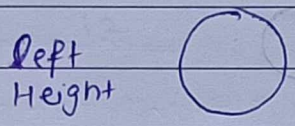
Lecture - 49

AVL and Heap

Balanced binary search tree have some properties of AVL tree.



obs (Left Height - right Height) > 1

↳ then unbalance

Then How to know which unbalance is used Rotation :

- → Left - left
- → Left - Right
- → Right - Right
- → Right - left

This is done by :

We can initialise a variable Balance.

Balance = left Height - Right Height.

if Balance > 1

↳ then unbalance is in left side

if Balance < -1

↳ then unbalance is in right side.

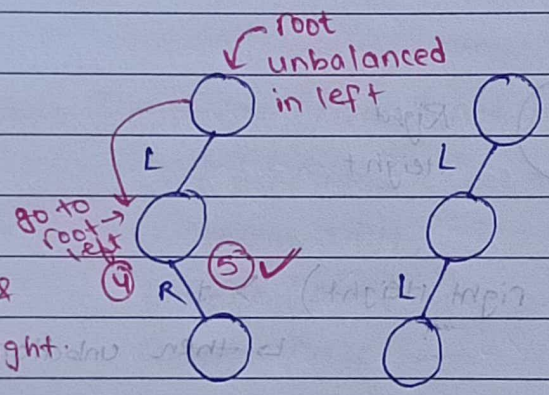
Till now, We know that either left or right balancing happen.

If left then

↳ left - left

↳ left - right

how we know about this.



ask left & right height.

Second place find करने के लिए जिधर ज्यादा होगा उधर ही unbalance होगा।

* 17

Height = 0 (17) Height = 0

* 15

Height = 1 (17) Height = 0

(15) Balanced.

* 12

Ht = 2 17 Ht = 0

L/ 2 - 0 = 2 (unbalanced)

1 15 0 > 1 (left)

L/

12

L - L unbalance

↓

So balance this.

15

/ \

12 17

→ 18 1 15 Ht = 3 2 (1 - 2) = -1 (Balanced)

/ \

12 0 17 Ht = 2

1 (0 - 1) balanced.

\

18 Ht = 0 + 1

→ 20 lt = 1 15 Ht = 4 Rt = 3 (1 - 3) = -2 (Right unbalance)

/ \

12 lt = 0 17 Ht = 3 Rt = 2 (0 - 2) = -2 unbalanced.

\R

R - R unbalance. lt = 0 18 Ht = 2 Rt = 1 (0 - 1) = -1 balanced

\R

20 Ht = 1

lt = 0 Rt = 0

Balance

15

/ \

12 18

/ \

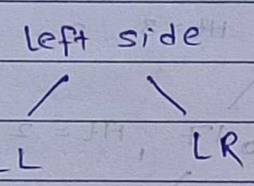
17 20

1. First Enter Element in our tree.

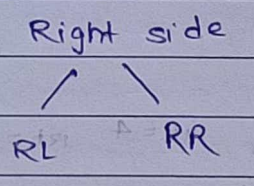
2. Either element goes in left or right till reach to NULL.

3. Till it finds NULL, create itself & return.

4. Check the Balancing:
   - First update the Height
   - Balance = Left - Right.
   - Balance > 1



   - Balance < -1



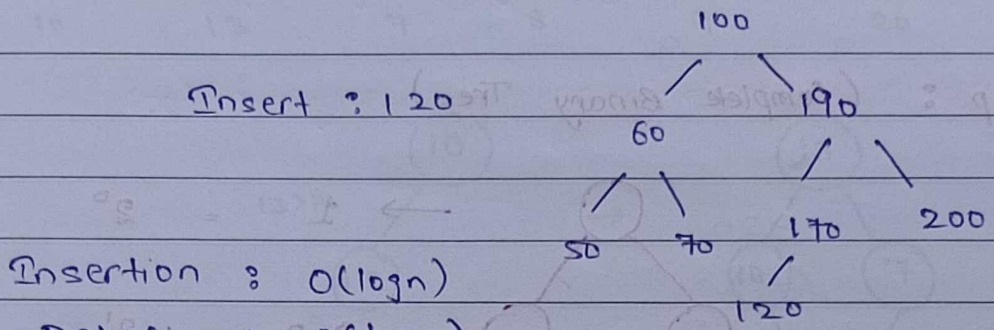
Balancing हमलोग वापस आते time check करते है।

```
Void updateHeight (Node * root){
    int leftHeight = getHeight (root -> left);
    int rightHeight = getHeight (root -> right);

    root -> height = 1 + max (leftHeight, rightHeight);

}

Node* Balance (Node * root) {
        if (!root)
            return NULL;

        updateHeight (root);

    int balance = getHeight (root -> left) - getHeight (root -> right);
        // Balance > 1 : Left unbalance
        if (balance > 1) {
            // Left Left
            if (getHeight (root -> left -> left) >= getHeight (
                                        root -> left -> right) {
                root = rotateRight (root);
            }
            // Left right
            else {
                root -> left = rotateLeft (root -> left);
                root = rotateRight (root);
            }
        else
        // Balance < -1 : Right unbalance.
        else if (balance < -1) {

            if (getHeight (root -> right -> right) >= getHeight (root -> right
                                                    -> left)) {
```

```
            root = rotateleft(root);
        }
        else{
            root->right = rotateright(root->right);
            root = rotateright(root);
        }
    }
    return root;
}

Void inorder( Node * root ){
        if(!root)
            return;
        inorder(root->left);
        Cout<<root->data<<" ";
        inorder(root->right);
}

Void preorder(Node * root){
        if(!root)
            return;

        Cout<<root->data<<" ";
        preorder(root->left);
        preorder(root->right);
}
```

* Balanced Binary search tree : O(log N)

Insertion :



Insert : 120

Insertion : O(logn)

Deletion : O(logn)

* Binary search tree :

Insertion : O(N)

Deletion : O(N)

* Binary tree :

Insertion : O(N)

Deletion : O(N)

Example :

Flight :

4 Airplanes have to land

Which lands first

	1	2	3	4
Emergency	6	10	8	2

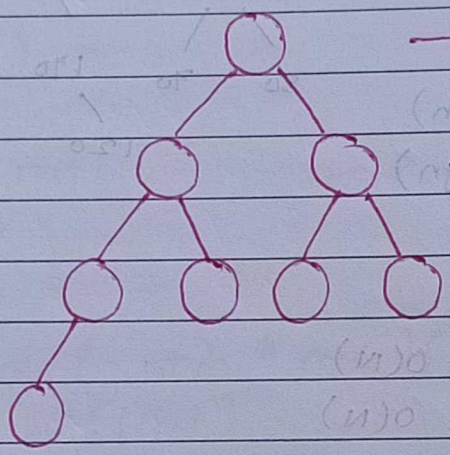
We give permission of landing by the help of Emergency factor.

Airplanes landed : 2 3 1 4

This is done by the help of priority basis.

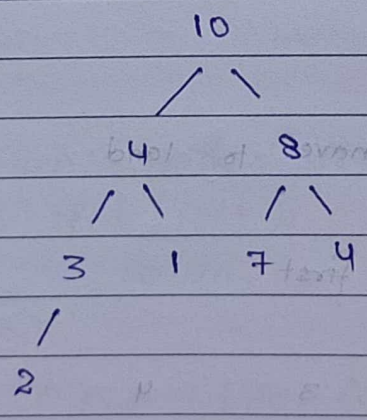
Those have high priority, they have land first.

* Heap : (Complete Binary Tree)

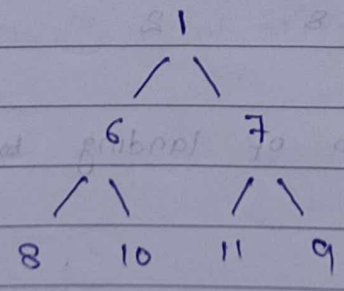


last level सिर्फ property hold Nahi karega.

Max Heap :



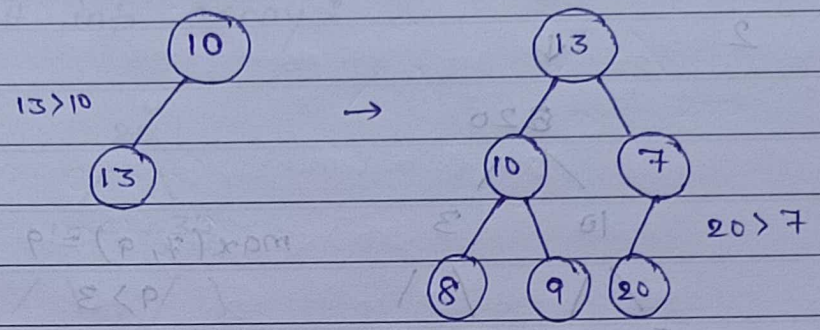
Min Heap :

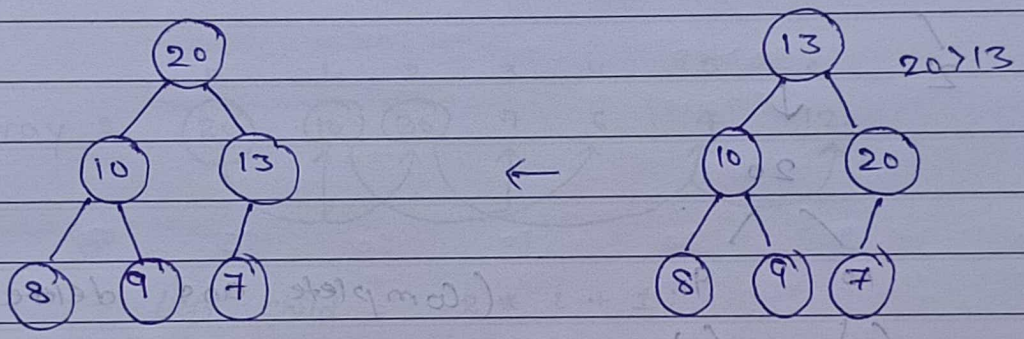


Max - Heap

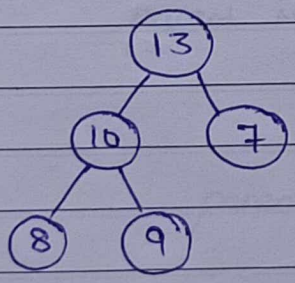
1	2	3	4	5	6
10	13	7	8	9	20

priority :



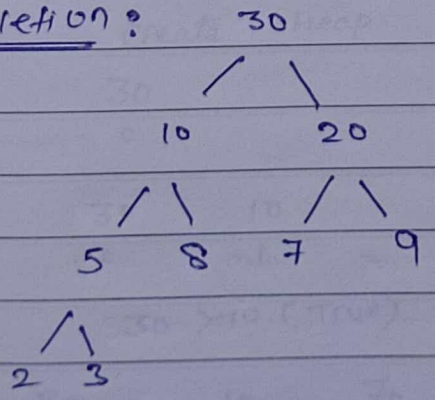


Delete 20



~~→ Not true because this is Not Complete~~

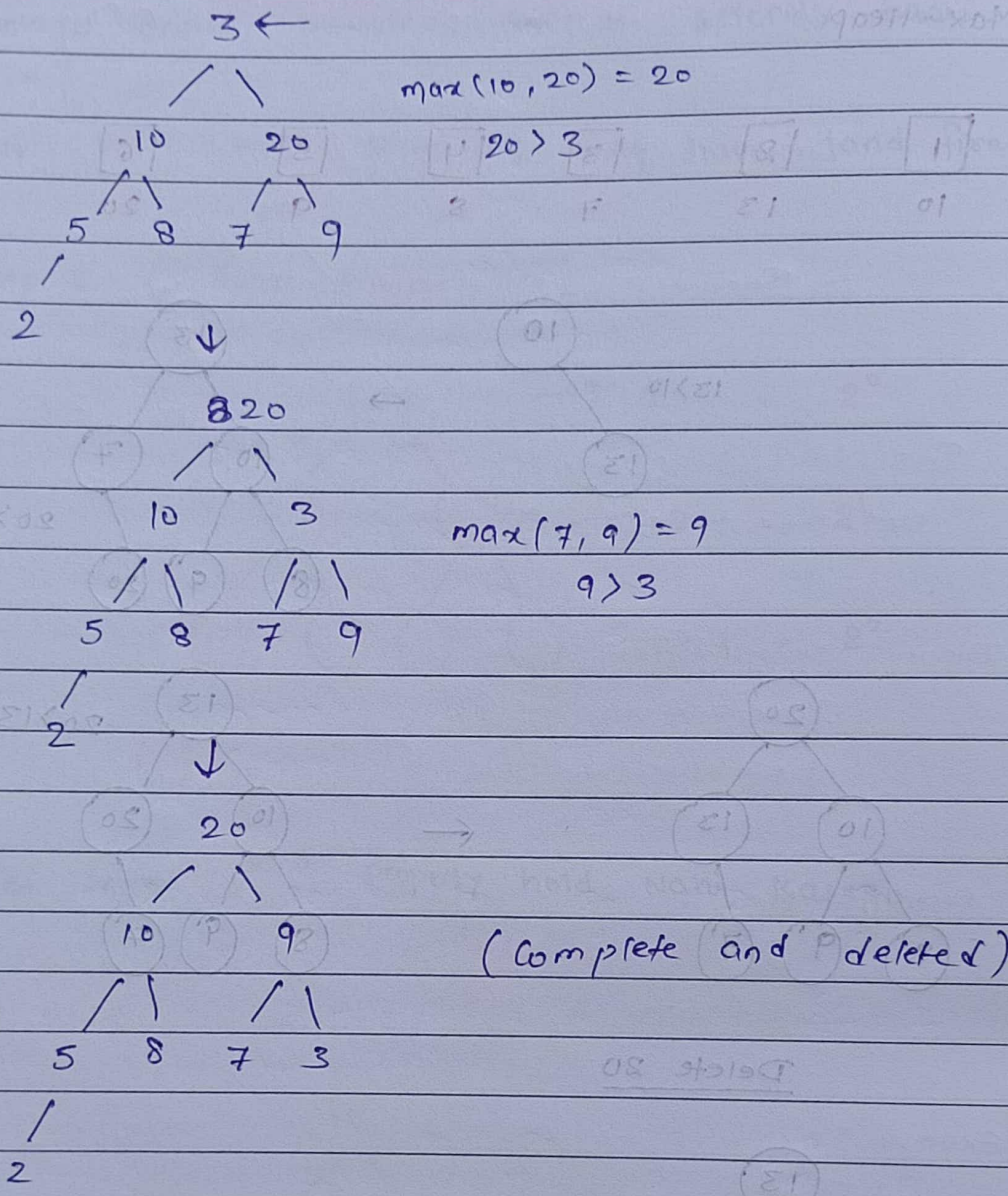
* Deletion :



Delete 30

How to do.

Delete 30 and put the last element of 3.

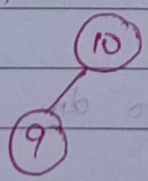


(Complete and deleted).

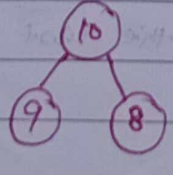
* **Insertion :**

Insert 10

Insert 9



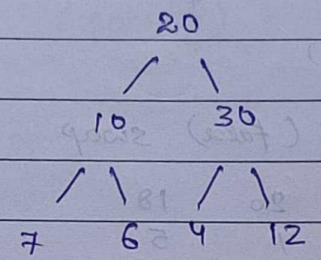
Insert 8

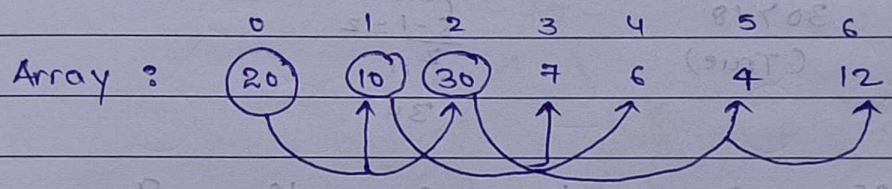


for Insertion We use level order traversal.

→ Better Approach:

Convert it into array:





Left child = 2 * i + 1;

Right child = 2 * i + 2;

parent Node : $\frac{(i-1)}{2}$;

parent Node >= Child Node.

[30 10 70 20 25 18 9 11]

Create Heap

→ 30
0

→ 30 10
0 1

30 >= 10 (True)

→ 30 10 70
0 1 2
↓
parent = 30

30 > 70 (false)
swap

→ 70 10 30 20
0 1 2 ↓
parent
||
10

10 > 20 (false) swap

→ 70 20 30 10 25
0 1 2 3 4

20 < 70 (True)

20 > 25 (false) swap

→ 70 25 30 10 20 18
0 1 2 3 4 5

30 > 18 (True)

parent
$\frac{5-1}{2} = 2$
||
30

→ 70 25 30 10 20 18 9
0 1 2 3 4 5 6

30 > 9 (True) $\frac{6-1}{2} = 2 = 30$

→ 70 25 30 10 20 18 9 11
0 1 2 3 4 5 6 7

10 > 11 (false) $\frac{7-1}{2} = 3 = (10)$

swap

→ 70 25 30 11 20 18 9 10