subtree, right subtree, root

postorder: 11, 12, 10, 16, 19, 18, 20, 15 — 2

5.b

1: abef->c or g should be covered 1M

2: abefcgd correct 1M

3: adgebcf correct 1M

5M

4: adbc->e or f should be covered 1M

6.a

```
ALGORITHM HuffmanTree(A[ ], N)
// where A is the information about the character & their frequencies, and N is the total
number of character appearing in the text file.
BEGIN:
        Initialize(PQ) //Initialize a PriorityQueue PQ, which contains N elements in A
        FOR I = 1 to N DO
                Z=MakeNode(A[i])
                PQInsert(PQ, Z)

        FOR I = 1 to N-1 DO
                X= PQDelete(PQ)
                Y= PQDelete(PQ)
                Z= MakeNode( )
                Z→Data = X→Data + Y→Data
                Z→Left = X
                Z→Right = Y
                PQInsert(PQ, Z)
END;
```

3M

Message "MAHARASHTRA"

Step 1 Create frequency table for all alphabets in given message.

Character	Count
M	1
A	4
H	2
R	2
S	1
T	1

4M

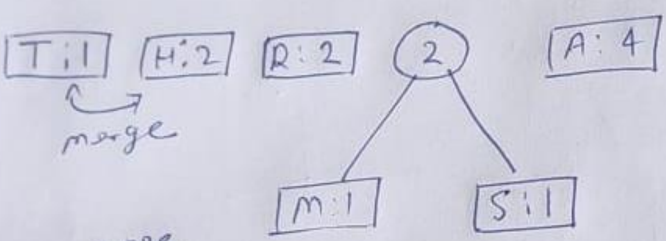
Step-2 → arrange in ascending order of their frequencies.

merge

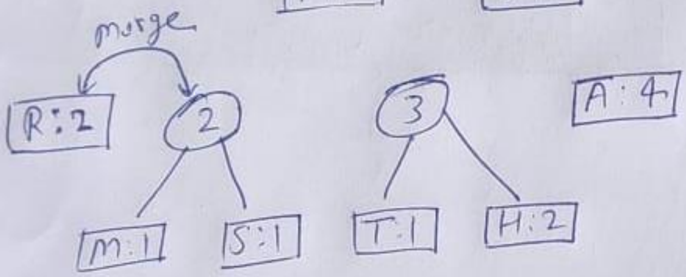
$\boxed{M:1}$ $\boxed{S:1}$ $\boxed{T:1}$ $\boxed{H:2}$ $\boxed{R:2}$ $\boxed{A:4}$

6M

Step-2

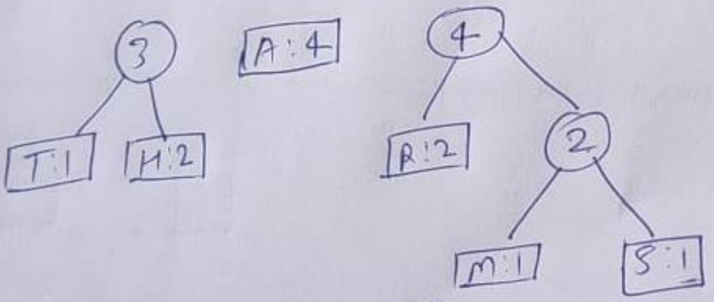


Step-3:

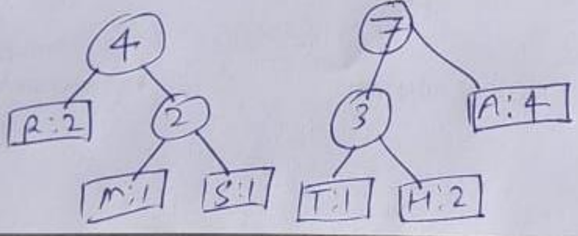


7M

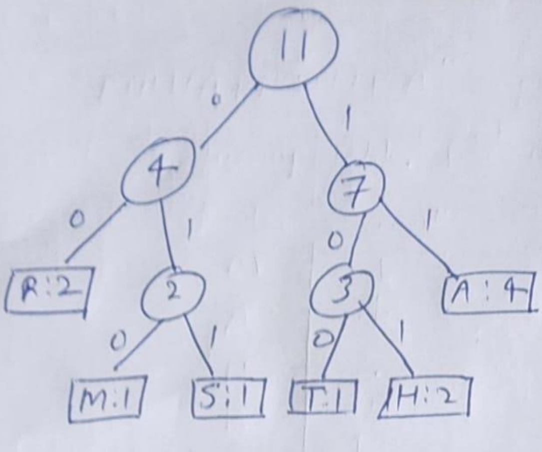
Step-4:-



Step-5:-



Step-6:-



8M

Final tree

M — 010
A — 11
H — 101
R — 00
S — 011
T — 100

9M

encodede message: 010 11 101 1100 11 011 101 100 00 11
10M M A H A R A S H T R A

6.b

Sparse Matrix - In computer science, a sparse matrix in the data structure is a matrix that has a large number of zero values compared to its total number of elements. Storing these matrices using a dense array representation would be inefficient in terms of both memory and computation time. To overcome this problem, sparse matrices can be represented using a more efficient data structure that only stores the non-zero elements. This can save memory and computation time for certain types of operations, such as matrix multiplication and inversion, and is an important concept in data structures and algorithms. 1M

Representation of Sparse Matrix in Data Structure

Sparse matrix in data structure can be represented in two ways:

1. Array representation
2. Linked list representation

Array Representation of Sparse Matrix in Data Structure 2M

Storing a sparse matrix in a 2D array can result in a lot of memory wastage because zero elements in the matrix do not carry any useful information, yet they occupy memory space. To overcome this limitation and optimize memory usage, we can represent a sparse matrix by only storing its non-zero elements. This approach reduces both traversal time and storage space requirements. By storing only the non-zero elements, we can achieve a more efficient representation of sparse matrices in data structures.

In the 2D array representation of a sparse matrix, there are typically three fields used to store the relevant information.
The given below are fields of the array representation of the sparse matrix:

- **Row index field:** This field stores the row indices of non-zero elements in the matrix.
- **Column index field:** This field stores the column indices of non-zero elements in the matrix.
- **Value field:** This field stores the actual values of the non-zero elements in the matrix.

The array representation of the sparse matrix in the data structure program converts a given matrix into its array representation where the non-zero elements of the matrix are stored in separate arrays for row, column, and value. It first counts the number of non-zero elements in the matrix, then initializes the arrays and stores the non-zero elements in them. Finally, it displays the array representation of the matrix. This program can reduce the storage space required for large matrices with many zero elements.

Linked List Representation of Sparse Matrix in Data Structure 3M

The sparse matrix in data structure can be represented using a linked list data structure, which offers the advantage of lower complexity when inserting or deleting a node compared to an array.

The linked list representation differs from the array representation in that each node has four fields, which are:

- **Row:** It indicates the index of the row where the non-zero element is located.
- **Column:** It indicates the index of the column where the non-zero element is located.
- **Value:** It represents the value of the non-zero element located at the index (row, column).
- **Pointer to Next node:** It stores the memory address of the next node in the linked list.

In the implementation of a linked list representation of a sparse matrix, the Node class represents a node in the linked list for a particular row. The SparseMatrix class contains a dynamic array of pointers to nodes, where each pointer points to the head of the linked list for a particular row. The insert function inserts a new element into the matrix by creating a new Node and inserting it into the appropriate linked list, sorted by column index. The display function prints out the entire matrix by iterating over each row and column and checking the linked list for each row to see if there is a corresponding element

Q.6 (b)

(i) - Sparsity = No of zero values / Total count of el

= 25/36

$\boxed{\text{Sparsity} = 0.69}$

0.69 > 0.5 hence sparse matrix

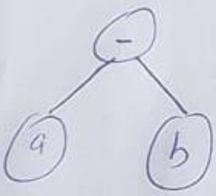
5M

(ii) - Expression tree: It is a special kind of tree which is used to represent expressions. In expression tree operands are leaf nodes and operators are interna nodes.

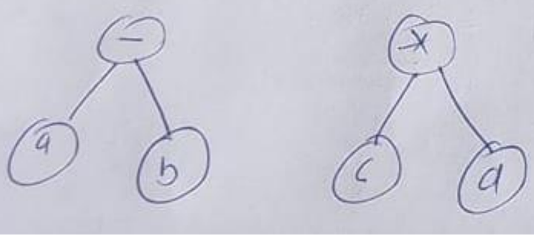
2M

expression: $(a-b)/((c*d)+e)$

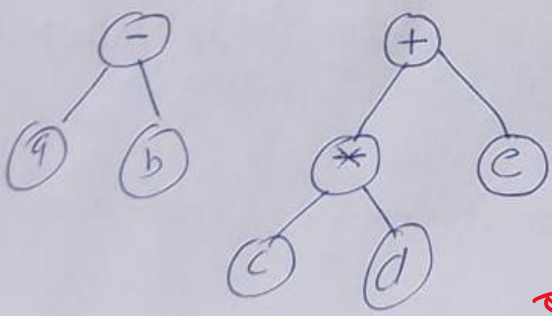
step-1:-



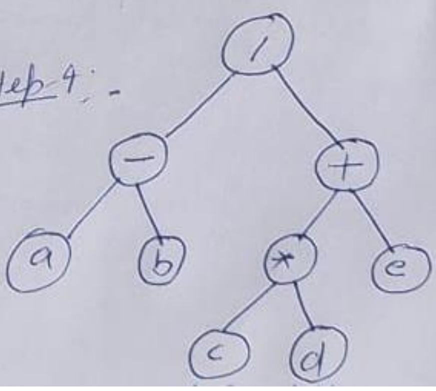
step-2:-



Step-3:-



Step-4:-



SM