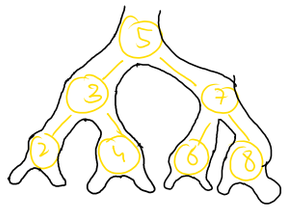
```java
public static int sum(Node node) {
  // write your code here
  if(node == null) return 0;
  int left = sum(node.left);
  int right = sum(node.right);
  return left + right + node.data;
  // return node == null?0:sum(no
}
```



Pre order: 5 3 2 4 7 6 8

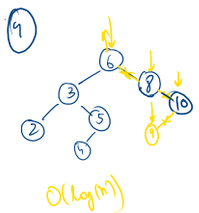
Post order: 2 4 3 6 8 7

```java
public static int sum(Node node) {
  // write your code here
    if(node == null) return 0;
    int left = size(node.left);
    int right = size(node.right);
    return left + right + node.data;
    // return node == null?0:sum(no
}
```

```java
public static int size(Node node) {
  // write your code here
  if(node == null) return 0;
  int left = size(node.left);
  int right = size(node.right);
  return left + right + 1;
    // return node == null?0:size(no
}
```

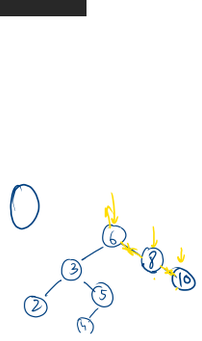
0 1 0 1 0 1

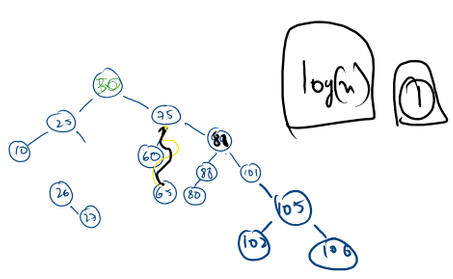
```java
public TreeNode insertIntoBST(TreeNode root, int val){
    if(root == null){
        TreeNode newNode = new TreeNode(val);
        return newNode;
    }

    if(val < root.val){
        root.left = insertIntoBST(root.left,val);
        return root;
    }else{
        root.right = insertIntoBST(root.right,val);
        return root;
    }
}
```



O(log n)

```java
public static int max(Node node) {
  // write your code here
  Node temp = node;
  while(temp.right != null) temp = temp.right;
  return temp.data;
}

public static int min(Node node) {
  // write your code here
  Node temp = node;
  while(temp.left != null) temp = temp.left;
  return temp.data;
}
```





log(n) (1)

→ Leaf Node

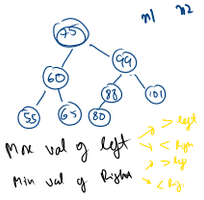
Return Null

→ Only Left Child

Node.left

→ Only Right Child

Node.right

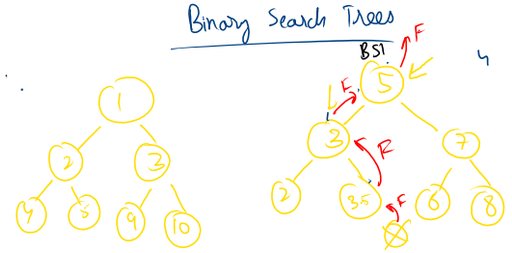


Max val of left

Min val of Right

CODE

Binary Search Trees

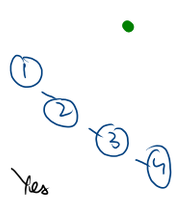


BST

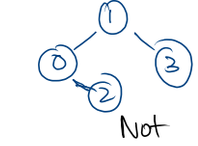
① For every node, left subtree has smaller values, right subtree has larger values

② Inorder will be sorted

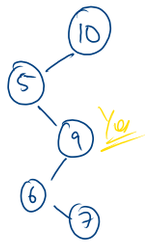
Inorder: 2 3 4 5 6 7 8



Yes



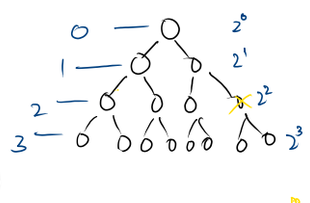
Not



Yes

LCA

Binary



TC ⇒ height ⇒ log(n)

2 3 4 5 (7 8 9 10)

$2^0 = 1024 \Rightarrow 10$

log(n) (1)

→ (Node, val)

if(val < Node) {

  goto left

} else if(val > Node) {

  goto Right

} else {

  if(L.C. == Null & R.C == Null) ret Null

  else if(L.C. == NULL) retu R.C

  else if(R.C == Null) retu L.C.

  else { // Both are k

    int minRight = min(R.C)

    Root.val = minRight

    Root.right = deleteBST(Root.right, minRight)

    retu Root;

} }