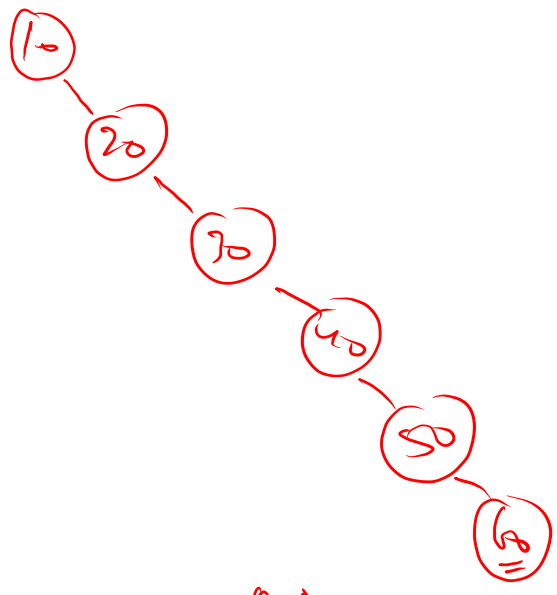
BST ✓

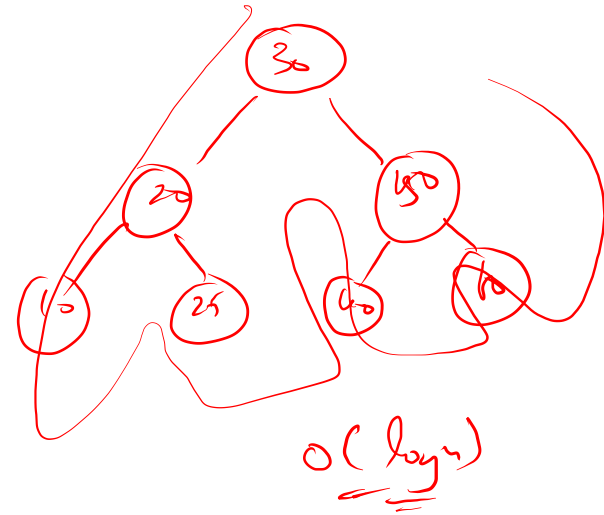
Deletion

Insert

Find

$O(h) \Rightarrow O(n)$





$O(\log n)$

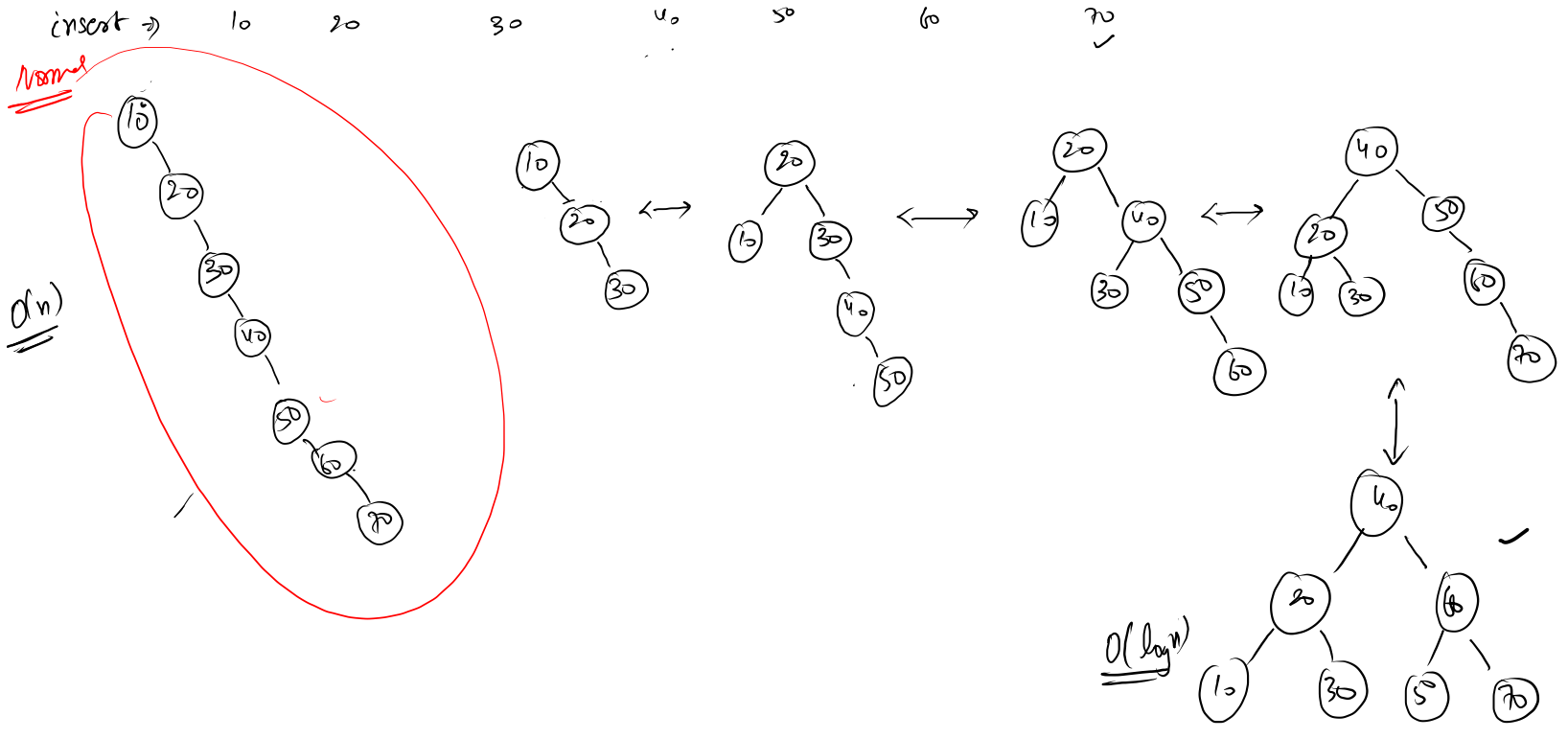
$O(n)$

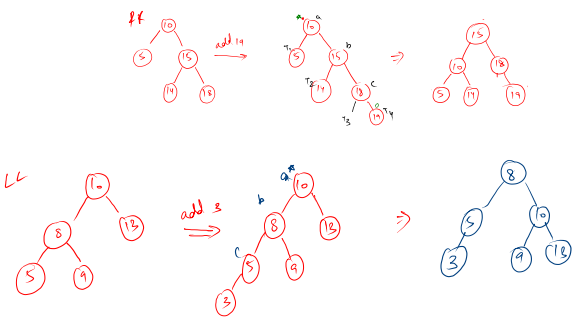
BST

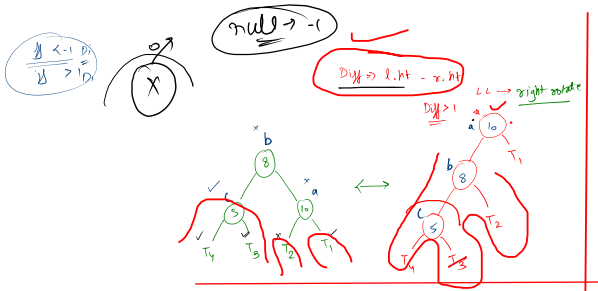
$O(h) \simeq O(n)$

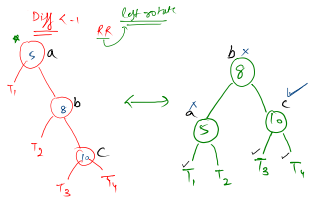
AVL ① Self balancing BST → insert, delete

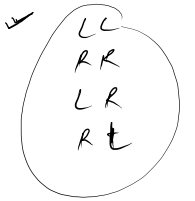
② Used to prevent skewed Binary search Tree

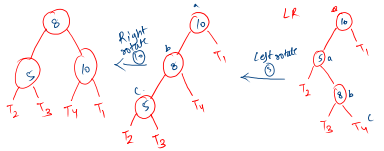


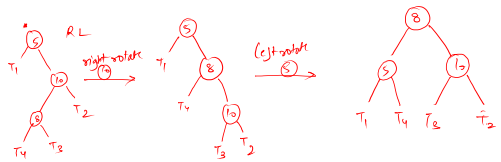

LL
add 3


null → -1
Diff => l.ht - r.ht
LL → Right rotate
Diff > 1

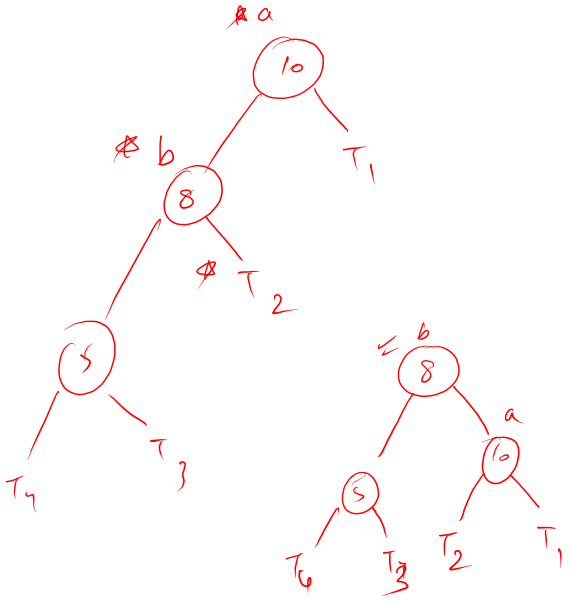

Diff < -1
RR
Left rotate

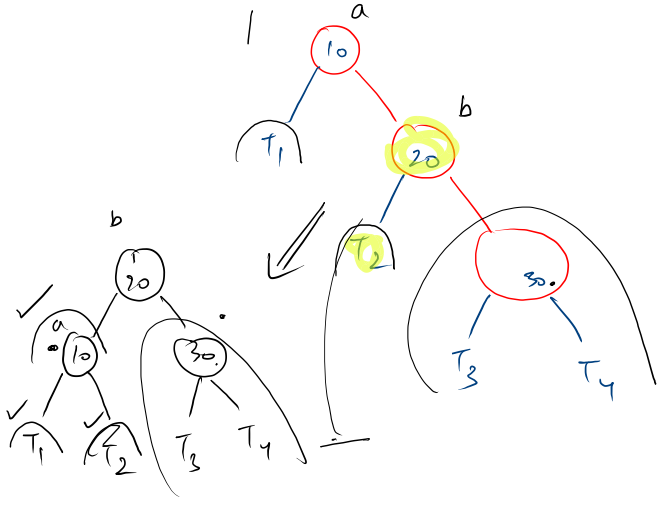

LL
RR
LR
RL


Right rotate
Left rotate
LR


RL
right rotate
Left rotate

```java
public Node leftRotate(Node a){
    Node b = a.right;
    Node t2 = b.left;

    b.left = a;
    a.right = t2;

    a.height = Math.max(height(a.left), height(a.right))+1;
    b.height = Math.max(height(b.left), height(b.right))+1;
    return b;
}

public Node rightRotate(Node a){
    Node b = a.left;
    Node t2 = b.right;

    b.right = a;
    a.left = t2;

    a.height = Math.max(height(a.left), height(a.right))+1;
    b.height = Math.max(height(b.left), height(b.right))+1;
    return b;
}
```





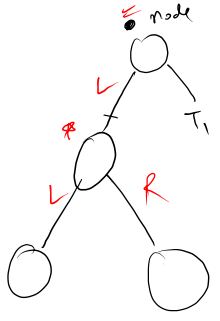
```java
public  Node insertToAVL(Node node,int data)
{
    if(node == null){
        return new Node(data);
    }
    if(data < node.data){
        node.left = insertToAVL(node.left, data);
    }else if(node.data < data){
        node.right = insertToAVL(node.right, data);
    }
    int lht = height(node.left) , rht = height(node.right);
    node.height = Math.max(lht, rht)+1;
    int diff = lht - rht;

    Node newRoot = node;
    if(diff > 1){ // left
        if(data < node.left.data){ // LL
            newRoot = rightRotate(node);
        }else if(data > node.left.data){ // LR
            node.left = leftRotate(node.left);
            newRoot = rightRotate(node);
        }
    }else if(diff < -1){ // right
        if(data < node.right.data){ // RL
            node.right = rightRotate(node.right);
            newRoot = leftRotate(node);
        }else if(data > node.right.data){ // RR
           newRoot = leftRotate(node);
        }
    }
    return newRoot;
}
```

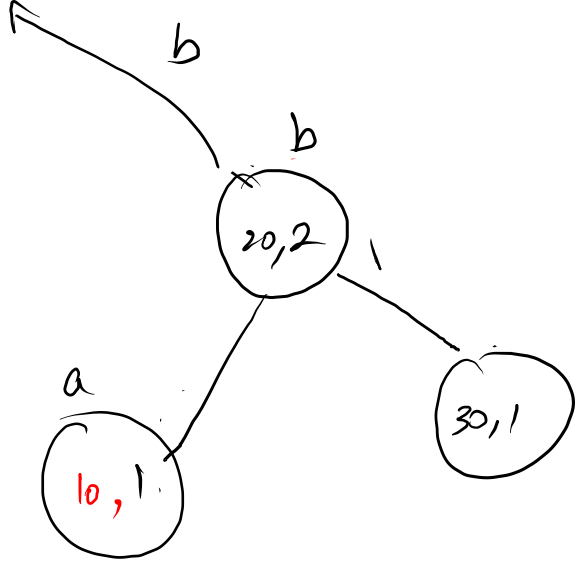
```java
public  Node insertToAVL(Node node,int data)
{
    if(node == null){
        return new Node(data);
    }
    if(data < node.data){
        node.left = insertToAVL(node.left, data);
    }else if(node.data < data){
        node.right = insertToAVL(node.right, data);
    }
    int lht = height(node.left) , rht = height(node.right);
    node.height = Math.max(lht, rht)+1;
    int diff = lht - rht;

    Node newRoot = node;
    if(diff > 1){ // left
        if(data < node.left.data){ // LL
            newRoot = rightRotate(node);
        }else if(data > node.left.data){ // LR
            node.left = leftRotate(node.left);
            newRoot = rightRotate(node);
        }
    }else if(diff < -1){ // right
        if(data < node.right.data){ // RL
            node.right = rightRotate(node.right);
            newRoot = leftRotate(node);
        }else if(data > node.right.data){ // RR
           newRoot = leftRotate(node);
        }
    }
    return newRoot;
}
```

```java
public int height(Node node){
    if(node == null){
        return 0;
    }else{
        return node.height;
    }
}
```

height

nR=10





insert = 20

= 30

S.C. :) O(n)

Ans T.C. :) O(h) ⇒ O(logn)

```java
public Node rightRotate(Node a){
    Node b = a.left;
    Node t2 = b.right;

    b.right = a;
    a.left = t2;

    a.height = Math.max(height(a.left), height(a.right))+1;
    b.height = Math.max(height(b.left), height(b.right))+1;
    return b;
}
```

```java
public Node leftRotate(Node a){
    Node b = a.right;
    Node t2 = b.left;

    b.left = a;
    a.right = t2;

    a.height = Math.max(height(a.left), height(a.right))+1;
    b.height = Math.max(height(b.left), height(b.right))+1;
    return b;
}
```

BST -> Delete

- → leaf ✓✓
- → Single child
- → 2 child

remove 20