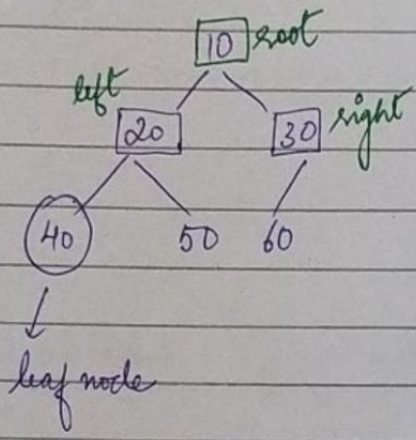
Binary Tree

* At most 2 children
*



leaf node

30 — right == null

* Basic structure

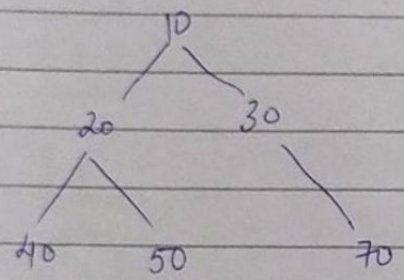
```
public class BinaryTree {                    → outer class

    private class Node {                     → Inner class
        int data;
        Node left;
        Node right;
    }

    private Node root;

}
```

* Take Input (parent, boolean)

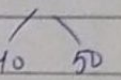
1K, true → left
3K, false → right

BT Construction → Code Github

Q1. BT → Size
Max
Find
Height



⇒ 2k ki rec. call : 20 subtree ke ~~height~~ size

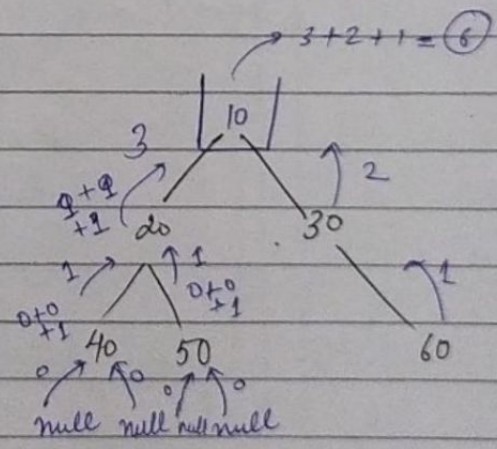


⇒ S"by right subtree ka size

⇒ Dono subtrees ke size ko add krke +1 krdo apne node ke naam ka.

Code :- SIZE OF BT

```
private int size (Node node) {
    if ( node == null ) {
        return 0;
    }
    int ls = size ( node.left);
    int rs = size ( node.right);

    return ls + rs + 1;

}

public int size () {
    return size(root);
}
```



MAX OF BT

→ left node se puchenge ki lere subtree me max value kya hai
→ right node se puchenge same

Code :-

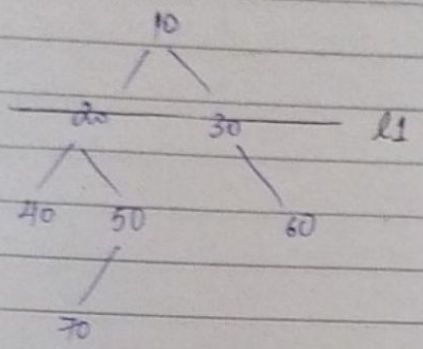
```
private int max (Node node){
    if(node == null) {
        return Integer.MIN_VALUE;
    }
    int lm  = max (node.left);
    int rm  = max (node.right);

    return Math.max( node.data, Math.max (lm, rm) );
}
public int max () {
    return max (root);
}
```

FIND IN BT

```
private boolean find (Node node, int item) {
    if( node == null ){
        return false;
    }
    if( node.data == item) {
        return true;
    }
    boolean lr  = find( node.left, item);
    boolean rr  = find (node.right, item);

    return lf || rf
}
```

```
public boolean find(
        int item){
return find(root, item);
}
```

HEIGHT OF BT



$l_0 \rightarrow$ leaf

$l_1 \rightarrow$ leaf $\}$ + 1

$2k, 3k$

smaller problems

$\left\{ \begin{array}{l} 2 \leftarrow 2k \rightarrow \text{leaf} \\ 1 \leftarrow 3k \rightarrow \text{leaf} \end{array} \right\} \max^m$

$\max^m = 2$

$l_1 \rightarrow$ leaf : 2

$2 + 1 = 3 \rightarrow$ height

$(l_1 \rightarrow l_0)$

Code :-

```
private int ht(Node node) {
    if(node == null) {
        return -1;
    }
    int lh = ht(node.left);
    int rh = ht(node.right);
    return Math.max(lh, rh) + 1;
}
public int ht() {
    return ht(root);
}
```

(Agar return o kaise toh $\max(lh, rh) + 1 = 0 + 1 = 1$ ho jata which is not right)

TC:

$$T(n) = 2T(n/2) + 1$$

$$T(n/2) = 2T(n/4) + 1 \quad \times 2$$

$$T(n/4) = 2T(n/8) + 1 \quad \times 2^2$$

$$\vdots$$

$$T(1) = 1$$

$$\hookrightarrow n T\left(\frac{n}{n}\right) = n$$

GP sum formula

$$\hookrightarrow \frac{a(r^n - 1)}{r - 1}$$

$$n = 2^{\log_2 n}$$

$$2^0 + 2^1 + 2^2 + 2^3 \; - - - + 2^{\log_2 n}$$

$$T(n) = \frac{2^{\log_2 n + 1} - 1}{2 - 1} = 2^{\log_2 n + 1} - 1$$

$$= n$$

$$T(n) = O(n)$$

DIAMETER

```
private int diameter2(Node node) {
    if (node == null) {
        return 0;
    }

    // max dist b/w 2 leaf nodes might lie in the left subtree : left ka diameter
    int ld = diameter2(node.left);
    int rd = diameter2(node.right); // right ka diameter

    // self node might be the root of diameter
    int sd = ht(node.left) + ht(node.right) + 2;

    return Math.max(sd, Math.max(ld, rd));
}
```

TC:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$

$$T\left(n/2\right) = 4T\left(n/4\right) + 1 \quad \times 4$$

$$\times 4^2$$

$$T(n/n) = 1$$

$$\left(\right), \; T\left(\frac{n}{2^{\log_2 n}}\right)$$

$$4^0 + 4^1 + 4^2 + \cdots + 4^{\log_2 n}$$

$$\frac{4^{\log_2 n + 1} - 1}{4 - 1} = \frac{4 \cdot 4^{\log_2 n} - 1}{3} \quad \text{(constants hatado)}$$

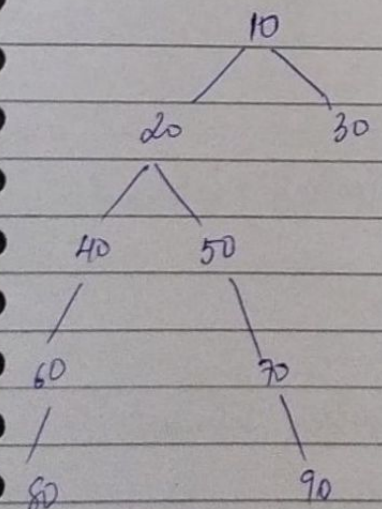
$$= 4^{\log_2 n}$$

$$= n^2$$

$O(n^2)$: TC

Diameter Better Approach

$n^2 \longrightarrow n$



$T(n) = 4T\left(\frac{n}{2}\right) + 1$

↳ $O(N^2)$

$T(n) = 2T\left(\frac{n}{2}\right) + 1$

↳ $O(N)$

* Ish kisi tarah recurrence relation ham ye wala bana dein, toh kaam ho jaayega.

(Toh bas sirf 4 calls dena hain)

Is Balanced

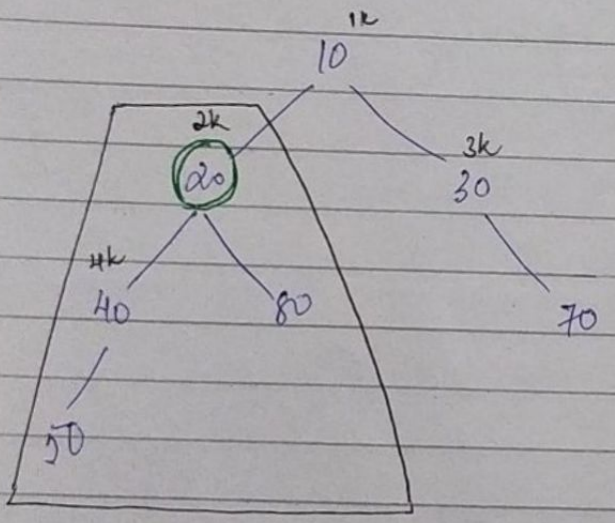
-1, 0, 1

Balanced : every node should've a bf of -1, 0, 1

Approach 1 :-

bf : right subtree height − left subtree height

Ek imp. thing tree me :-



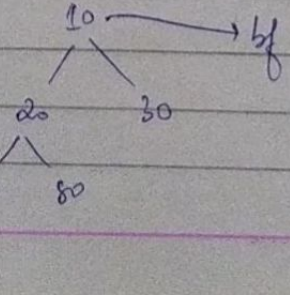
* 2k memory address pe, 2k wale pure subtree ki information stored hogi.
* Aisa nhi hoga ki, 2k memory address pe sirf (20) wali node ki information stored hogi.

① Yeh 2k ke rec call batayegi ki uske neeche ka tree (subtree) balanced hai ki nhi → smaller problem

② 30 → 3k wali rec call batayegi ki uske neeche ka subtree balanced hai ki nhi → smaller problem

③ Then we'll also check ki 10 wali node balanced hai ki nhi

Eg suppose tree aisa hota toh :



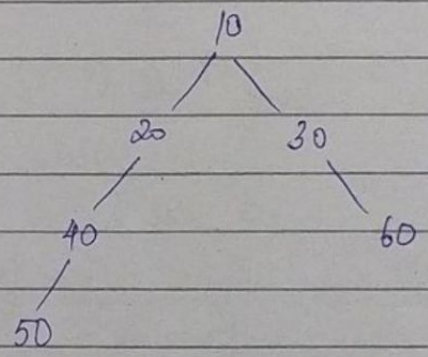
∴ Tree is not balanced

Summary :-

3 mandatory checks → 1. left st bal
2. right st bal
3. node bal.

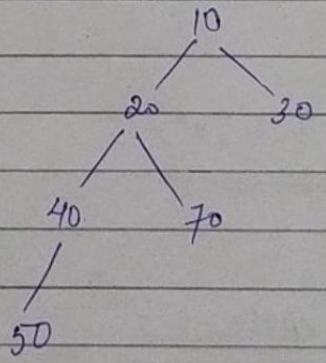
Why ? Let's see some eg.

→ Suppose we don't check left st bal



Here left st is not balanced
Right st ✓
node ✓

→ Suppose we don't check node



Here left st ✓
Right st ✓
node X

Code :-

```
public boolean isBalanced() {
    return isBalanced(root);
}
```

```
private boolean isBalanced (Node node) {
    if (node == null) {
        return true;
    }
    boolean lb = isBalanced (node.left);
    boolean rb = isBalanced (node.right);
    int hf = ht(node.left) - ht (node.right);
    if (lb && rb && hf >= -1 && hf <= 1) {
        return true;
    } else {
        return false;
    }
```

3.

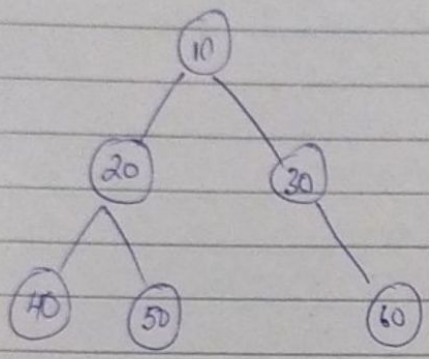
$TC = O(n^2)$ $\left(4T\left(\frac{n}{2}\right) + 1\right)$

Approach 2

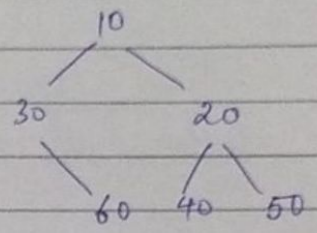
```
private class BalPair {
    boolean isBal = true;
    int ht = -1;
}

public boolean isBalanced (){
    return isBalanced (root).isBal;
}
```

```
private BalPair isBalanced(Node node) {
    if (node == null) {
        return new BalPair();
    }

    BalPair lbp = isBalanced(node.left);
    BalPair rbp = isBalanced(node.right);

    BalPair sbp = new BalPair();

    boolean lb = lbp.isBal;
    boolean rb = rbp.isBal;

    int bf = lbp.ht - rbp.ht;

    if (lb && rb && bf >= -1 && bf <= 1) {
        sbp.isBal = true;
    } else {
        sbp.isBal = false;
    }

    sbp.ht = Math.max(lbp.ht, rbp.ht) + 1;

    return sbp;
}
```

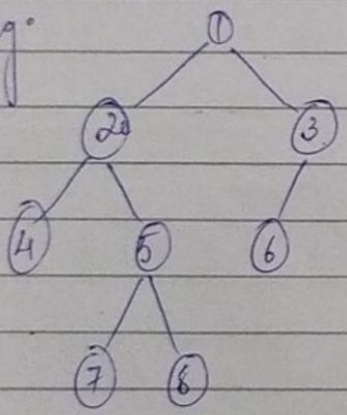
BT Flip Equivalent

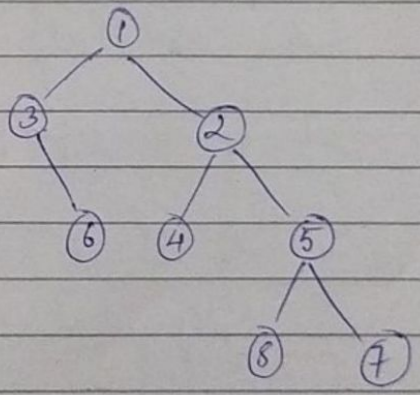


10 flip :



Eg.





Are they flip equivalent ?

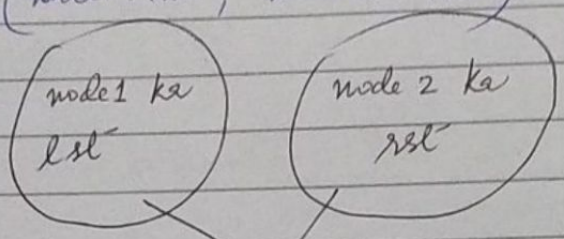
Yes

it flipped at nodes with 1, 3 and 5

* node 1. data == node 2. data
* 2 cases → Node 1 ko flip ✓
  flip nhi karaya ✗
* flip ✓ toh node 1 ka left subtree ko node 2 ke right subtree match karayenge &&
  node 1 ko right subtree & node 2 ka left st match hona chahiye.

* flip X (node1.lst, node2.lst) &&
(node1.rst, node2.rst)

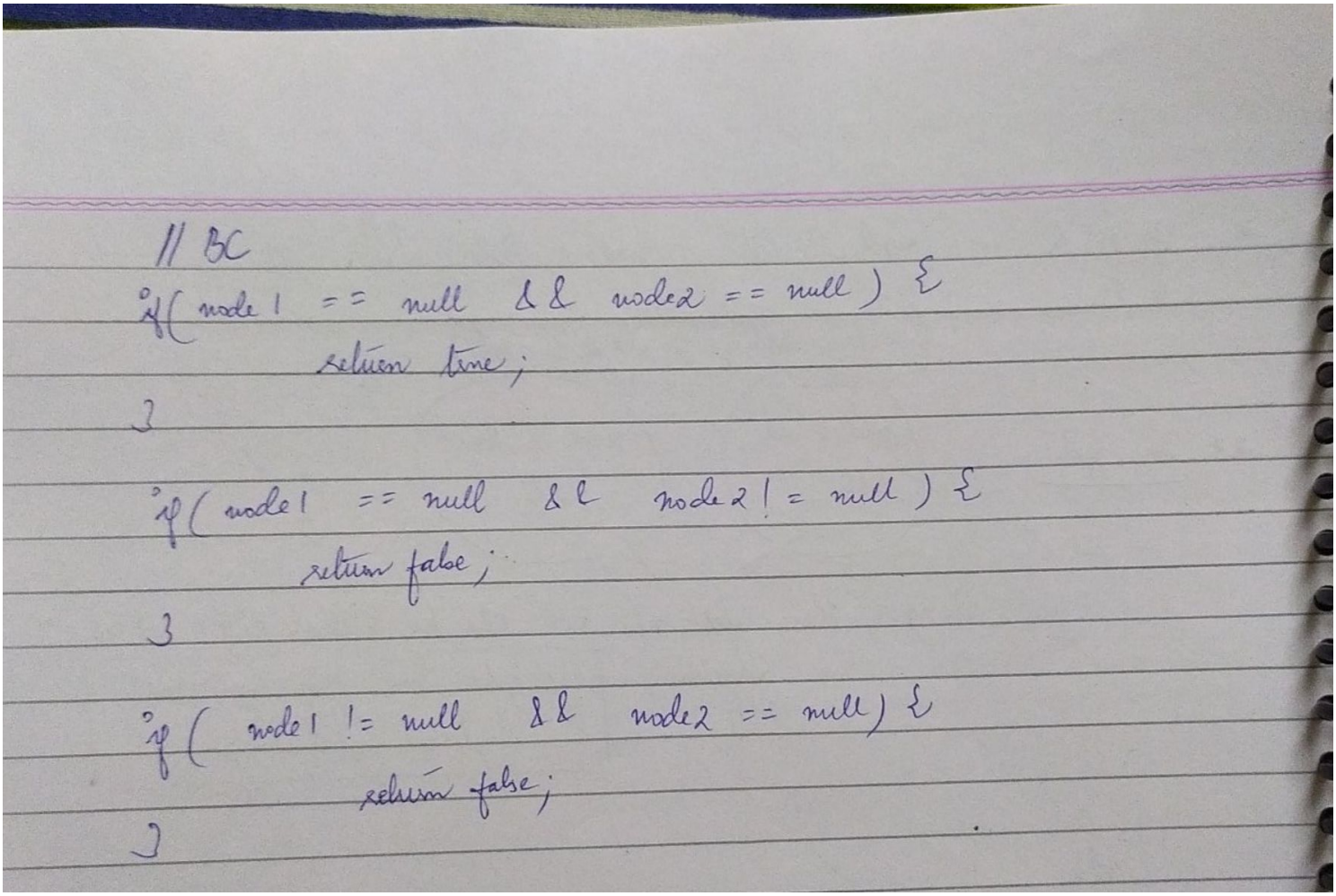
** Imp. note:



⇒ Hum yeh nhi keh rhe ki bilkul ditto match ho jayein, [ye apas me flip equivalent hone chaliye]

Code:

```
public boolean flipEquivalent (BinaryTree other) {
        return flipEquivalent (root, other.root);
}

private boolean flipEquivalent (Node node1, Node node2) {
        // BC
        if ( node1.data != node2.data ) {
                return false;
        }

        boolean ll = flipEquivalent (node1.left , node2.left );
        rr                           right       right
        lr                           left        right
        rl                           right       left

        return (ll && rr) || (lr && rl);
                ↓ not flip ✗   ↓ flip ✓
```



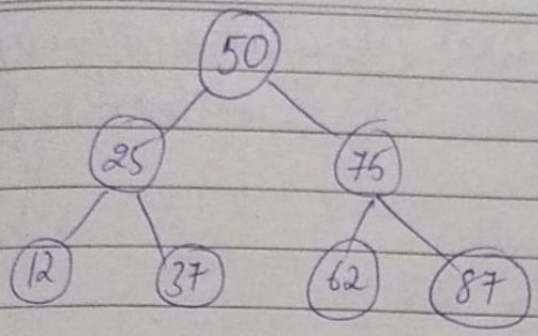
```
// BC
if( node1 == null && node2 == null ) {
        return true;
}

if( node1 == null && node2 != null ) {
        return false;
}

if( node1 != null && node2 == null ) {
        return false;
}
```

PAGE NO.
DATE : / /

Level Order

```
public void levelOrder() {

    LinkedList<Node> queue = new LinkedList<>();
    queue.add(root);

    while(!queue.isEmpty()) {

        Node rm = queue.removeFirst();
        System.out.print(rm.data + ", ");

        if(rm.left != null) {
            queue.addLast(rm.left);
        }

        if(rm.right != null) {
            queue.addLast(rm.right);
        }
    }
}
```

Is BT a BST ?



① Inorder traversal of the tree & store all value in an array list

12, 25, 37, 50, 62, 75, 87 ← inorder

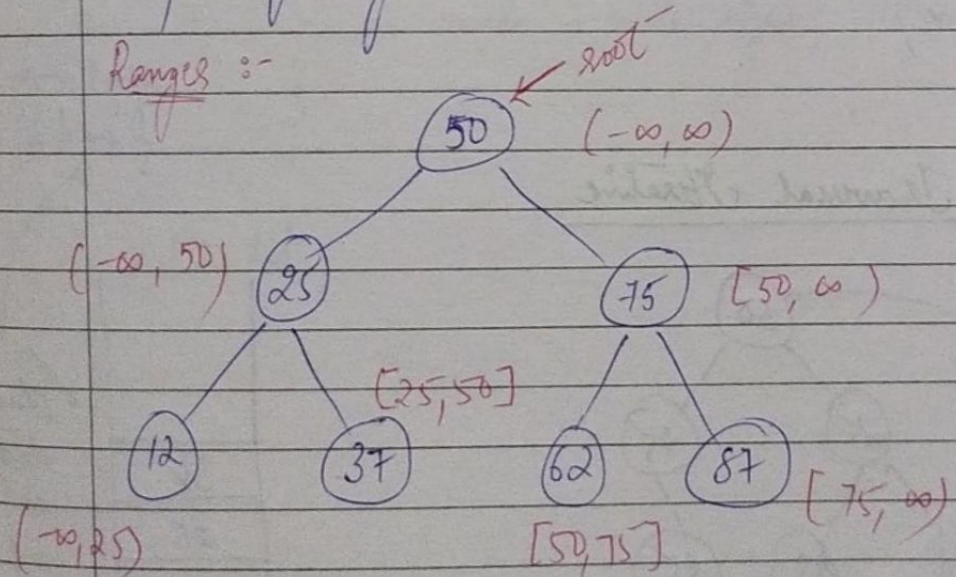
↓

sorted array mila

∴ BST ✓

② Compare by Range

Ranges :-

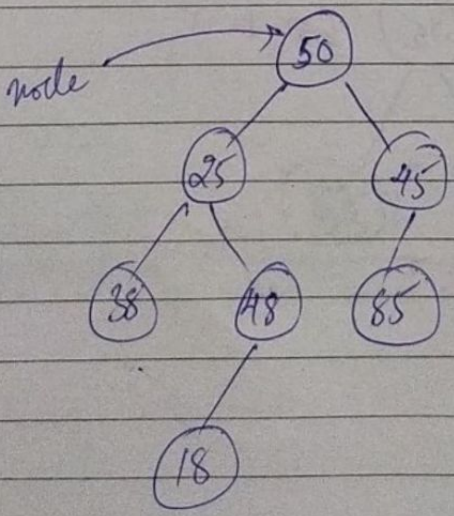


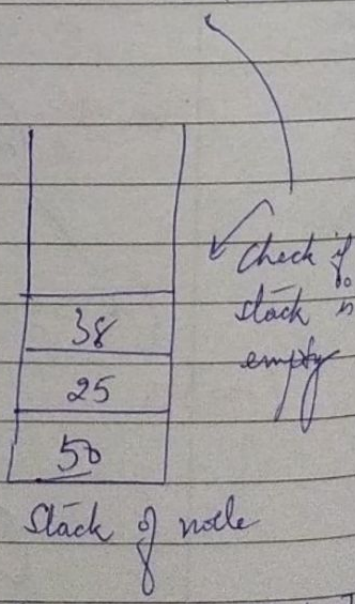
Code →

```
public static boolean isBST (BinaryTreeNode<Integer> root, int min,
                             int max) {

    if(root == null)
        return true;

    if(root.data > max || root.data < min)
        return false;

    else if(!isBST(root.left, min, root.data))
        return false;

    else if(!isBST(root.right, root.data, max))
        return false;

    return true;
}
```

3

Inorder Traversal Iterative





list : [38, 25, 18, 48, 50, 85, 45]

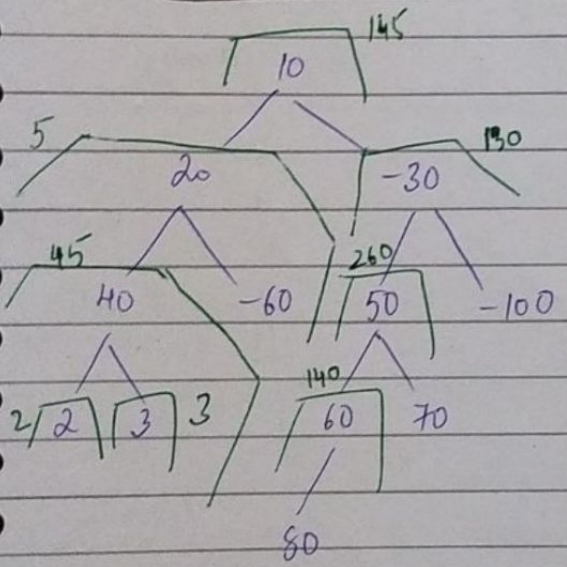
Leetcode sol" :—

```
Stack<TreeNode> st = new Stack<>();
ArrayList<Integer> list = new ArrayList<>();
while (true) {
    if (node != null) {
        st.push(node);
        node = node.left;
    }
    else {
        if (st.isEmpty()) {
            break;
        }
        node = st.pop();
        list.add(node.val);
        node = node.right;
    }
}
return list;
```

SUM OF ALL NODES

```
if(node == null){
    return 0;
}
int ls = sum(node.left);
int rs = sum(node.right);

return ls + rs + node.data;
```

MAX SUBTREE SUM



max subtree sum = 260

Code :-

```
// BC
int ls = sum(node.left);
int rs = sum(node.right);

return ls + rs + node.data
```

↳ sum of all nodes

Code :-

Max subtree sum

Logic yehi wala lagega

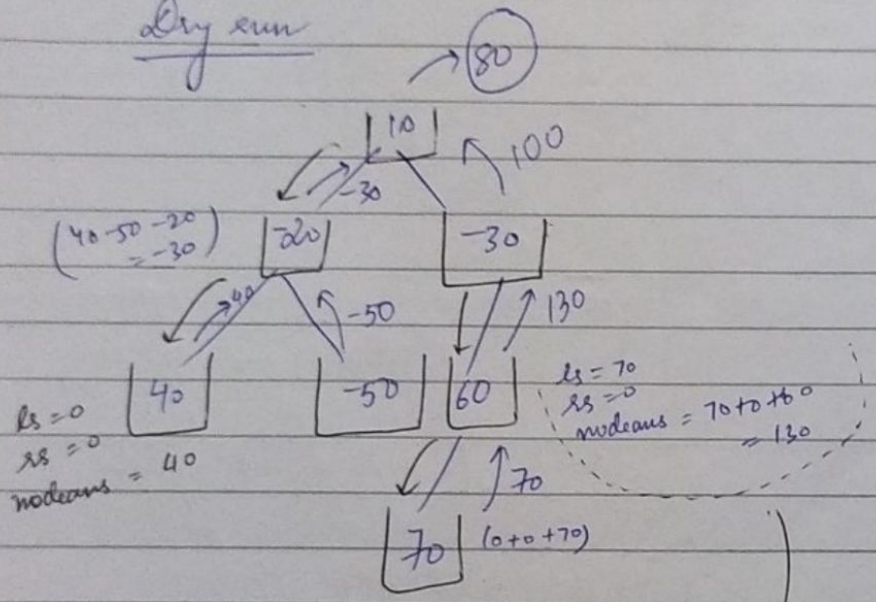
A1
A2
A3

Approach 1: Using global Variable

* Upar ek maxSum global variable rakhlenge.
  ↓
  initialized with $-\infty$

* Kaise subtree sum ko maxSum se compare karenge

Dry run



maxSum = $-\infty$
↳ Heap memory

$-\infty$ (-50 smaller)

-50 (-30 smaller)

-30 (70 is greater)

70 (130 is greater)

130 (100 is not greater)

(80 is not greater)

ye fx^n frame khatam hone par destroy ho jayega (saare var. khatam ho jayenge)

Ans = 130

Code :

```
int maxSum = Integer.MIN_VALUE;
public int maxSubtreeSum() {
        maxSubtreeSum(root);
        return maxSum;
}

private int maxSubtreeSum(Node node) {
        if(node == null) {
                return 0;
        }

        int ls = maxSubtreeSum(node.left);
        ss                    (node.right);

        int nodeans = ls + ss + node.data;

        if( nodeans > maxSum ) {
                maxSum = nodeans;
        }

        return nodeans;

}
```

Approach 2 : Recursion is returning something (max^m subtree sum)

* Rec call ka meaning clear hona chahiye mind me

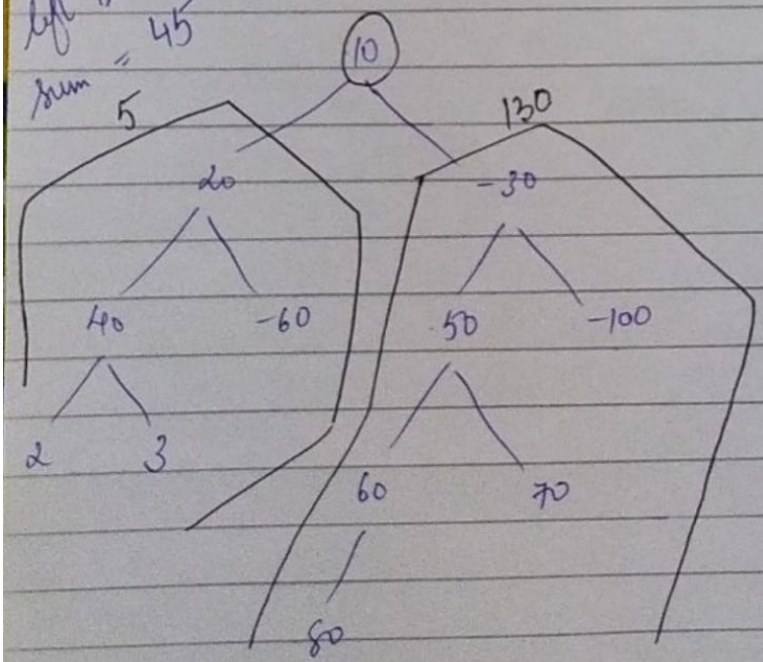
* 45 → left : max st sum
  260 → right : max st sum
  → Recursive calls

They are different

node data + left entire sum + right entire sum

10 + 5 + 130

145



right max st sum = 260

Max (45, 260, 145)

↓

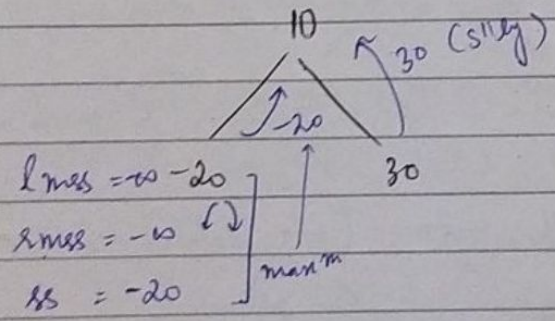
Ans //

Code:

```
if (node == null) {
    return Integer.MIN_VALUE;
}

int lMaxSubtreeSum = maxSubtreeSum(node.left);
int rMaxSubtreeSum = maxSubtreeSum(node.right);
int selfSum = sum(node);

return Math.max(selfSum, Math.max(lMaxSubtreeSum, rMaxSubtreeSum));
```

Dry run

lmss = -20, rmss = 30, ss = 40] → max^m = 40



Iss code me 4 rec calls lag rhi hain

sum(node) → sum(node.left), sum(node.right)

Approach 3:

Pair use kake hoga

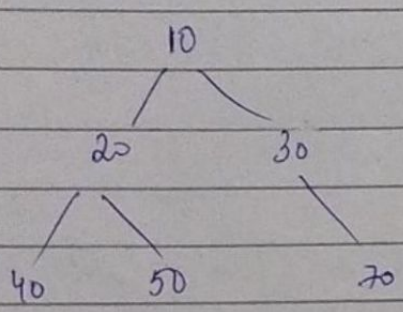
(Reduce TC)

TC ↑

Code :

```
private class maxSubtreeSumPair {
    int entireSum = 0;
    int maxSubtreeSum = Integer.MIN_VALUE;
}

public int maxSubtreeSum() {
    return maxSubtreeSum(root).maxSubtreeSum;
}

private maxSubtreeSumPair maxSubtreeSum(Node node) {
    // BC
    maxSubtreeSumPair lp = maxSubtreeSum(node.left);
                       rp                (node.right);

    sp = new maxSubtreeSumPair();

    sp.entireSum = lp.entireSum + rp.entireSum + node.data;

    sp.maxSubtreeSum = Math.max(sp.entireSum,
                        Math.max(lp.maxSubtreeSum, rp.maxSubtreeSum));

    return sp;

// BC
if (node == null) {
    return new maxSubtreeSumPair();
}
```

Imp° Construct Tree from Preorder & Inorder traversal
Postorder & Inorder



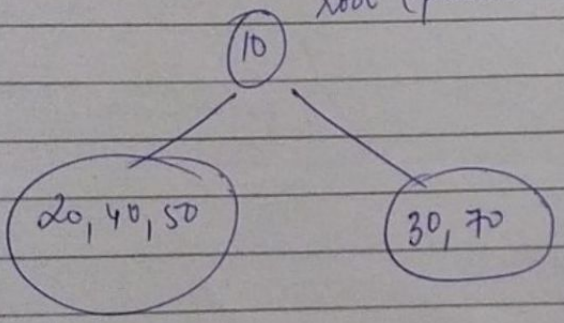
(NLR) Preorder : 10 20 40 50 30 70

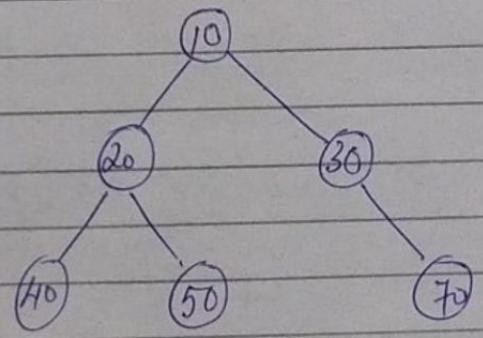
(LNR) Inorder : 40 20 50 10 30 70

L(3) R(2)

L (3) R (2)

root (preorder se pata chala)





L R (Inorder ko dekhke pata chala)

20, 40, 50 (NLR)

40, 20, 50 (LNR)

L R

left subtree ka root

root

30, 70 (NLR)

30, 70 (LNR)

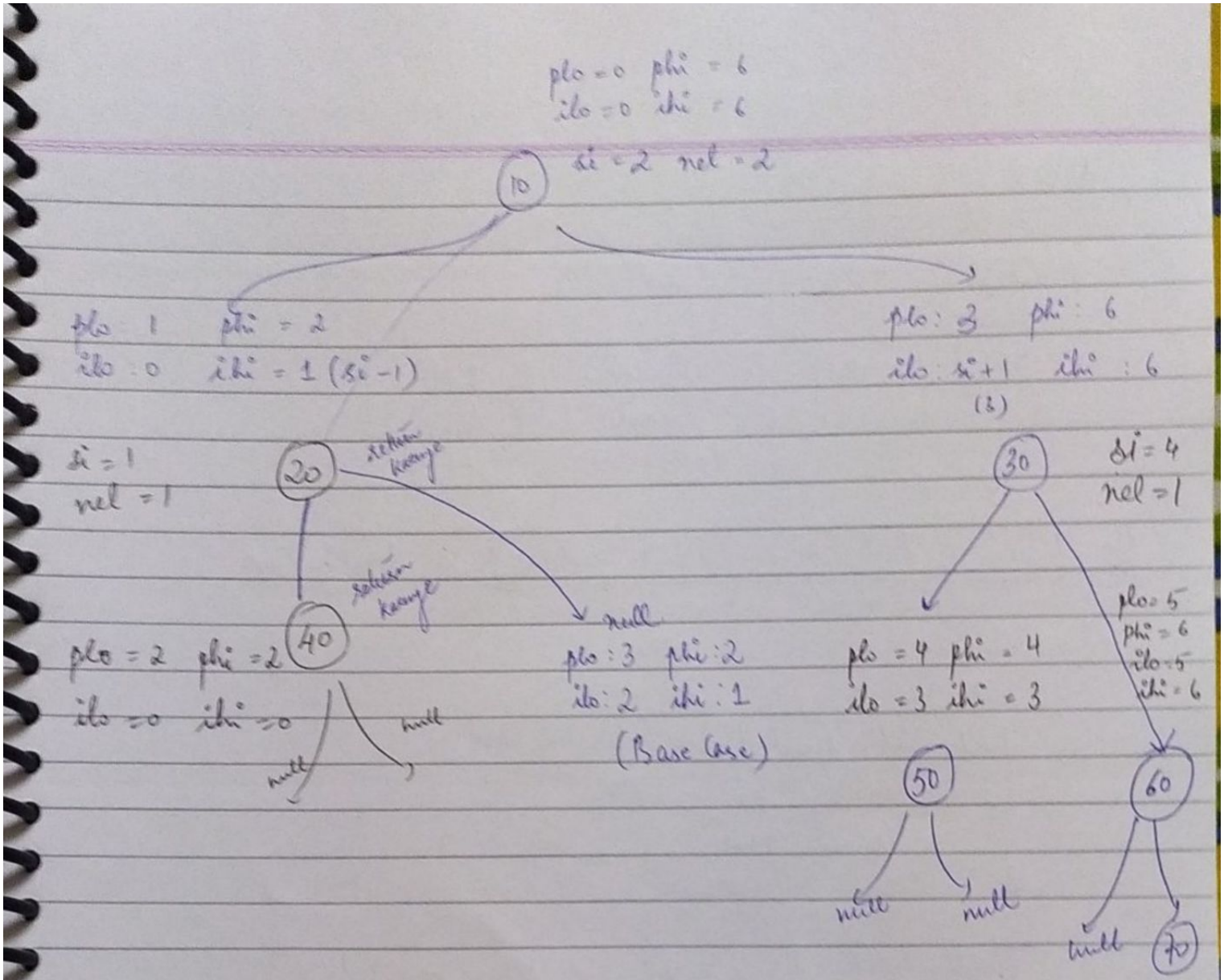
right

Preorder : [10, 20, 40, 30, 50, 60, 70] NLR

Inorder : [40, 20, 10, 50, 30, 60, 70] LNR

plo = 0 phi = 6
ilo = 0 ihi = 6

```
private Node construct(int[] pre, int plo, int phi, int[] in, int ilo, int ihi){
//BC ① Create a new node with plo

    Node nn = new Node();
    nn.data = pre[plo];                    // 10 wali node

② search for pre[plo] in inorder

    int si = -1, nel = 0;
    for(int i = ilo; i <= ihi; i++){
        if(pre[plo] == in[i]){
            si = i;                        // si = 2 (index of 10)
            break;
        }
        nel++;                             // nel = 2 (10 se pehle no of elem (40,20))
    }

③ left and right child call

    nn.left = construct(pre, plo+1, plo+nel, in, ilo, si-1);
    nn.right = construct(pre, plo+nel+1, phi, in, si+1, ihi);
    return nn;
```



// Base Case

```
if( ilo > ihi || plo > phi ) {
        return null;
}
```

BFS & DFS in Tree

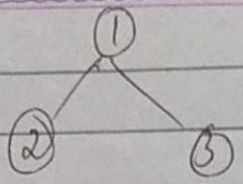
BFS → Level Order

DFS → 1. Inorder (LNR)
2. Preorder (NLR)
3. Postorder (LRN)

Q. **Sum of Root to Leaf Binary Numbers** Leetcode

```
class Solution {
    int sum = 0;
    public int sumRootToLeaf(TreeNode root){
        dfs(root, "");
        return sum;
    }

    public void dfs(TreeNode root, String curr) {
        if(root == null) {
            return;
        }

        if(root.left == null && root.right == null){
            sum += Integer.parseInt(curr + root.val, 2);
        }

        dfs(root.left, curr + root.val);
        dfs(root.right, curr + root.val);
    }
}
```

9. Similar Ques



12 + 13 = 25

```
class Solution {
    int sum = 0;
    public int sumNumbers (TreeNode root) {
        dfs (root, 0);
        return sum;
    }

    public void dfs (TreeNode root, int curr) {
        if (root == null)
            return;

        curr = curr * 10 + root.val;
        if (root.left == null && root.right == null) {
            sum += curr;
        }

        dfs (root.left, curr);
        dfs (root.right, curr);
    }
}
```