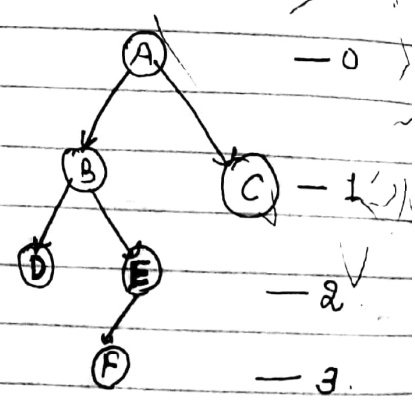
3/10/18

TREE

Non-linear data structure used to represent hierarchical form.

1. Root :- Every tree has unique root.
2. Node :- Data elements are represented in form of nodes.
   a) Internal b) External
3. Edge :- It connect two nodes or it is a connecting branch.
   Every node should have unique parent in the tree.
   Only root node do not have parent.
   Immidiate predecessor is k/as parent.
   Successor node of any node is called child node.
4. Degree of a node :- No of children is k/as degree of node.
5. Degree of tree :- Max^m degree.
6. Level :- Level of root is zero.
7. Siblings :- All children of same parent are k/as siblings.



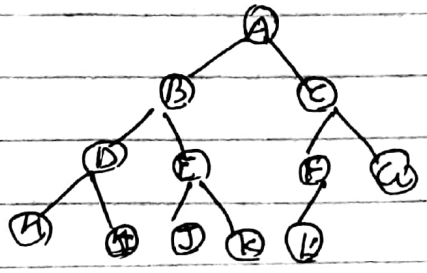
8. Ht. of tree :- Max^m no. of edges in the longest path.
9. Leaf :- The node which do not have any child {D, F, C}.
10. Depth of tree :- longest dist of node from any leaf
11. Subtress : Left subtree & Right subtree
    all nodes attached with left child — Node attached with right child.

Leaf are also called as external nodes.
Internal nodes have atleast one child.

⇒ BINARY TREE :- An node can have max^m two child node.

1. Full Binary Tree : All levels have max^m no of possible nodes.
   $\{2^0, 2^1, 2^2, 2^3 ----\}$
   at l^{th} level, there can be 2^l nodes.
   No. of internal nodes = n-1 ; n is no. of leaves.
   No. of external nodes = e+1 ; e is no. of internal nodes.

2. Complete Binary Tree :- All levels have max^m possible no of nodes except last level and the nodes at last level all the node must be as much as left aligned.



3. 2-tree :- A tree in which each node have either two or zero child.

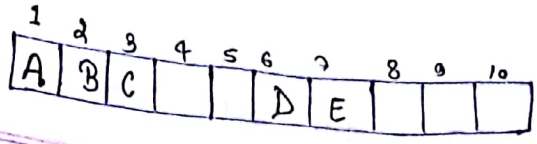
⇒ Representation of Binay tree :-

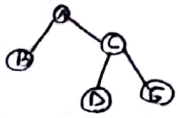
1. Sequential Representation (Array based)
2. Linked Represent-ation (Linked-List- based).

1. Sequential Representation :-

Parent - i^{th} location.

(i) left child $(2*i)^{th}$ location
(ii) Right- child $(2*i+1)^{th}$ location.





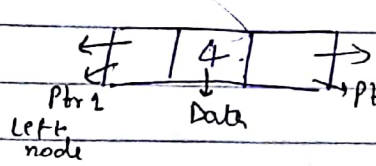
If a node is at i^th position,
parent = $[i/2]^{th}$ location.

Ex- $6^{th} = [6/2] = 3^{rd}$

$7^{th} = [7/2] = 3^{rd}$.

Disadvantages — for using Array based representation
i) Wastage of memory
ii) Insertion & deletion are cumbersome process.

2. Linked Representation:-

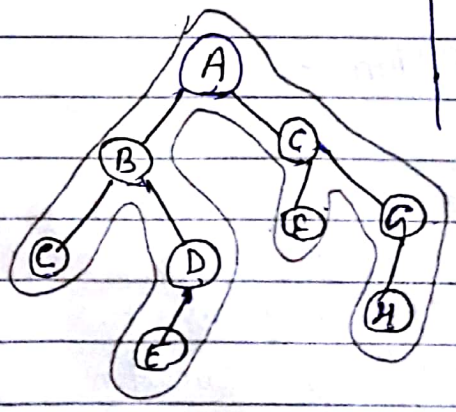


⇒ Tree traversal

i) Inorder traversal
   a) left- subtree
   b) Root-
   c) Right- subtree.

Algorithm:

```
while (node != NULL) {
  inorder (node → left);
  cout << node → data;
  inorder (node → right);
}
```

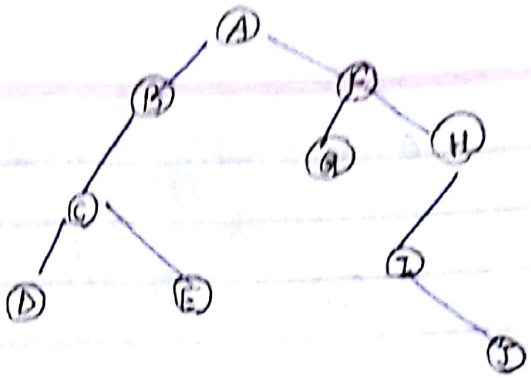


Inorder -
C B E D A F C H G

⇒ Print when encountered the node second time.

ii) Preorder traversal

a) Root (Process the root)
b) Left
c) Right



A B C D E F G H I J

⇒ Print the node in first encounter.

iii) Postorder traversal

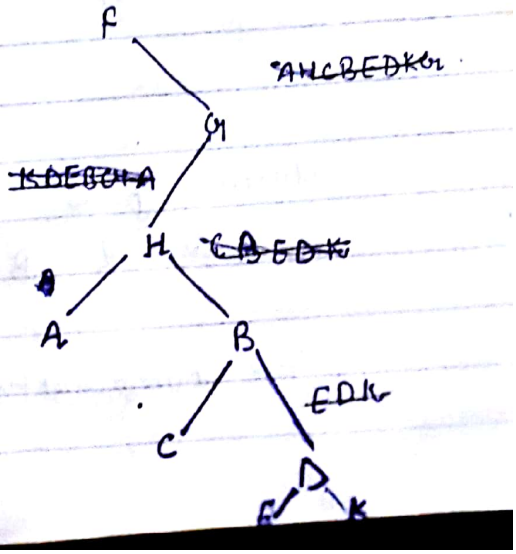
a) Left (subtree in postorder)
b) Right (subtree in postorder)
c) Root (Process the root)

D E C B A G J I H F A

⇒ Print the node that will not be visited again.

⇒ If inorder & preorder or inorder & postorder is given then a unique binary tree can be drawn.

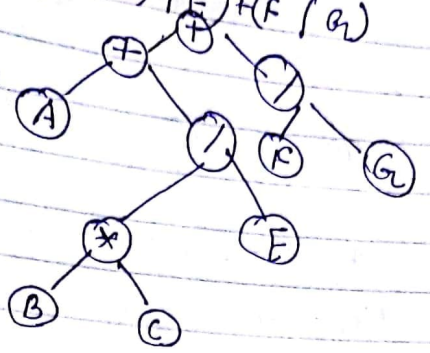
Q. IN F A H C B E D K G
POST A C E K D B H G F



Level Order Traversal :- Traverse the tree level wise.

Q

A+(B * C) / E) + (F / G)



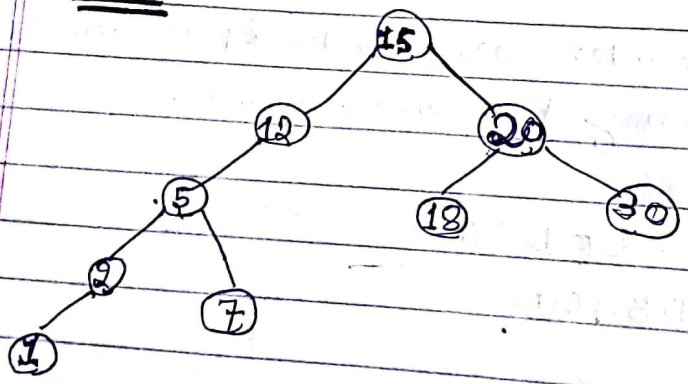
⇒ Binary Search Tree (BST).

It is a binary tree, in which for any node left-subtree contains smaller elements & right-subtree contain greater elements.

→ The left & right- subtrees each must also be a binary search tree.

Ex→ 15 12 5 2 1 7 20 30 18

BST



Note⇒ Inorder of BST :- 1 2 5 7 12 15 18 20 30

Inorder of BST gives elements in sorted order.

Time complexity :- O(log n). for searching.

Happy Navratri

10/10/18

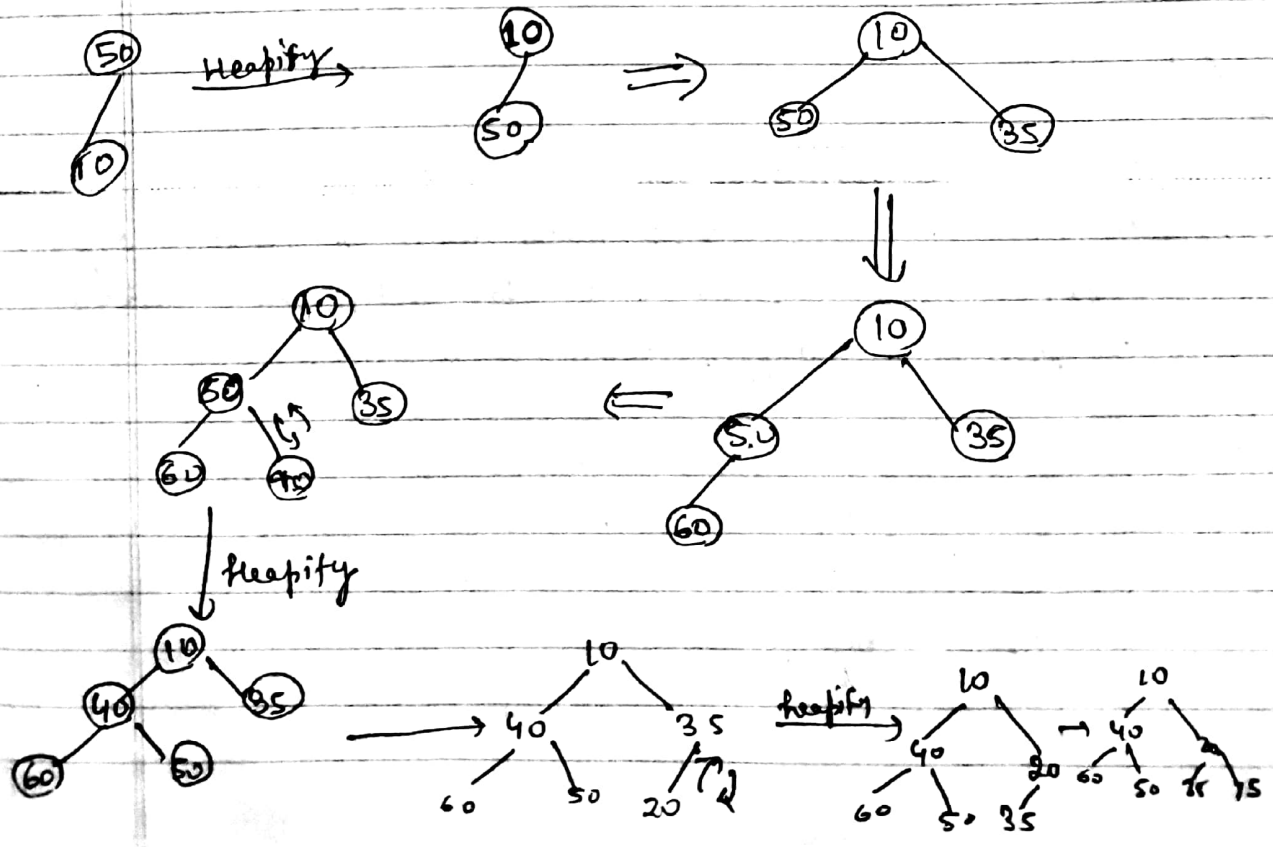
⇒ Deletion in BST :-

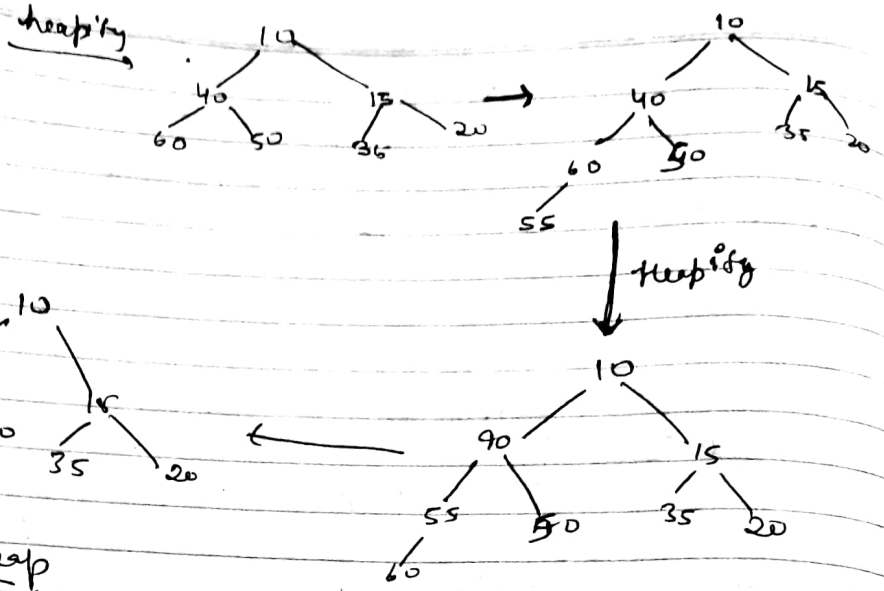
1) The node without child → delete the node
2) The node has single child → simply replace the node with child & delete child.
3) The node has two child → Inorder Successor will replace.

⇒ Heap tree :- → Min heap, → Max heap

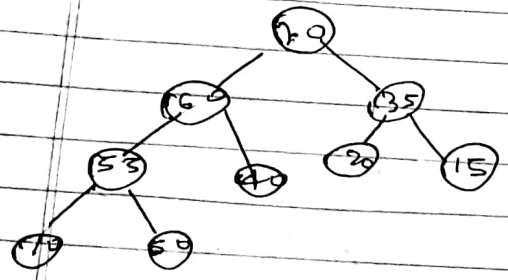
i) It is a complete binary tree.
ii) The parent- node should always be less than or equal to its children → Min Heap
The parent- node should be always greater than or equal to its children. → Max Heap.

50, 10, 35, 60, 40, 20, 15, 85, 70





Min-Heap



→ On deletion, the root node will be deleted by last node. After that for both child will be compare & swapped accordingly.

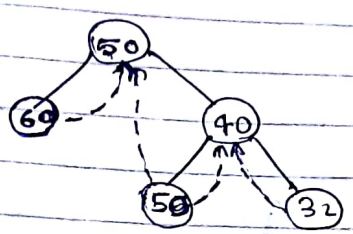
→ It will give elements in sorted order.

→ In case of heap tree, from memory point of view array is better than linked list as it is a complete binary tree..

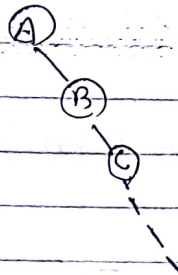
⇒ Threaded Binary Tree: A binary tree with 'n' nodes has 2n pointers out of which (n+1) are always null, hence they can be used to store the value of inorder successor & inorder Predecessor. These pointers are called threads.

1. 2 tree or extended tree is

Left pointer → inorder Predecessor
Right pointer → inorder Successor.



⇒ AVL Tree:- Balanced Binary Search Tree.



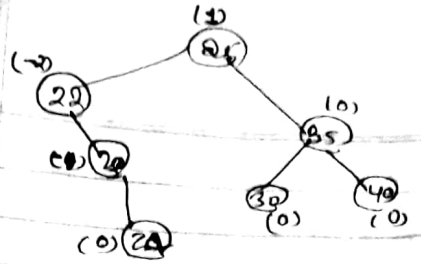
→ Skewed tree
→ time taken to search is O(n)
→ no advantage.

→ there will be advantage in case of balanced tree.
O(log n).

Balance factor → $|h(T^L) - h(T^R)| \leq 1$ ⇒ {0, -1, 1}.

$h(T^L)$ → height of left subtree.
≤ 1 → must be true for all node.

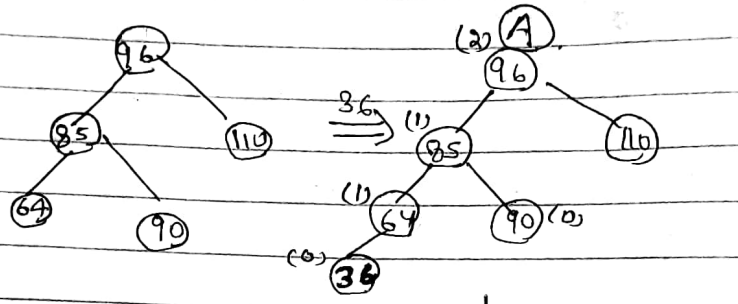
⇒ To convert a tree into AVL Tree → we perform rotation.
Types of Rotation
1. LL → left subtree of left subtree to A
2. RR → Rotate along right subtree of right subtree to A.

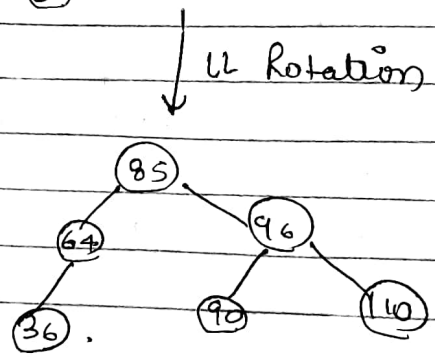


Not an AVL tree
(22 does not satisfy).

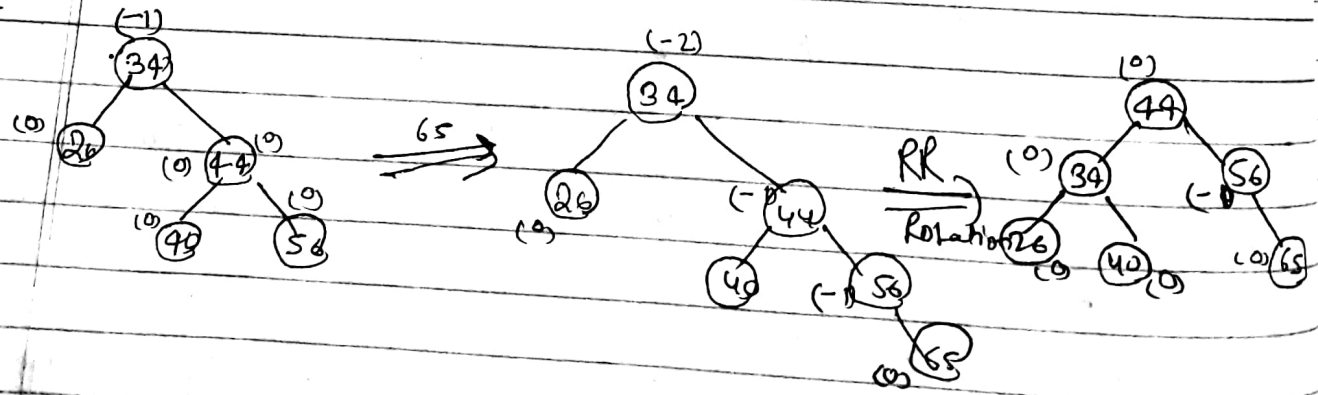
(III) LR - Rotate along right-subtree of left subtree from A.
(IV) RL - Rotate along left-subtree of Right-subtree from A.

1. LL

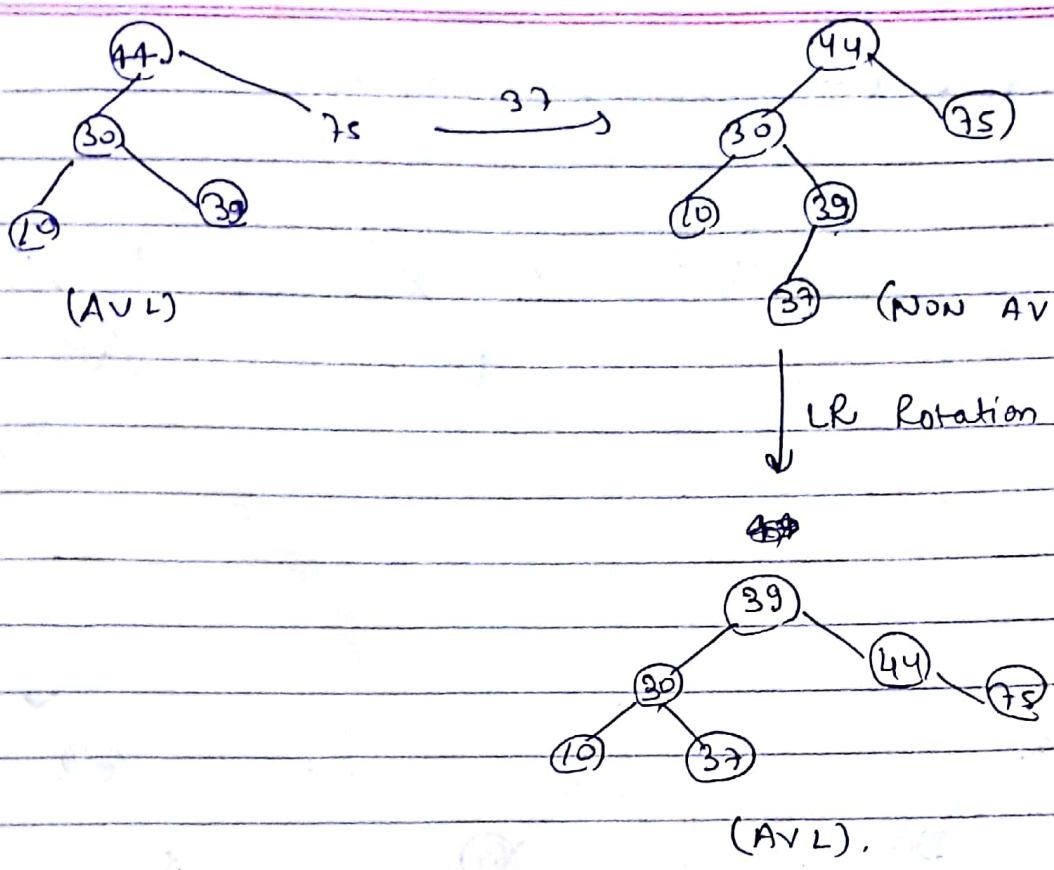




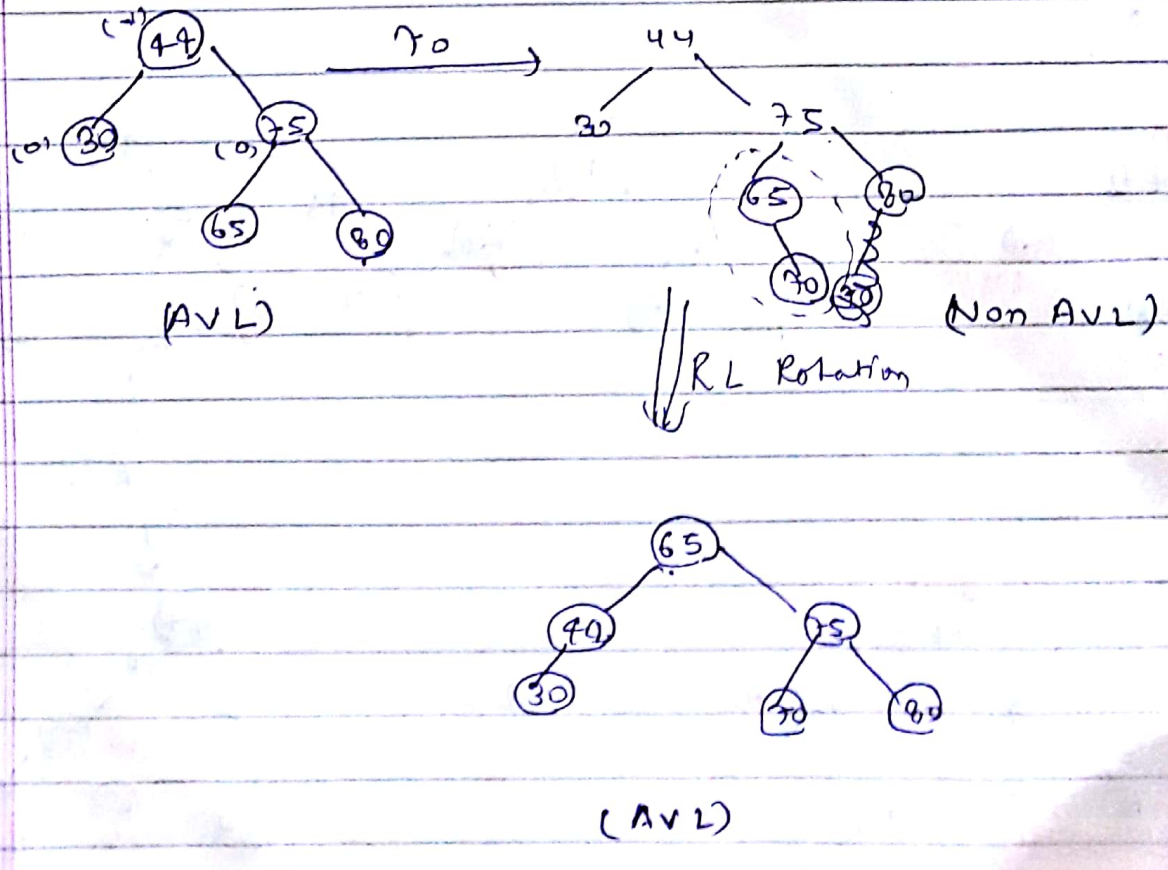
2.



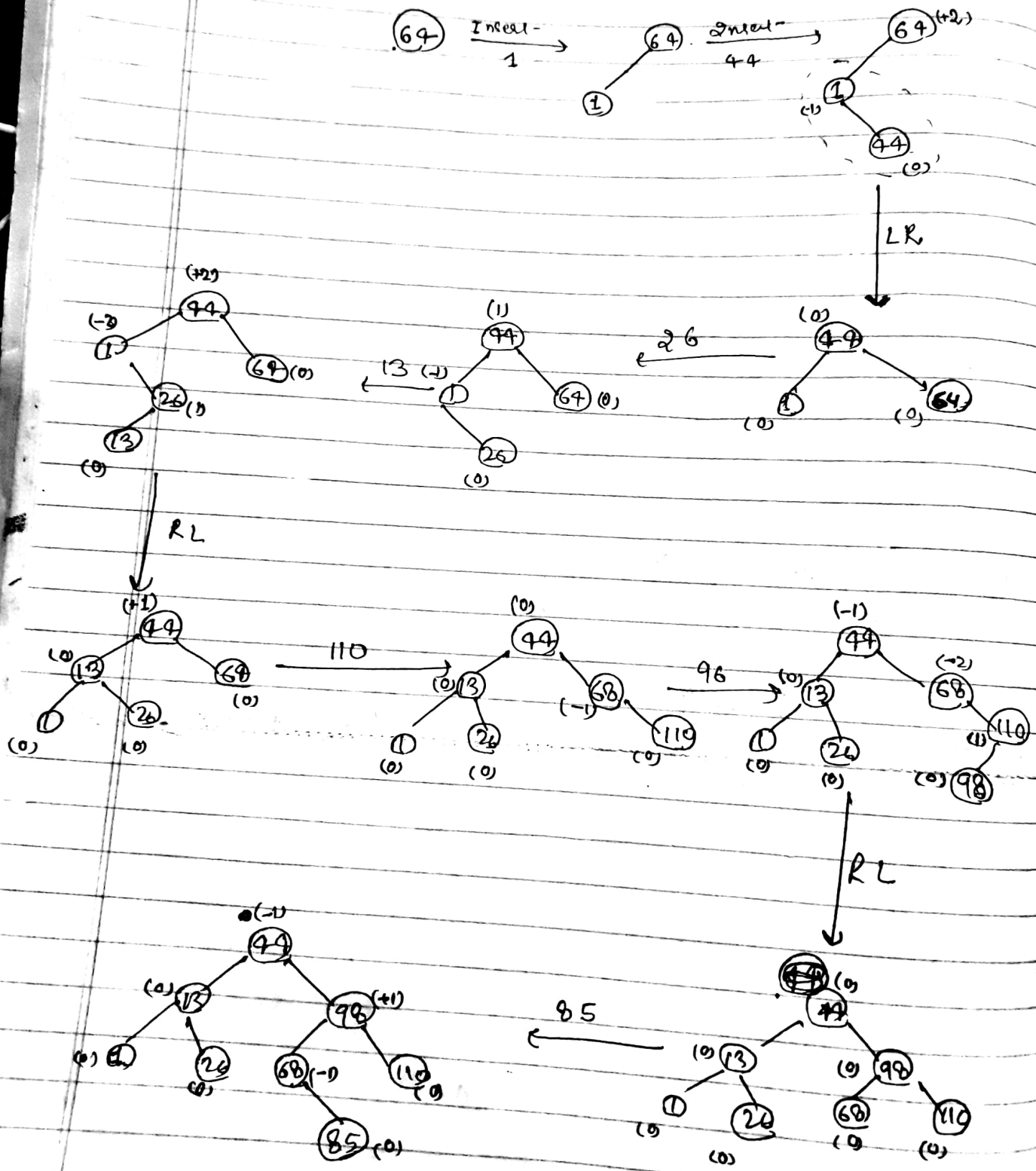
3.



4.



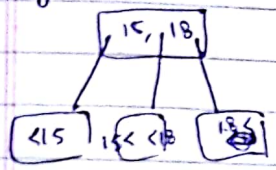
Q: 64, 1, 44, 26, 13, 110, 98, 85



⇒ **B-Tree** : No restriction regarding no of childs

1) B-Tree of order 'm' (max^m 'm' child).
2) All leaves are at same lvl.
3) Root of a B-tree can have min^m two & max^m 'm' childs. or (m-1) keys/element-s.
4) Non-roots $\lceil m/2 \rceil$ childs or $(\lceil m/2 \rceil - 1)$ keys.

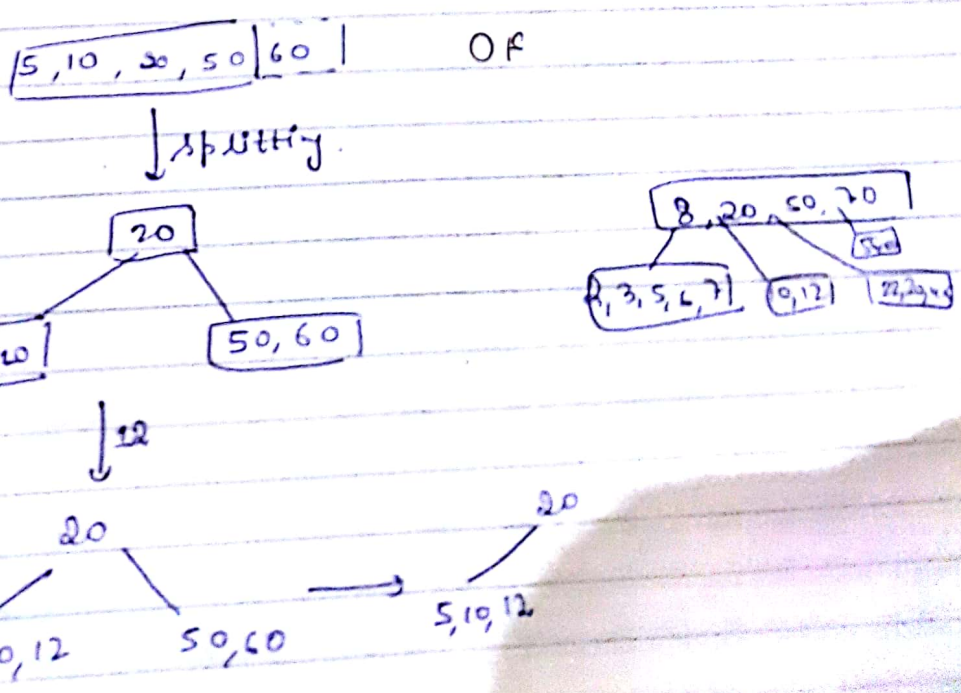
eg.



5) Root can have min 2 child (1 key) & max^m m child.
6) All leaf-nodes are at same level.

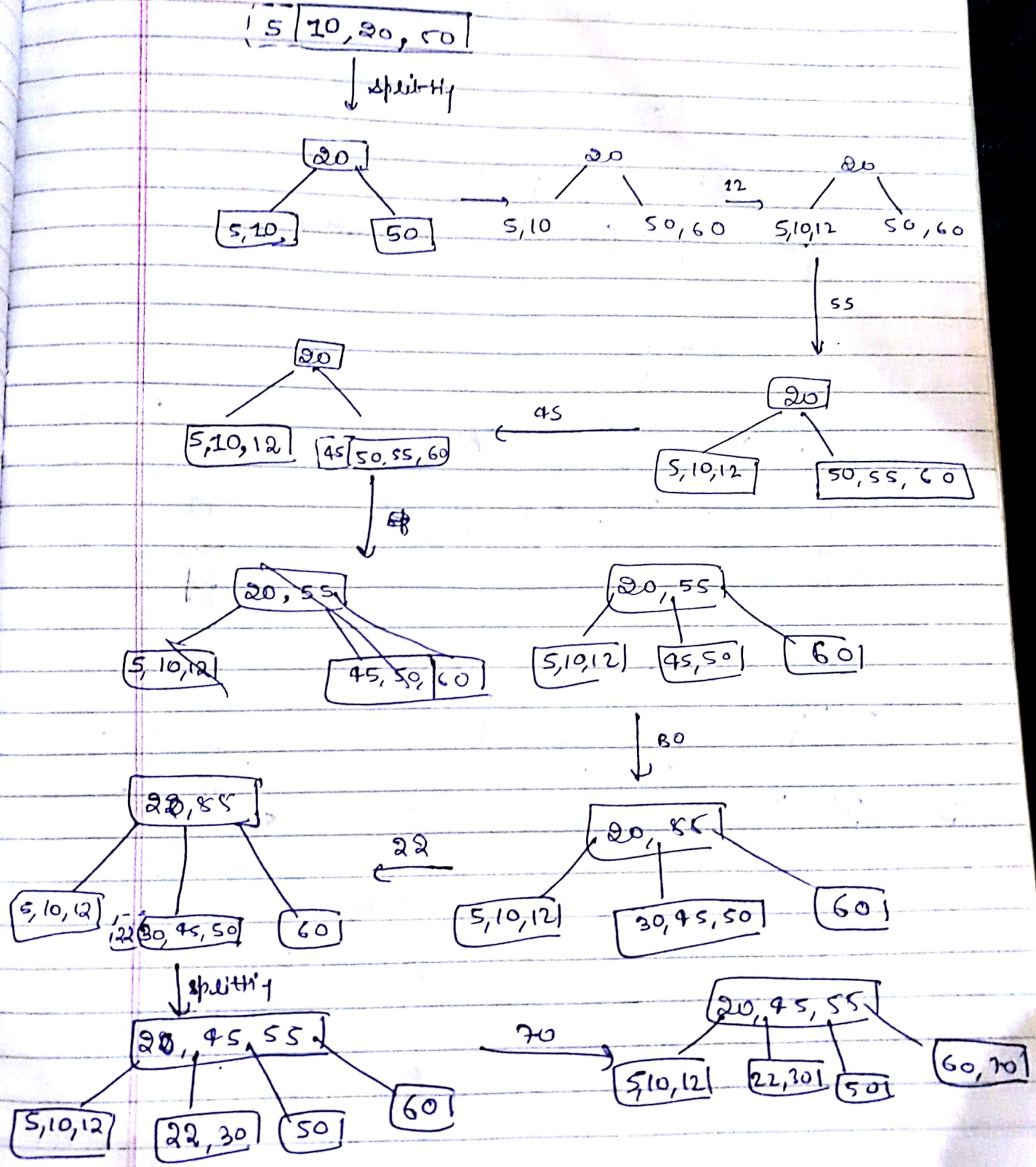
2) also satisfies all prop. of BST.

ex 20, 10, 30, 5, 60, 40, 12, 55, 45, 30, 22, 70, 8, 2, 75, 80.



Order '4'

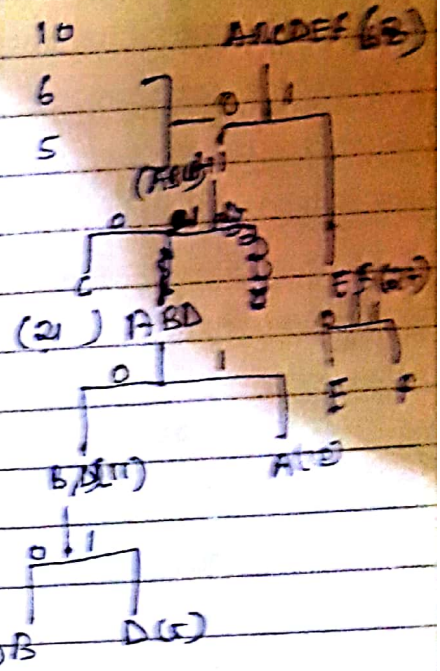
20, 50, 10, 5, 60, 12, 55, 45, 30, 22, 70, 8, 2, 75, 80, 6, 3, 7



Huffman tree :- Order of decreasing frequency

A	10
B	5
C	20
D	6
E	12
F	15

C	20
F	15
E	12
A	10
B	6
B	5



C	20
F	15
E	12
B, D	11
A	10

ABD	21
C	20
F	15
E	12

EF	27
ABD	21
C	20

ABCD	41
EF	27

→ ABCDEF (68)

Codes

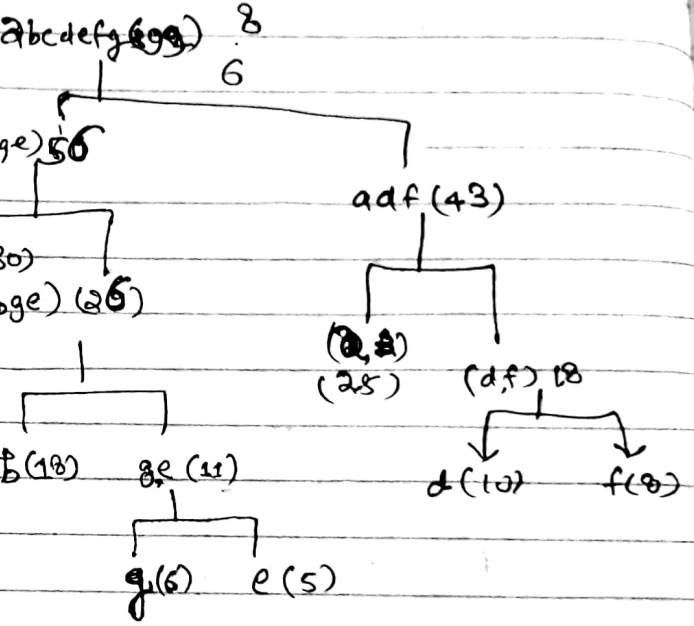
A	011
B	0100
C	00
D	0101
E	10

F 13

smaller → left
Bigger → Right

Q

Symbol / data	Freq
a	25
b	15
c	30
d	10
e	5
f	8
g	6



① c 30
a 25
b 15
d 10
f 8
g 6
e 5

② c 30
a 25
b 15
~~d 10~~
(g,e) 11
d 10
f 8

③ c 30
a 25
(d,f) 18
b 15
g,e 11

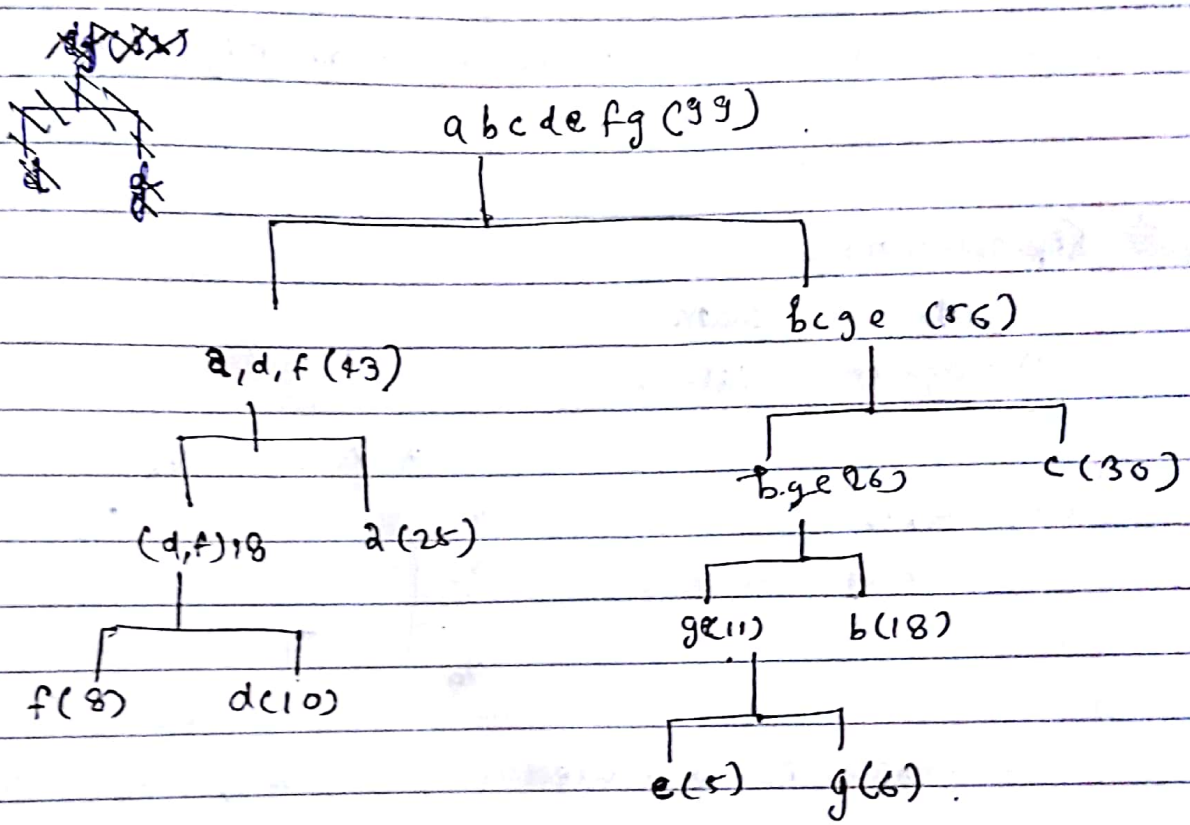
④ c 30
bge 25
a 25
df 18

⑤ adf 43
c 30
bge 25

⑥ a df bcge
25 18 55

⑦ 55 + 43 → abcdefg (99)

⇒ Huffman tree :-



30/10/18

GRAPH

Tree is a graph not having cycle.

$<u,v>$ → means there exist an edge b/w u & v.

⇒ Representation.

(i) Adjacency Matrix
(ii) Adjacency List.

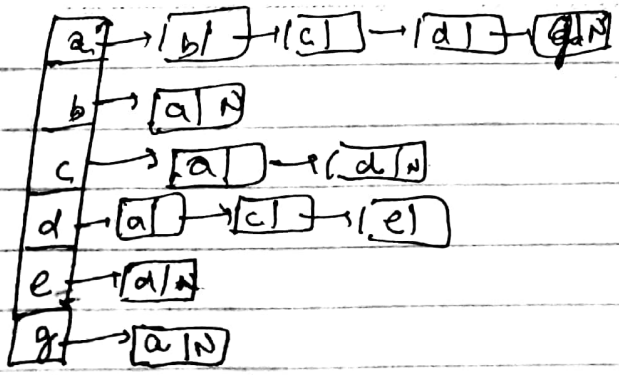
(i) $n \times n$
↓
no. of vertices.

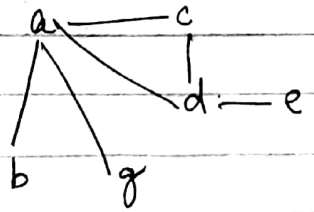
	v_1	v_2	...	v_n
v_1	-	-	-	-
v_2	-	-	-	-
⋮				
v_n	-	↓	-	→

elements are in form 0, 1

if graph is weighted graph, then wt. is place in place of 0, 1.

(ii) Adjacency List :- Array of Structure.



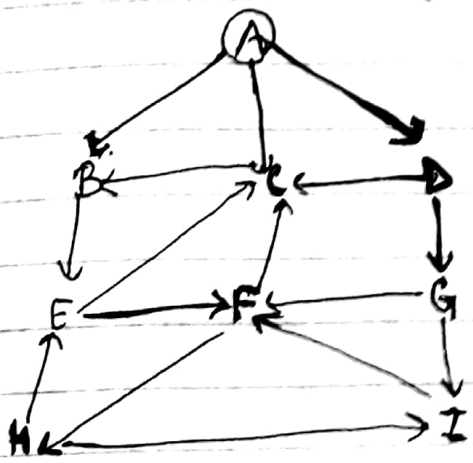


For searching whether two nodes are connected → O(v)
For insertion O(1).

⇒ Graph Traversal :-

1. BFS (Breadth First- Search) → Queue
2. Depth First- Search (DFS) → Stack.

Example :-



BFS :- A B C D E G F I H.

Dfs :- A D G I F H E C B.

⇒ Minimum Spanning tree :-

Graph → n, e

Spanning tree → n, $n-1$; $(e = n-1)$

$e C_{n-1}$

No. of spanning tree of graph $= \left(|E| C_{n-1} - \text{no of cycles} \right)$

⇒ Algorithms to find minimum Spanning tree :-

1. Prim's Algo.
2. Kruskal's Algo.

1. Prim's Algo. (Greedy Algo.)

(i) Select - min^{m} wt. edge.

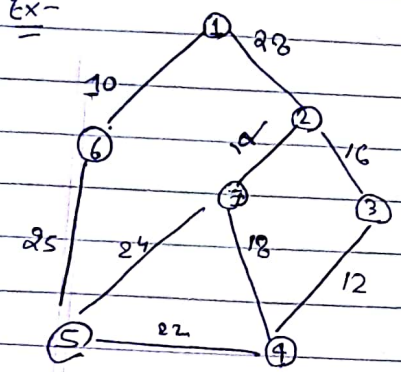
(ii) Select - the min^{m} wt edge connecting any of the two vertices to new vertex

Repeat - the (ii) step until all vertices are there.

2. Kruskal's Algo.

Find the min^{m} edge that may be disjoint also.

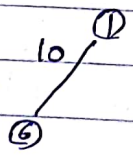
Ex-



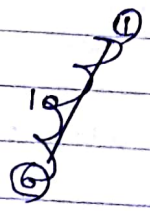
$\{10, 12, 16, 18, 22, 24, 25, 28\}$

① Prim's Algo.

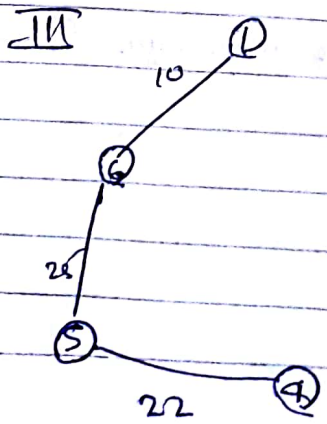
I



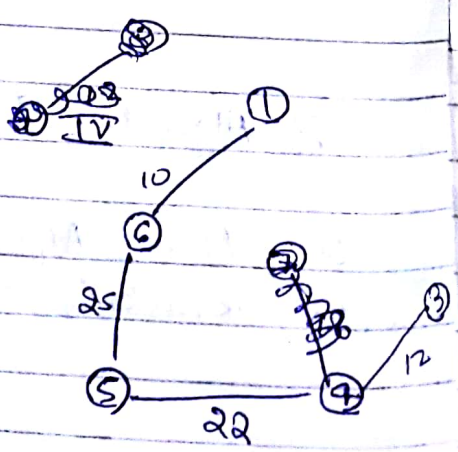
II



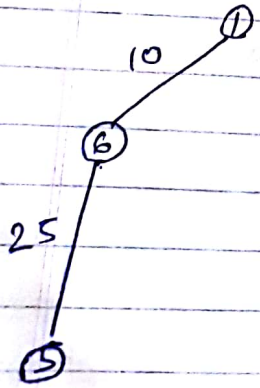
III



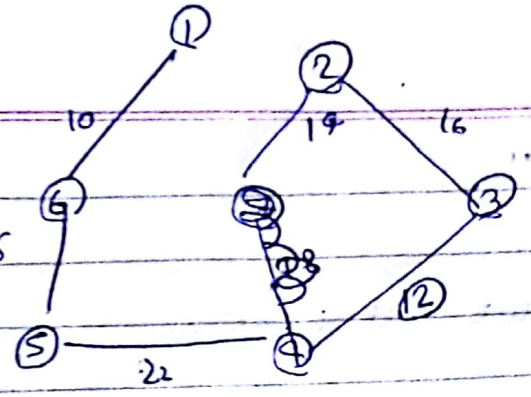
IV



V

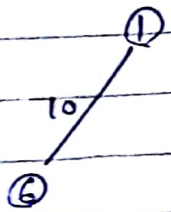


(V)

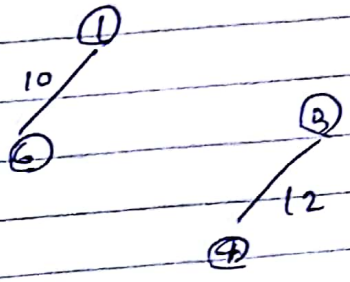


(VI)

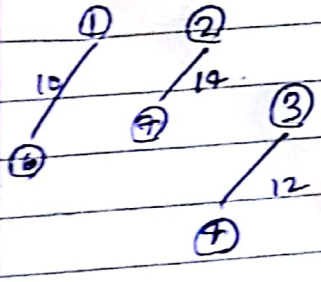
I.



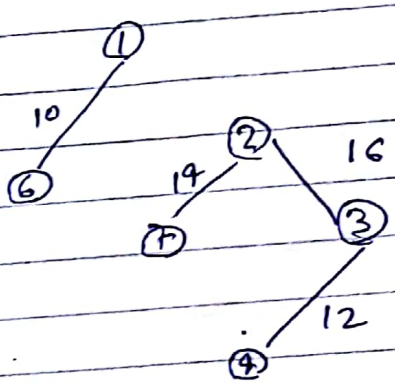
II



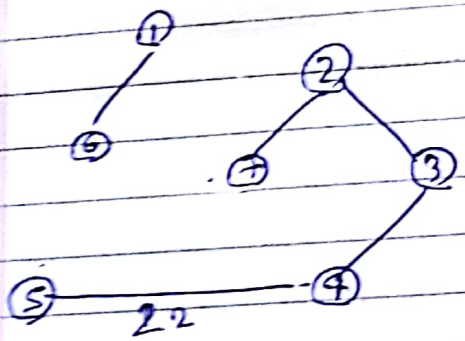
III.



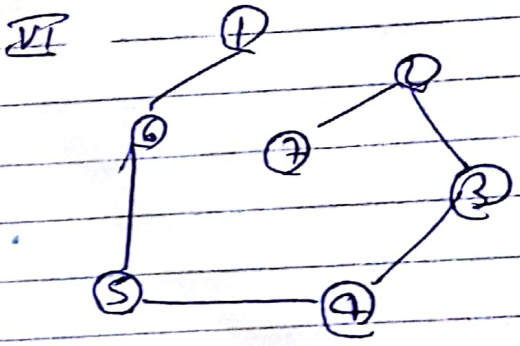
IV



V



VI



99