⊙ Deletion in binary search tree has 3 cases:

1) Node has no child : In this case we simply delete the node.

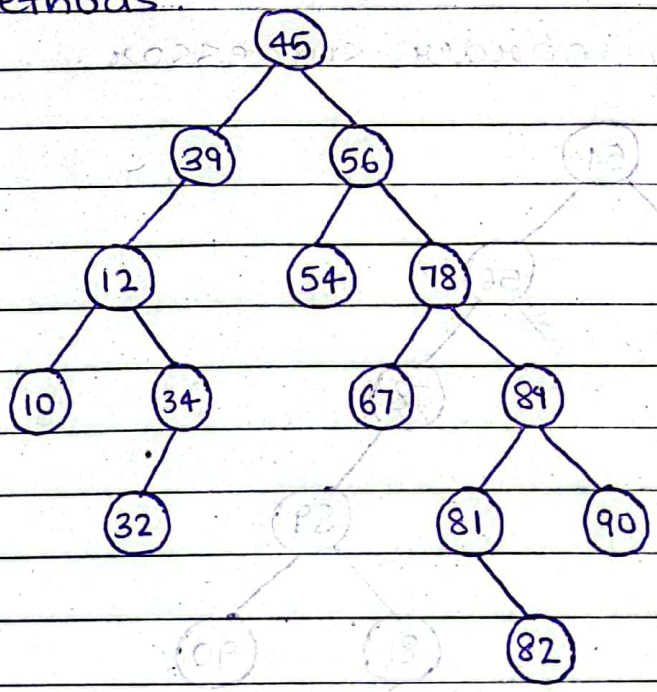
2) Node has one child : In this case the child node replaces the parent node

3) Node has two child nodes : For this case, there are two techniques:

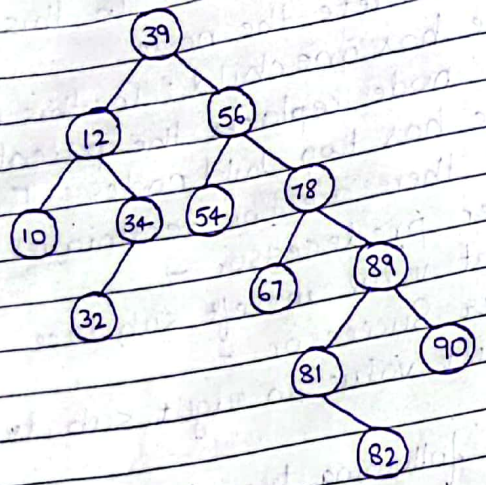
(i) Inorder predecessor -
Largest value in left sub tree

(ii) Inorder Successor -
Smallest value in right sub tree

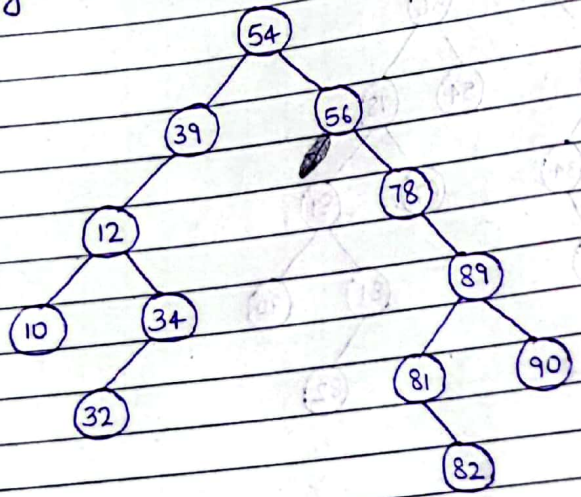
Q. From the following tree delete 45 using both methods.



Deleting 45 using inorder predecessor:



Deleting 45 using inorder successor:



* Algorithm for Searching in Binary Search Tree :

STEP 1 : Start
STEP 2 : Take from user the value that is to be searched. (value)
STEP 3 : search (root, value)

```
{
    if (root = null)
        return
    else if (root->data = value)
        return root
    else if (root->data > value)
        search(root-> left, value)
    else
        search(root -> right, value)
}
```

STEP 4 : Stop.

20/06/22.

* Algorithm for Insertion in Binary Search Tree

STEP 1 : Start
STEP 2 : Get data from user that has to be insert (value)
STEP 3 : insert (root, value)

```
{
    if (root = null)
    {
        newnode = (struct node *) malloc (sizeof (struct node))
        root → data = value
        root → left = root → right = null
    }
    else if ( root → data > value)
    {
        insert ( root → left, value)
    }
    else
    {
        insert (root → right, value)
    }
}
```

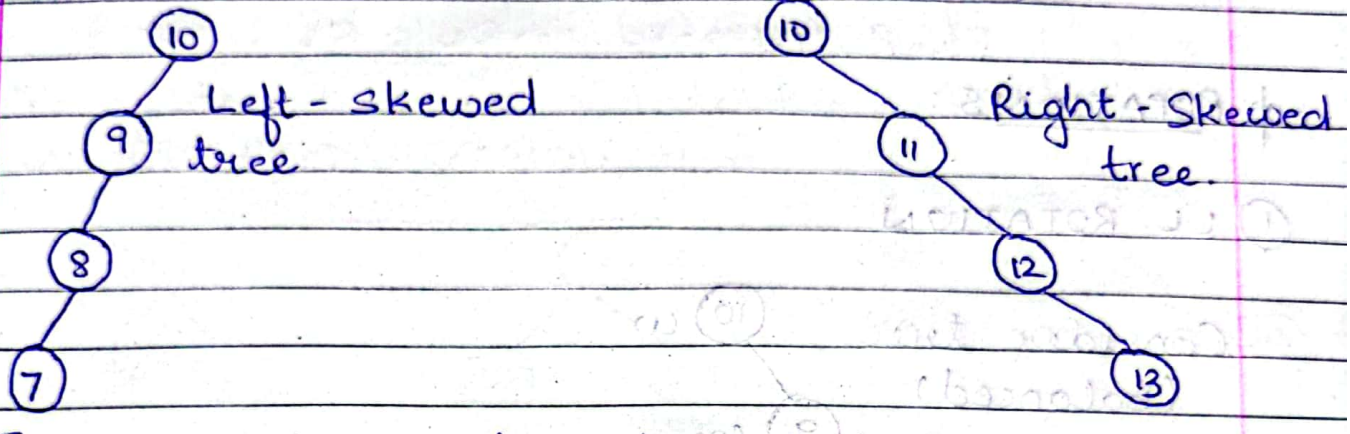
STEP 4 : Stop

20/06/22.

* AVL TREE

- Invented by Adelson-Velsky and E.M. Landis in 1962.
- It is a self-balancing binary search tree.
- In an AVL tree, searching is easier and less complex compared to a binary tree.

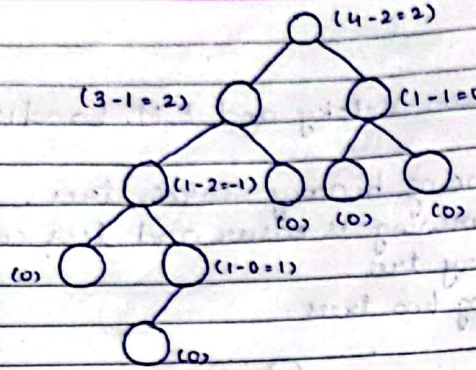
Consider the following two trees:



- For searching an element in such trees, we need to perform n number of comparisions.
- For a binary tree searching,
  Best case - Order (log n)
  Average case - Order (log n)
  Worst case - Order (n).
- We can modify such trees and convert them into balanced binary tree for efficient searching.
- A node is called balanced if its balance factor is 0, 1 or -1.

- Balance factor = Height of left sub-tree - Height of right binary tree.
- A node that has no child will have balance factor 0, as it has no sub-trees.

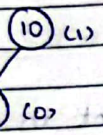
eg)



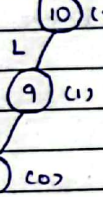
* ROTATIONS:

(1) LL ROTATION

Consider tree (balanced)

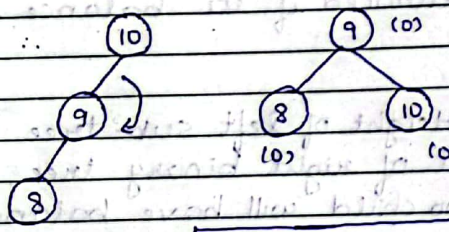


Insert 8



This tree becomes unbalanced due to insertion in left sub-tree of left sub-tree.

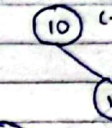
To balance this we rotate the unbalanced node in clockwise direction.



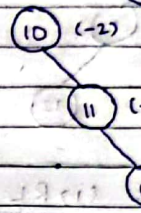
∴ LL ROTATION :- CLOCKWISE

(2) RR ROTATION

Consider tree (Balanced)

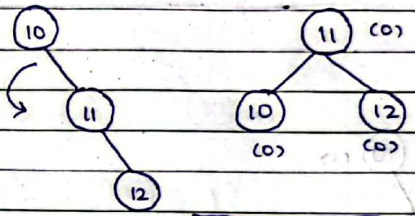


Insert 12



This tree becomes unbalanced due to insertion in right sub-tree of right sub-tree.

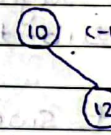
To balance this, we rotate the unbalanced node in anticlockwise direction.



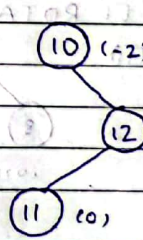
∴ RR ROTATION :- ANTICLOCKWISE

(3) RL ROTATION.

Consider tree (Balanced)

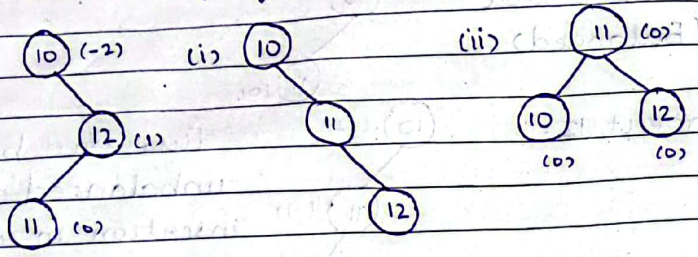


Insert 11



This tree becomes unbalanced due to insertion in left sub-tree of the right sub-tree.

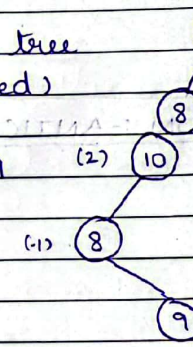
To balance this, we first swap the last two nodes and then perform RR Rotation.



∴ RL ROTATION: (i) RL : RL
(ii) ANTICLOCKWISE

(4) LR ROTATION

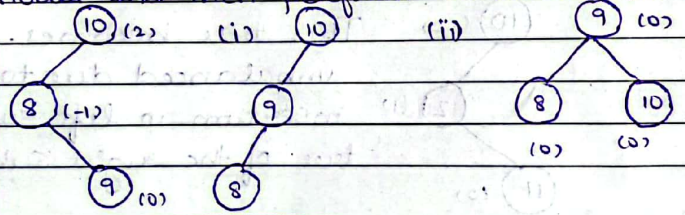
Consider tree (Balanced)



Insert 9

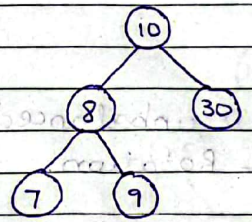
This tree becomes unbalanced due to insertion in right sub-tree of left sub-tree.

To balance this, we first swap the last two nodes and then perform LL ROTATION.

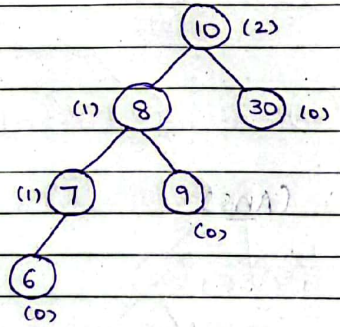


∴ LR ROTATION: (i) LR : LR
(ii) CLOCKWISE

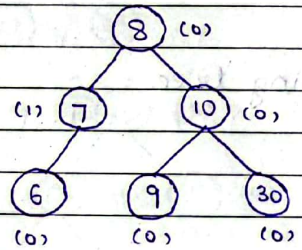
eg) Insert 6 in the following tree:



→

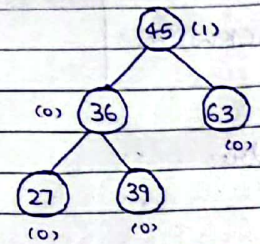


10 is unbalanced.
LL Rotation

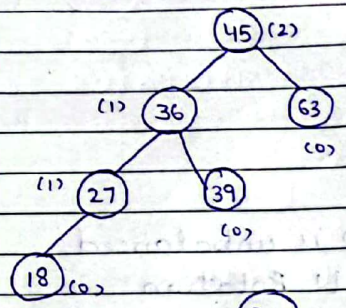


(Ans)

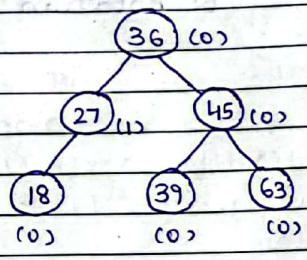
Q. In the following tree, insert 18



→

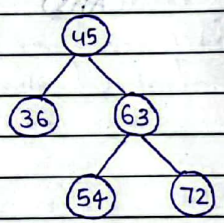


45 is unbalanced.
LL Rotation.

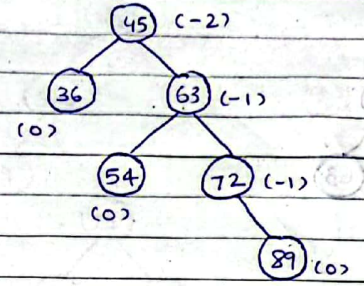


(Ans)

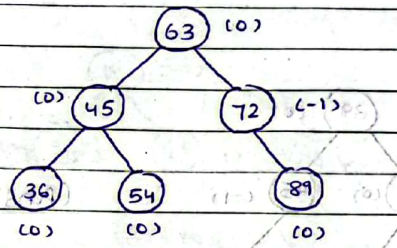
Q. Insert 89 in the following tree:



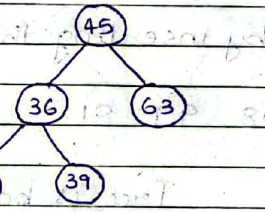
→



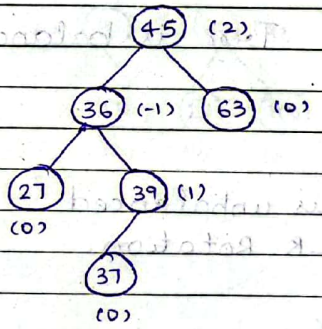
45 is unbalanced.
RR Rotation.



Q. Insert 37 in the following tree:

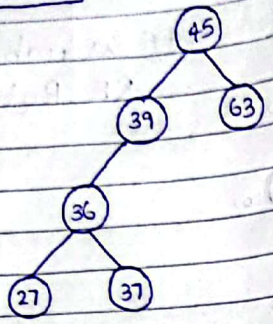


→

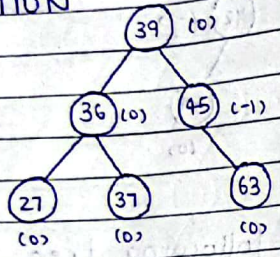


45 is unbalanced.
LR Rotation.

LR : LR



LL ROTATION



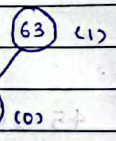
(Ans)

Q. Construct an AVL Tree by inserting the following elements:

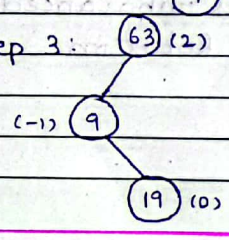
63 9 19 27 18 108 99 81

Step 1: 

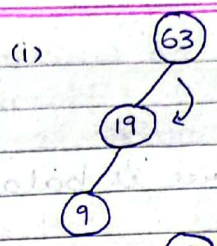
Tree is balanced

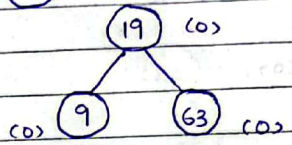
Step 2: 

Tree is balanced

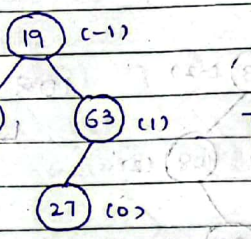
Step 3: 

63 is unbalanced
LR Rotation.

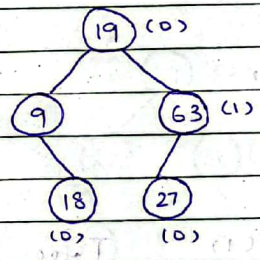
(i) 

(ii) 

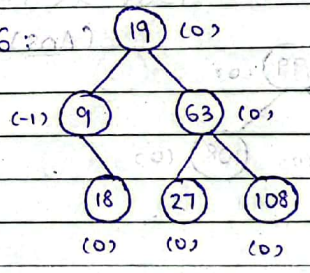
Tree is balanced.

Step 4: 

Tree is balanced.

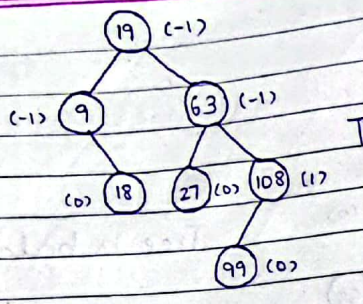
Step 5: 

Tree is balanced.

Step 6: 

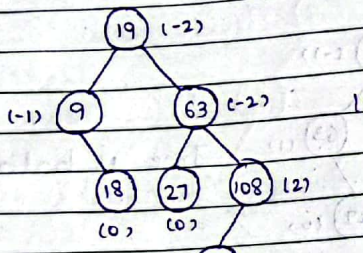
Tree is balanced.

Step 7:

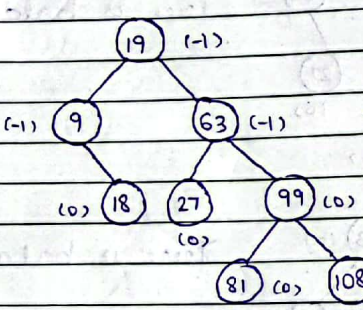


Tree is balanced.

Step 8:



108 is unbalanced
LL Rotation.



Tree is balanced.
(Ans)

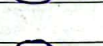
22/06/22.

Q. Construct an AVL Tree by inserting the following elements:

22 27 31 10 5 15 29 19 16 11 3 4 8

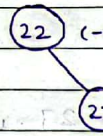
→

Step 1:



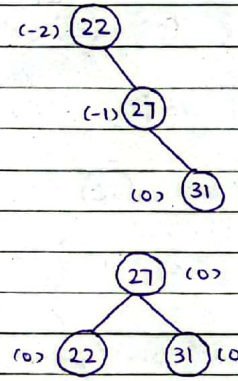
Tree is balanced

Step 2:



Tree is balanced

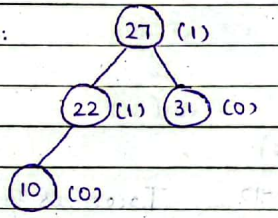
Step 3:



22 is unbalanced
RR Rotation.

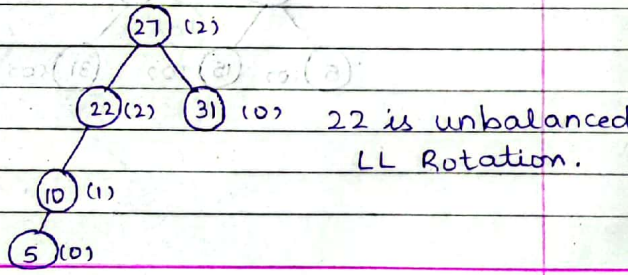
Tree is balanced.

Step 4:

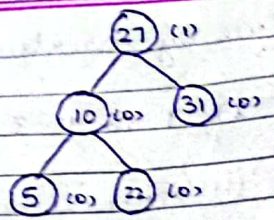


Tree is balanced.

Step 5:

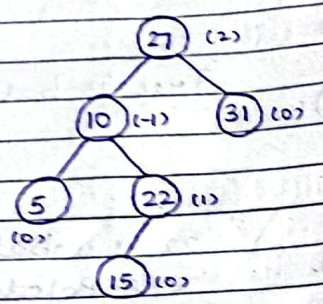


22 is unbalanced
LL Rotation.



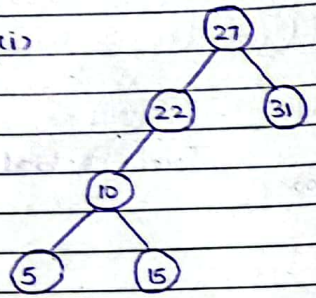
Tree is balanced.

Step 6:

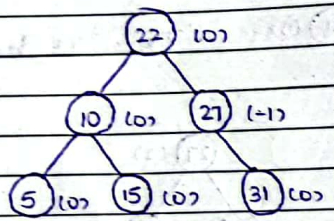


27 is unbalanced
LR Rotation

(i)

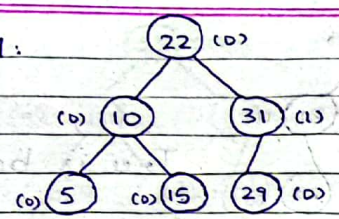


(ii)



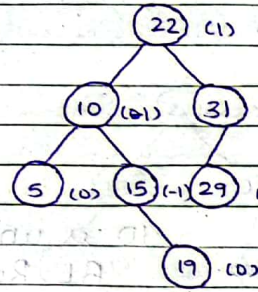
Tree is balanced.

Step 7:



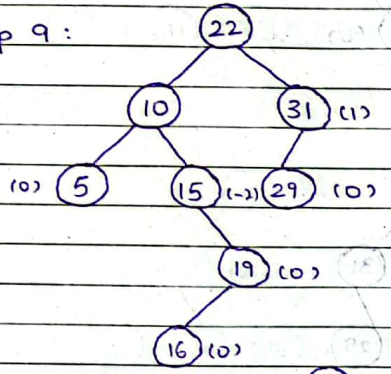
Tree is balanced.

Step 8:



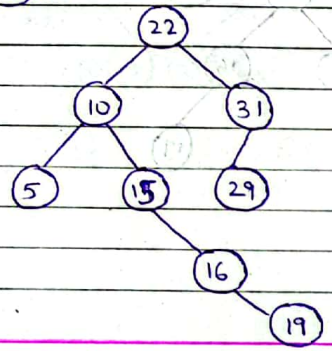
Tree is balanced.

Step 9:

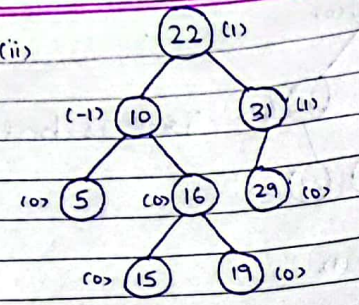


15 is unbalanced
RL Rotation.

(i)

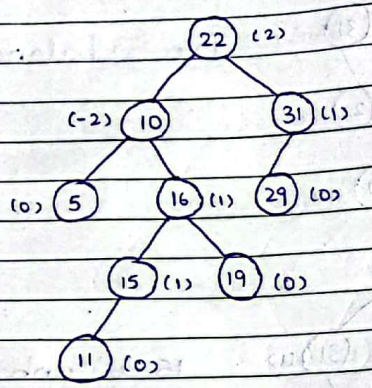


(ii)



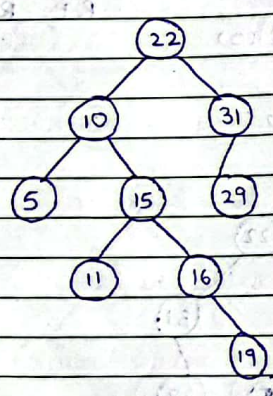
Tree is balanced

Step 10 :

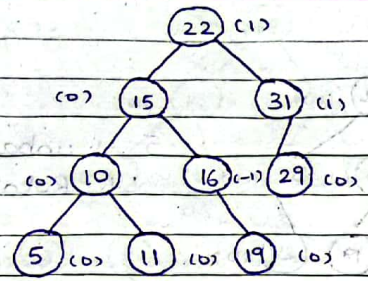


10 is unbalanced
RL Rotation.

(i)

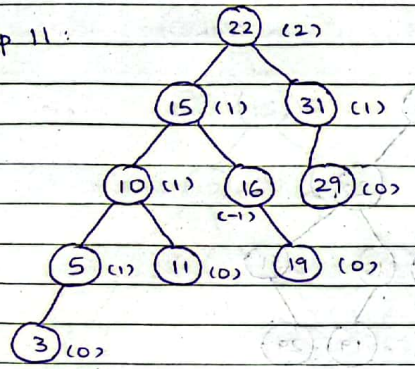


(ii)

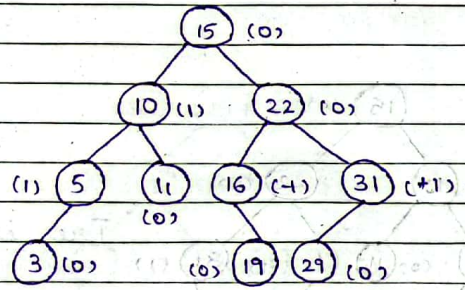


Tree is balanced.

Step 11 :

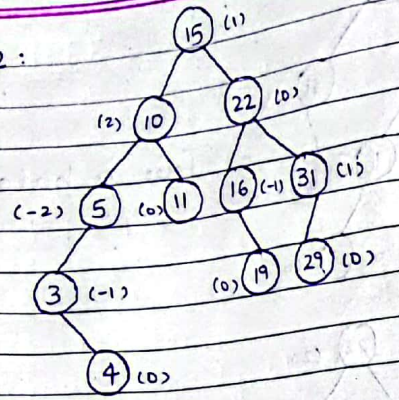


22 is unbalanced
LL Rotation



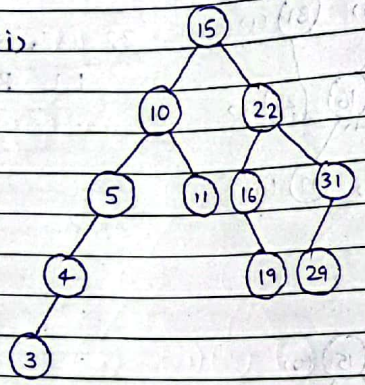
Tree is balanced.

Step 12 :

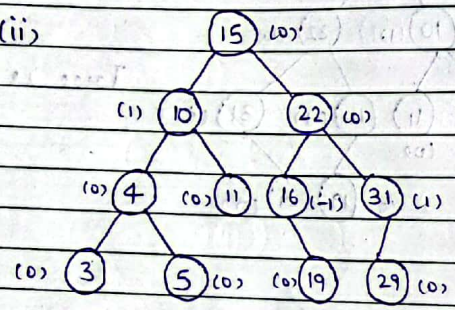


5 is unbalanced
LR Rotation

(i)

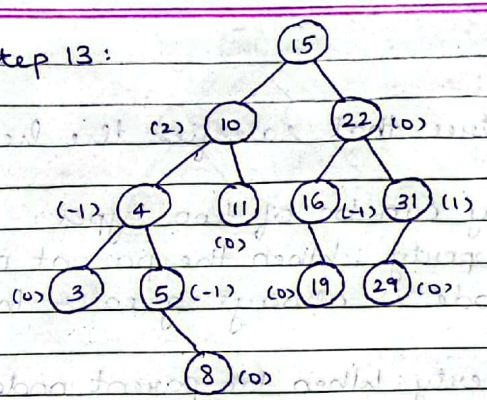


(ii)



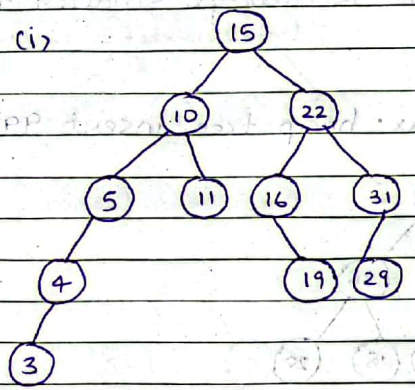
Tree is balanced.

Step 13 :

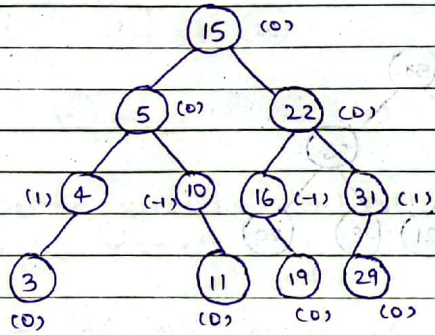


10 is unbalanced
LR Rotation.

(i)



(ii)

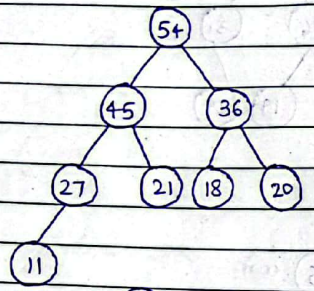


Tree is balanced.
(Ans)

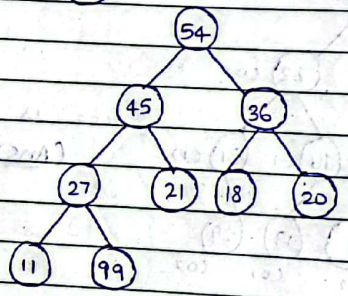
* HEAP TREE

- It is a binary tree that satisfies the heap property.
- The heap property can be of two types:
  - (i) Max heap property: When the parent node or the root node is always greater than the child node.
  - (ii) Min heap property: When the parent node or the root node is always smaller than the child node.

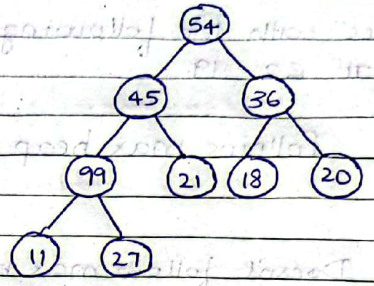
eg) In the following max heap tree, insert 99.



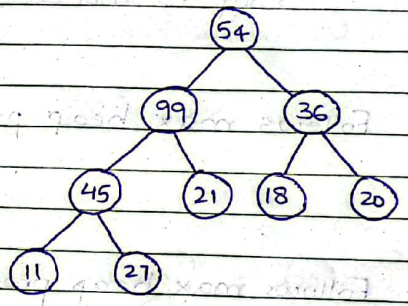
→



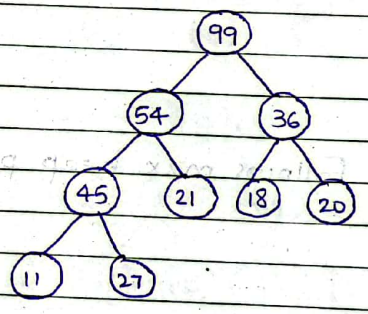
But this tree does not follow max heap property. So we swap parent node and child node till it does.



Does not follow max heap property.



Does not follow max heap property.



Max. heap Tree (Ans)

Q. Refer above question: What is the order of the tree after insertion of 99? (GATE).

99 54 36 45 21 18 20 11 27

Q. Create a max heap tree with the following elements:
45 51 24 32 64 71 62 19

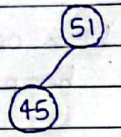
→

Step 1: 

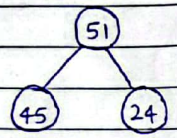
Follows max heap property

Step 2:

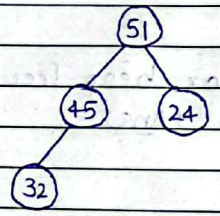
Doesn't follow max heap property
Swap 45 and 51



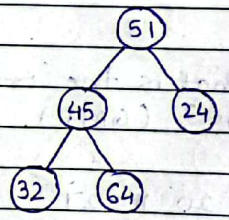
Follows max. heap property.

Step 3: 

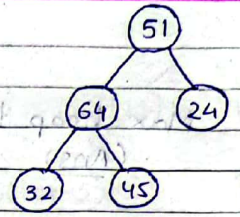
Follows max. heap property

Step 4: 

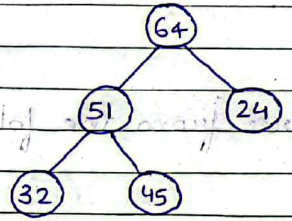
Follows max. heap property

Step 5: 

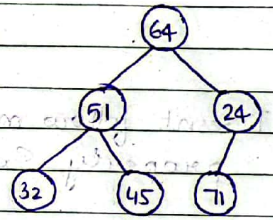
Doesn't follow max. heap property:
Swap 45 and 64



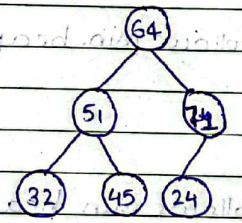
Doesn't follow max. heap property
Swap 64 and 51.



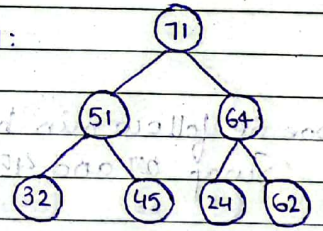
Follows max. heap property.

Step 6: 

Doesn't follow max. heap property
Swap 24 and 71



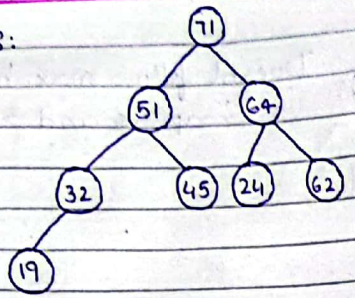
Swap 64 and 71

Step 7: 

Follows max heap property

22/06/'22

Step 8:

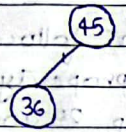


Max. heap tree (Ans)

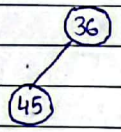
Q. Construct a min. heap tree from the following:
45, 36, 54, 27, 63, 72, 61, 18

Step 1:

Step 2:

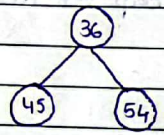


Doesn't follow min. heap property, Swap 45 and 36



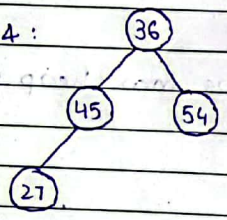
Follows min. heap property

Step 3:

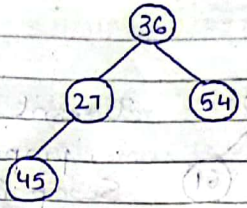


Follows min heap property

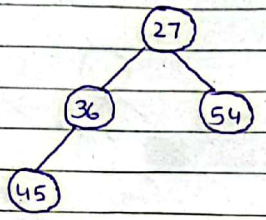
Step 4:



Doesn't follow min heap property
Swap 27 and 45

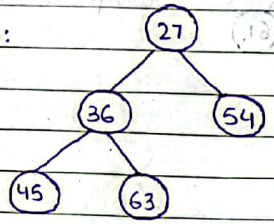


Swap 36 and 27



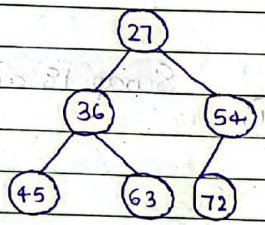
Follows min heap property

Step 5:



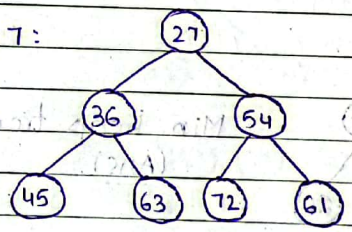
Follows min heap property

Step 6:



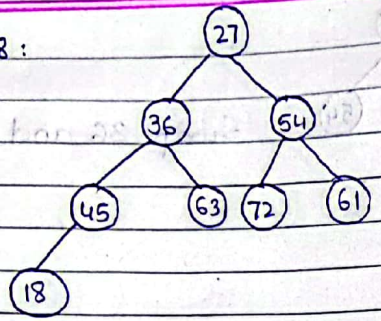
Follows min. heap property

Step 7:

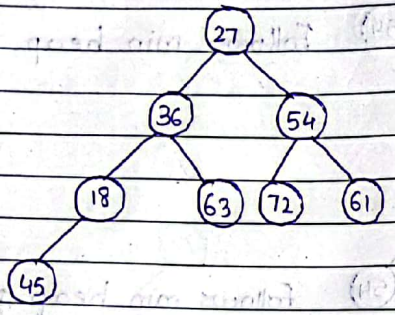


Follows min. heap property

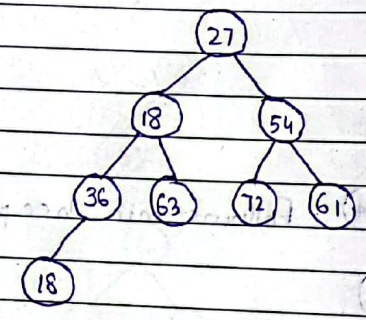
Step 8:



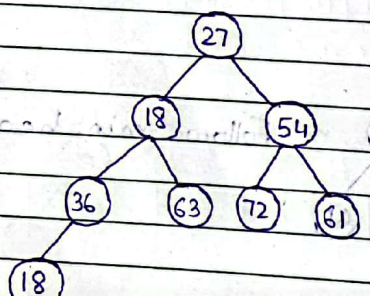
Doesn't follow min. heap property
Swap 18 and 45



Swap 18 and 36



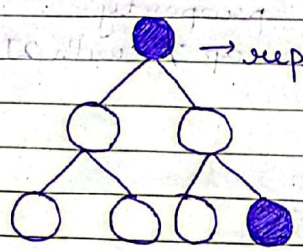
Swap 18 and 27



Min. heap tree
(Ans)

* DELETION IN HEAP TREE

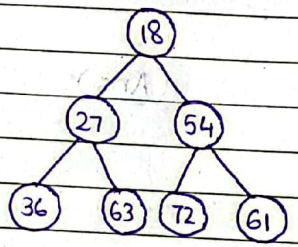
Constraint : We can only delete the root node or the node that has been inserted last.



→ replace with last element

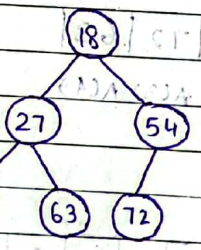
→ delete directly.

Q. In the following question, delete:
(i) The last node
(ii) The root node

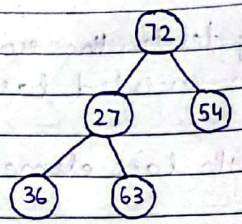


→

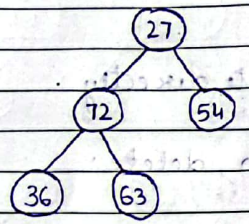
Deleting last node:



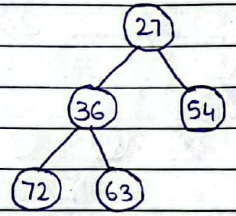
Deleting root node:



Doesn't follow min heap property.
Swap 72 with 27



Swap 36 with 72



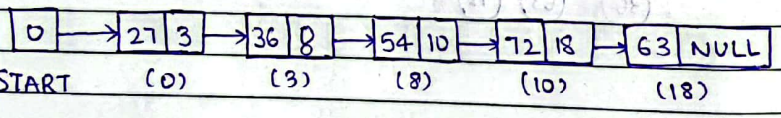
(Ans)

⊙ Memory representation for above heap tree:

Using Array:

27	36	54	72	63
A(0)	A(1)	A(2)	A(3)	A(4)

Using linked list:



* Algorithm for insertion in Heap Tree: (Max.)

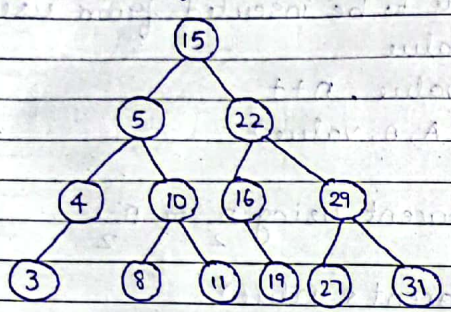
STEP 1: Start
STEP 2: Get value to be inserted from user.
STEP 3: Insert value
A[n] = value, n++
STEP 4: Insert (A, n, value)
{
Find parent using $\frac{n}{2}$ or $\frac{n-1}{2}$

if (parent < value)
{
temp = parent;
parent = value;
value = temp;
}
}
STEP 5: ~~while~~ n = parent
STEP 6: while (n > 0)
{
Insert (A, n, value)
}
STEP 7: Stop

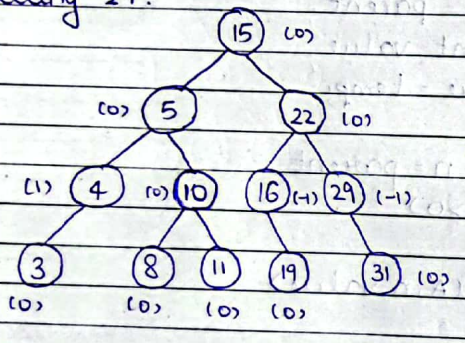
23/06/22.

* DELETION IN AVL TREE:

Eg) In the following tree, delete 27, 29, 16, 22 and 19

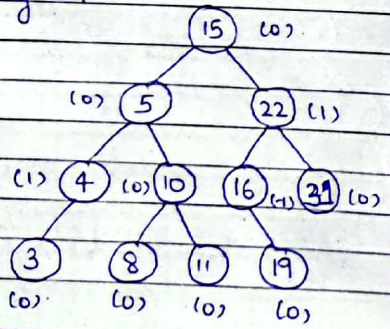


→ Deleting 27:



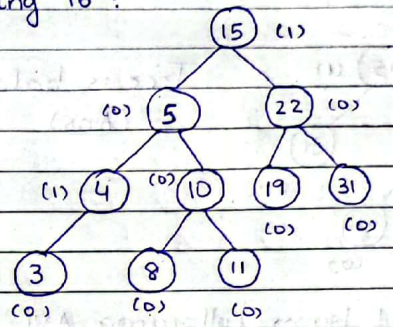
Tree is balanced.

Deleting 29:



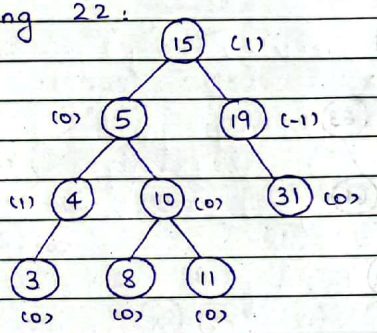
Tree is balanced.

Deleting 16:



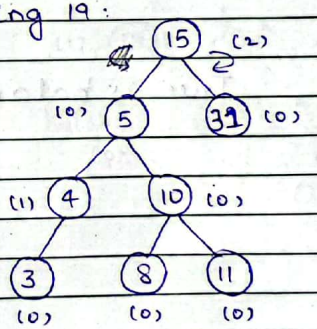
Tree is balanced.

Deleting 22:

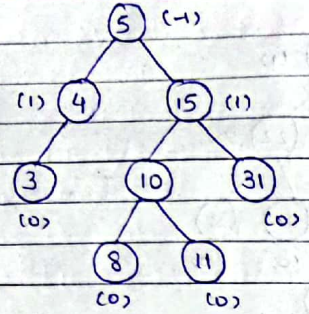


Tree is balanced.

Deleting 19:

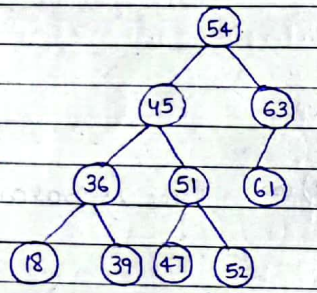


15 is unbalanced.
RO
LL Rotation

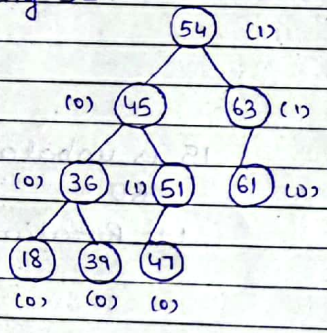


Tree is balanced (Ans)

Q. Delete 52, 36 and 61 from following AVL Tree:

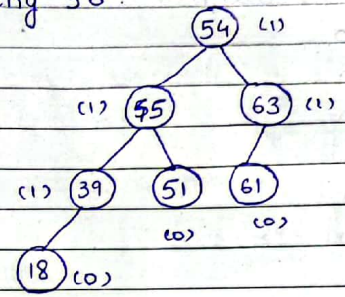


→ Deleting 52.



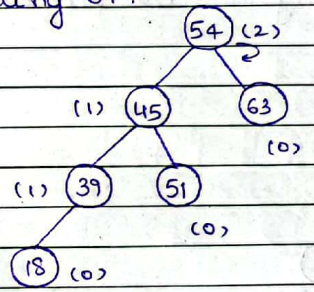
Tree is balanced.

Deleting 36:

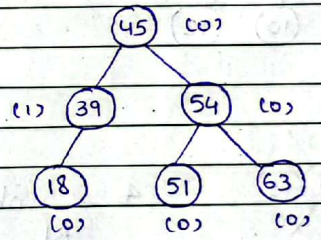


Tree is balanced.

Deleting 61:



54 is unbalanced.
RO Rotation
∴ RL rotation.



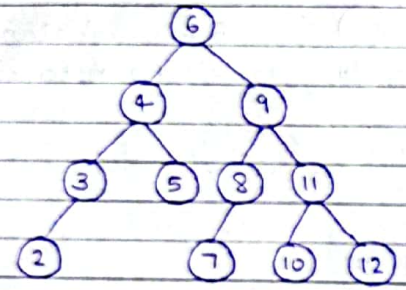
Tree is balanced (Ans)

23/06/'22.

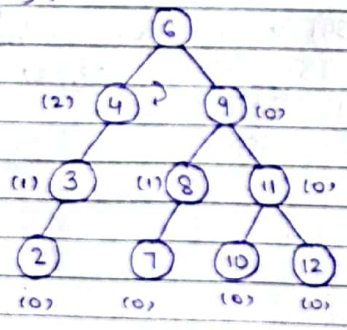
⊙ Deletion in AVL Tree Rotations:

L(0) : RR Rotation
L(1) : RL Rotation
L(-1) : RR Rotation
R(0) : LL Rotation
R(1) : LL Rotation
R(-1) : LR Rotation.

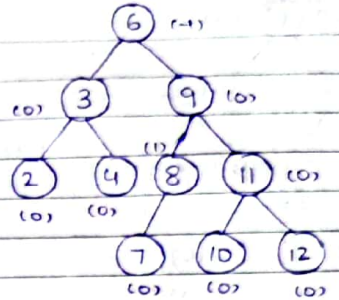
Q Delete 5, 9, 7 and 6 using inorder predecessor/successor in the following AVL Tree:



→ Deleting 5:

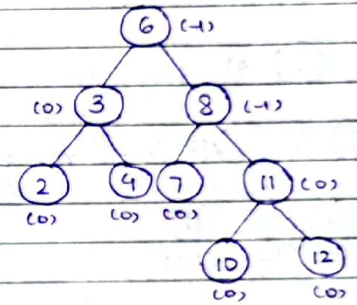


4 is unbalanced
R1
LL Rotation



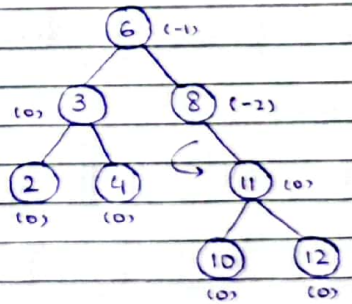
Tree is balanced

Deleting 9:

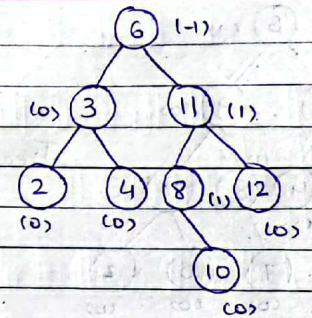


Tree is balanced

Deleting 7:

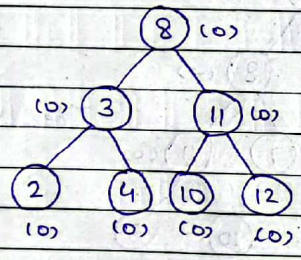


8 is unbalanced
LO
RR Rotation



Tree is balanced

Deleting 6:



Tree is balanced
(Ans)

25/06/22

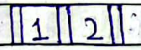
* B TREE / M-WAY TREE :

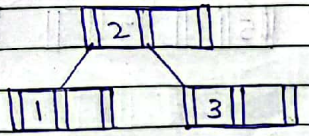
- B Tree stands for balanced tree which is a generalisation of binary search tree.
- It is a self-balanced tree in which each node can have more than two child nodes and multiple keys.
- A tree to be balanced, should satisfy certain properties :

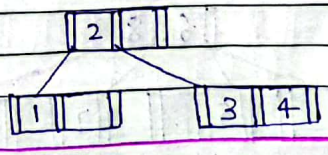
(1) All the leaf nodes should be at the same level.
(2) All the nodes except the root node must have at-least $\left[\frac{m}{2}\right]-1$ keys and can have maximum $[m-1]$ keys.
(3) Each node has a minimum of $\frac{m}{2}$ child nodes and can have maximum m child nodes.
(4) All the keys must be in ascending order.

Q. Insert elements 1-12 in a 3-way tree.

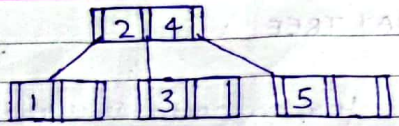
STEP 1: 

STEP 2: 

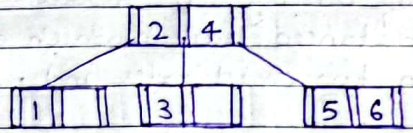
STEP 3: 

STEP 4: 

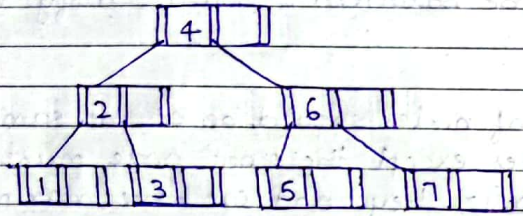
STEP 5:



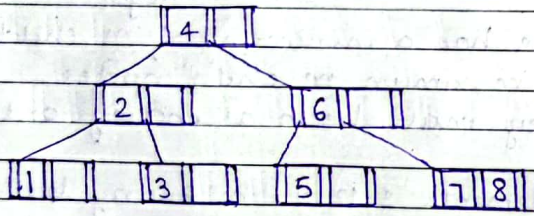
STEP 6:



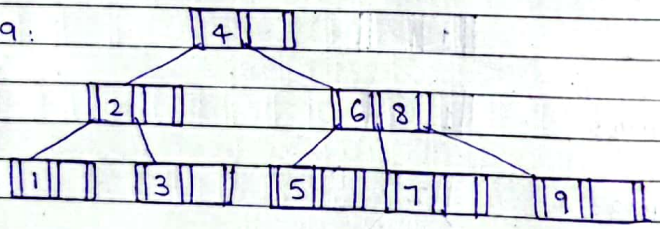
STEP 7:



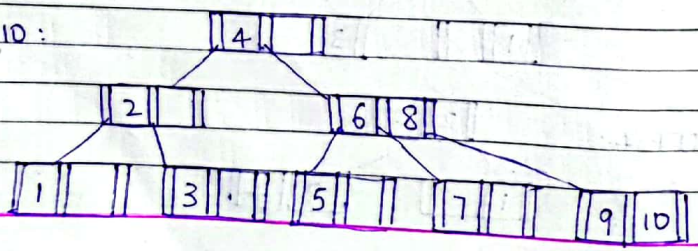
STEP 8:



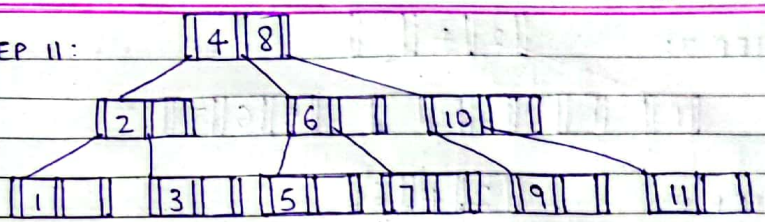
STEP 9:



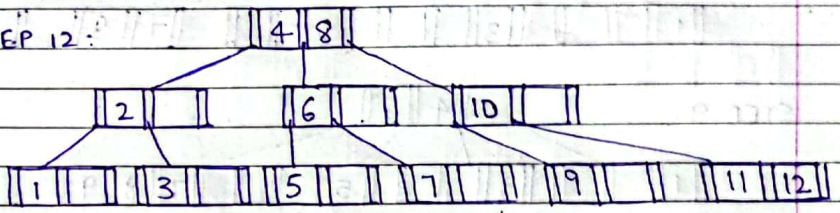
STEP 10:



STEP 11:



STEP 12:



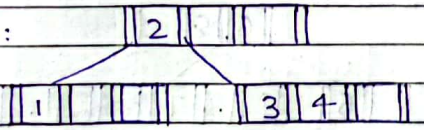
Q. Insert elements 1-12 in a 4-way tree:

STEP 1:

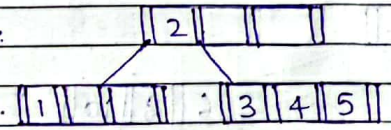
STEP 2:

STEP 3:

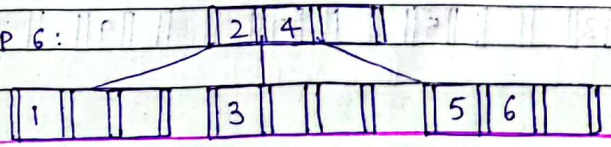
STEP 4:



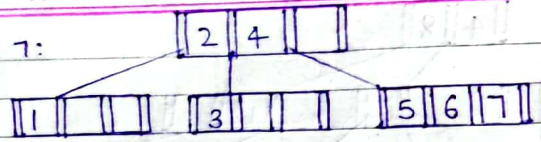
STEP 5:



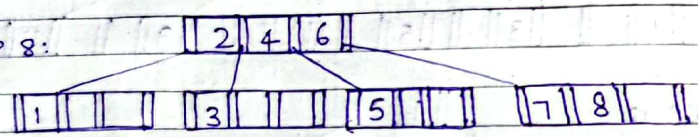
STEP 6:



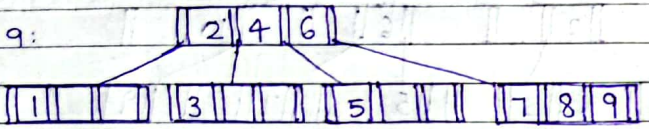
STEP 7:



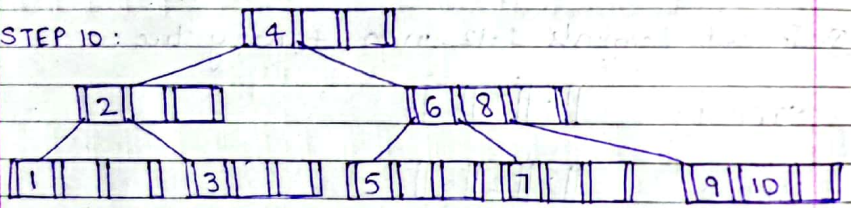
STEP 8:



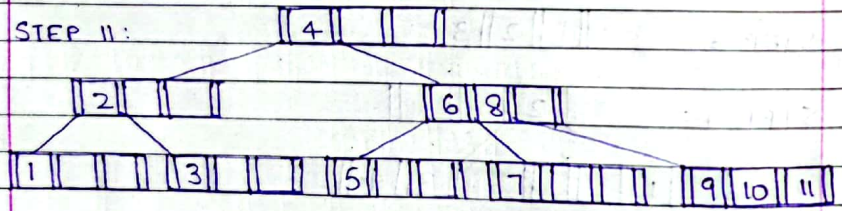
STEP 9:



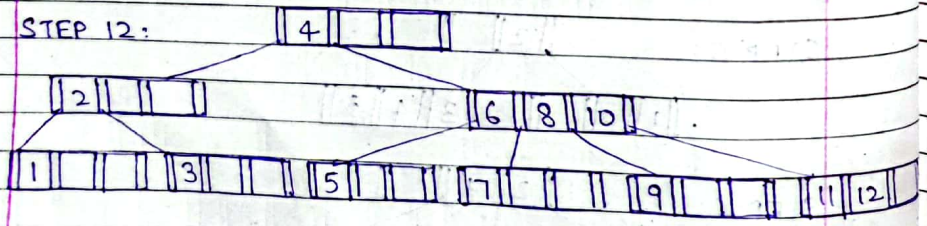
STEP 10:



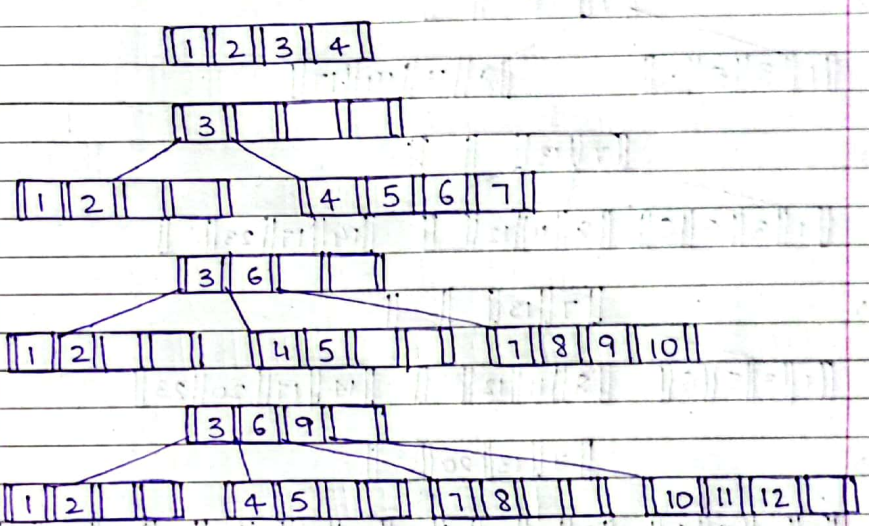
STEP 11:



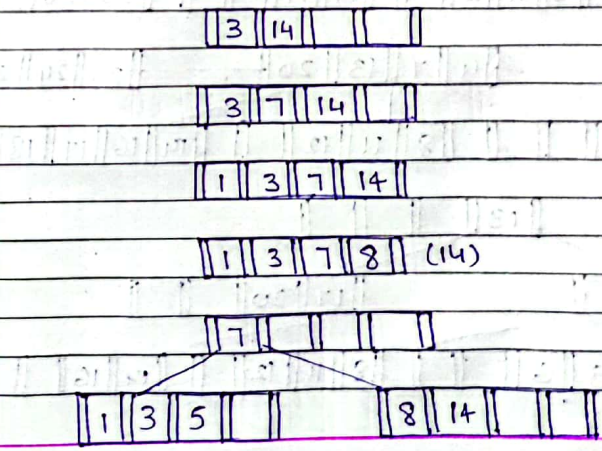
STEP 12:

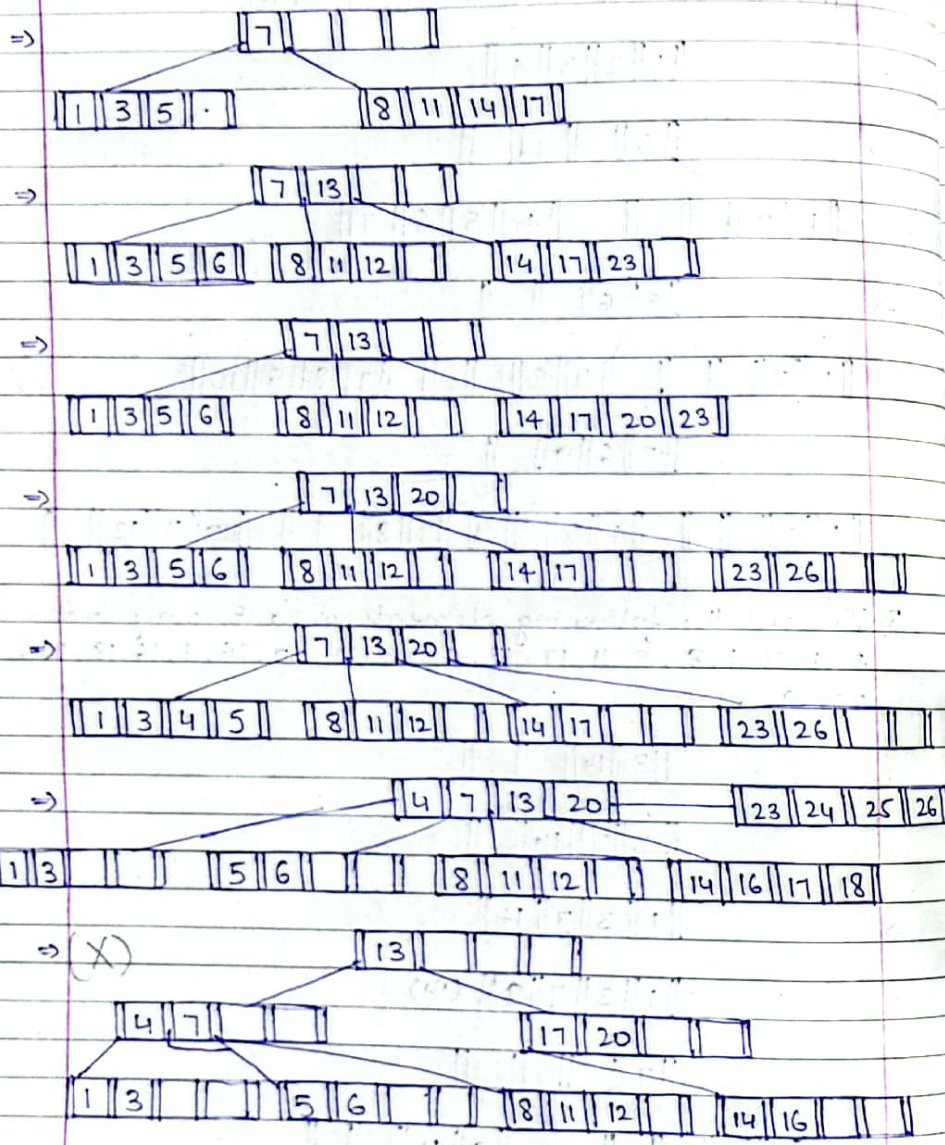


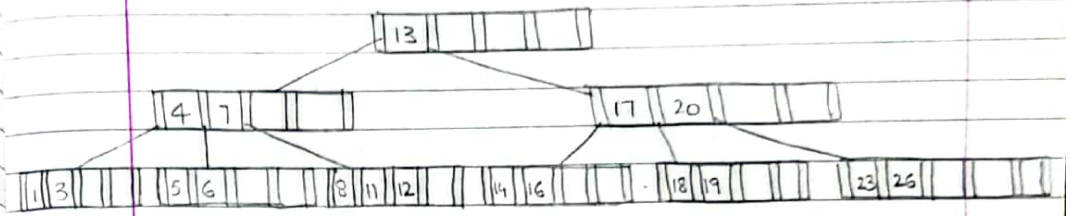
Q. Insert elements 1-12 in a 5-way tree



Q. Insert the following elements in a 5-way tree:
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19




7
1 3 5
8 11 14 17
7 13
1 3 5 6
8 11 12
14 17 23
7 13
1 3 5 6
8 11 12
14 17 20 23
7 13 20
1 3 5 6
8 11 12
14 17
23 26
7 13 20
1 3 4 5
8 11 12
14 17
23 26
4 7 13 20
23 24 25 26
1 3
5 6
8 11 12
14 16 17 18
(X)
13
4 7
17 20
1 3
5 6
8 11 12
14 16


13
4 7
17 20
1 3
5 6
8 11 12
14 16
18 19
23 26