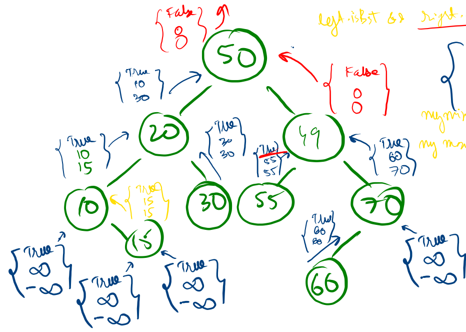


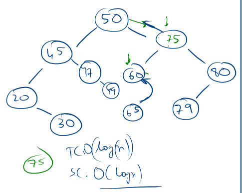
left.isBst && right.isBST && node.data > left.max && node.data < right.min

{ isBST, min, max }

my min = min(node.data, left.min)

my max = max(node.data, right.max)

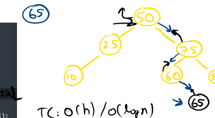
```
public TreeNode deleteNode(TreeNode node, int val){
    // WRITE YOUR CODE HERE
    if(node == null) return null;
    if(node.val < val){//the value i want to delete is in the right subtree
        node.right = deleteNode(node.right,val);
    }else if(node.val > val){//the value i want to delete is in the left subtree
        node.left = deleteNode(node.left,val);
    }else{//i am standing on the node which needs to be deleted

        if(node.left == null && node.right == null){//no children
            return null;
        }else if(node.left == null && node.right != null){
            return node.right;
        }else if(node.left != null && node.right == null){
            return node.left;
        }else{//i am having both children
            //get the left  subtrees max
            int leftMax = max(node.left); //returns the max value in left bst
            node.val = leftMax;
            node.left = deleteNode(node.left,leftMax);
        }

    }
    return node;
}
```



TC: O(log(n))

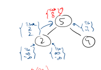
SC: O(log n)

```
public Node insertNode(Node root, int val){
    // WRITE YOUR CODE HERE
    if(root == null){
        Node nn = new  Node(val);
        return nn;
    }
    if(root.val == val) return root; //DO NOTHING
    if(val > root.val){
        root.right = insertNode(root.right,val);
    }else if(val < root.val){
        root.left = insertNode(root.left,val);
    }
    return root;
}
```



TC: O(h) / O(log n)

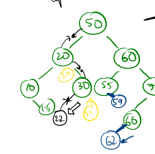
SC: O(h) / O(log n)



tc = O(n)

sc = O(log n)

update root of left subtree



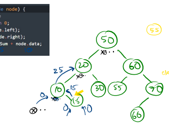
1) Find a vacant spot

2) Create a node and return.

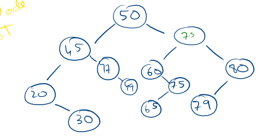
```
if(node==null){
    Node nn = new Node(val)
    return nn;
}
if(node.data < val){
    node.right = insert(node.r, val)
} else {
    node.left = insert(node.l, val)
}
return node
```

```
if(node==null) return false
if(target == node.data) return true
else if (target > node.data)
    return find(node.right)
else if(target < node.data){
    return find(node.left);
}
```

```
public static int sum(Node node) {
    // write your code here
    if(node== null) return 0;
    int leftSum = sum(node.left);
    int rightSum = sum(node.right);
    return leftSum + rightSum + node.data;
}
```



Delete Node in BST



left's largest

right's smallest

→ Node is having no child: return null

→ Node is having only left child: return left child

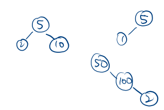
→ Node is having only right child: return right child

→ Node is having both children

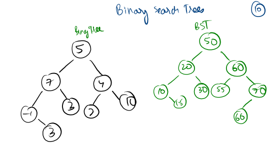
→ val of left's largest → x

→ Replace my v

→ delete node(node.left, x)



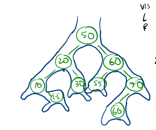
Binary Search Trees



Properties

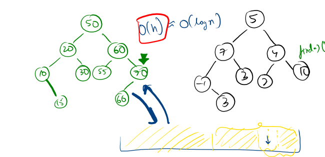
I) for every node the values of children in left subtree will be strictly smaller and for right subtree they are strictly bigger

II) In a BST the inorder is sorted in asc order



2nd In: 10, 15, 20, 30, 50, 55, 60, 66, 70

abs(depth of left - depth of right) <= 1



O(h) = O(log n)

find → O(n)

1024 → 10

$\log_2 1024 = 10$

boolean ? val if boolean is true : False