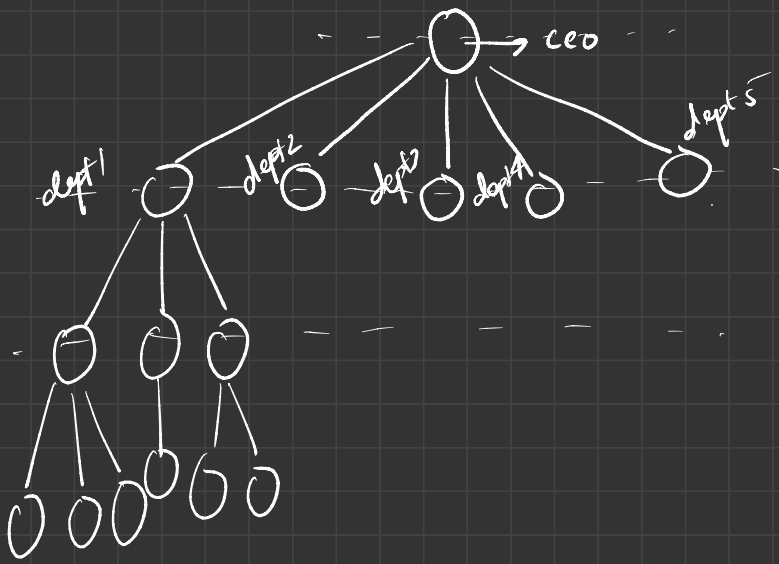
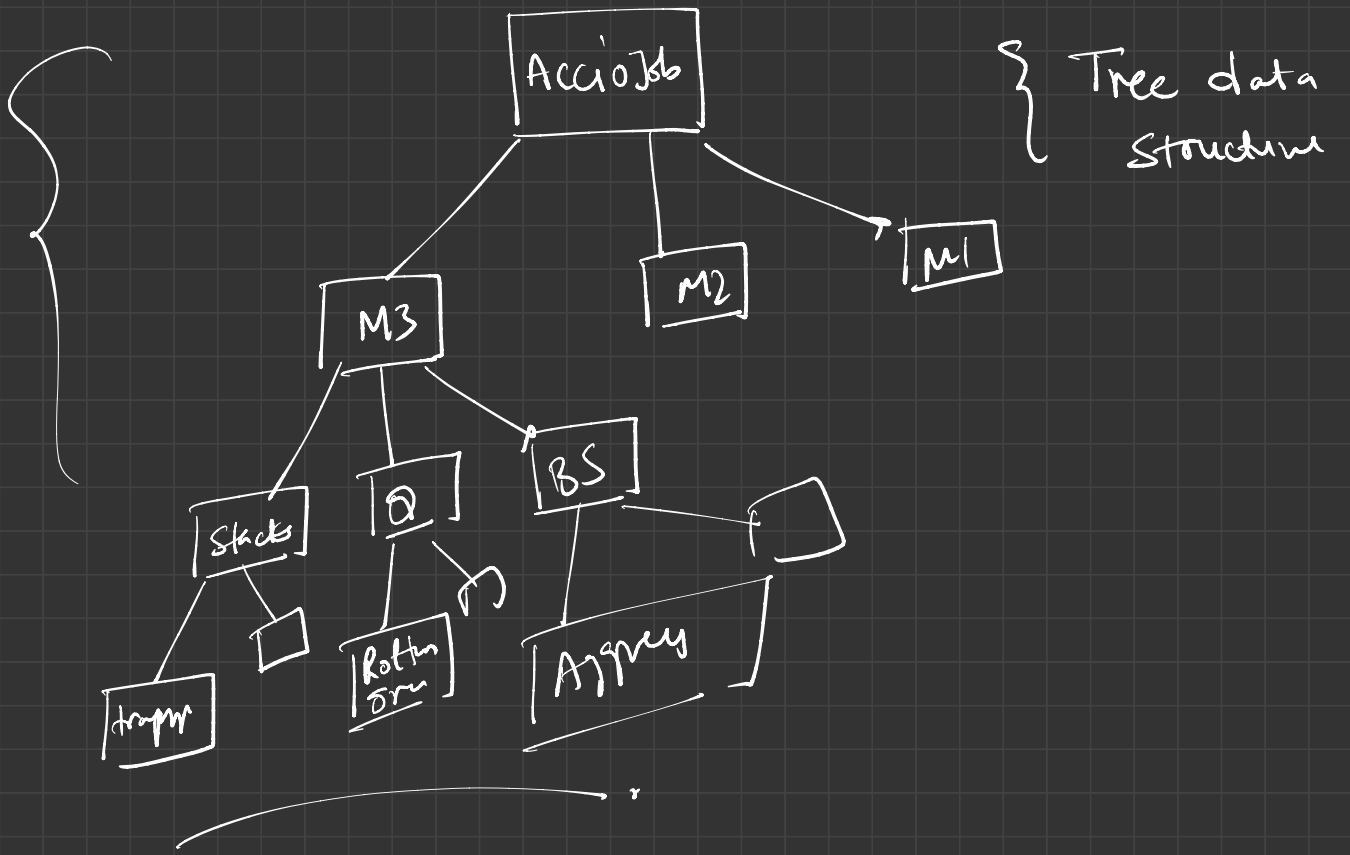
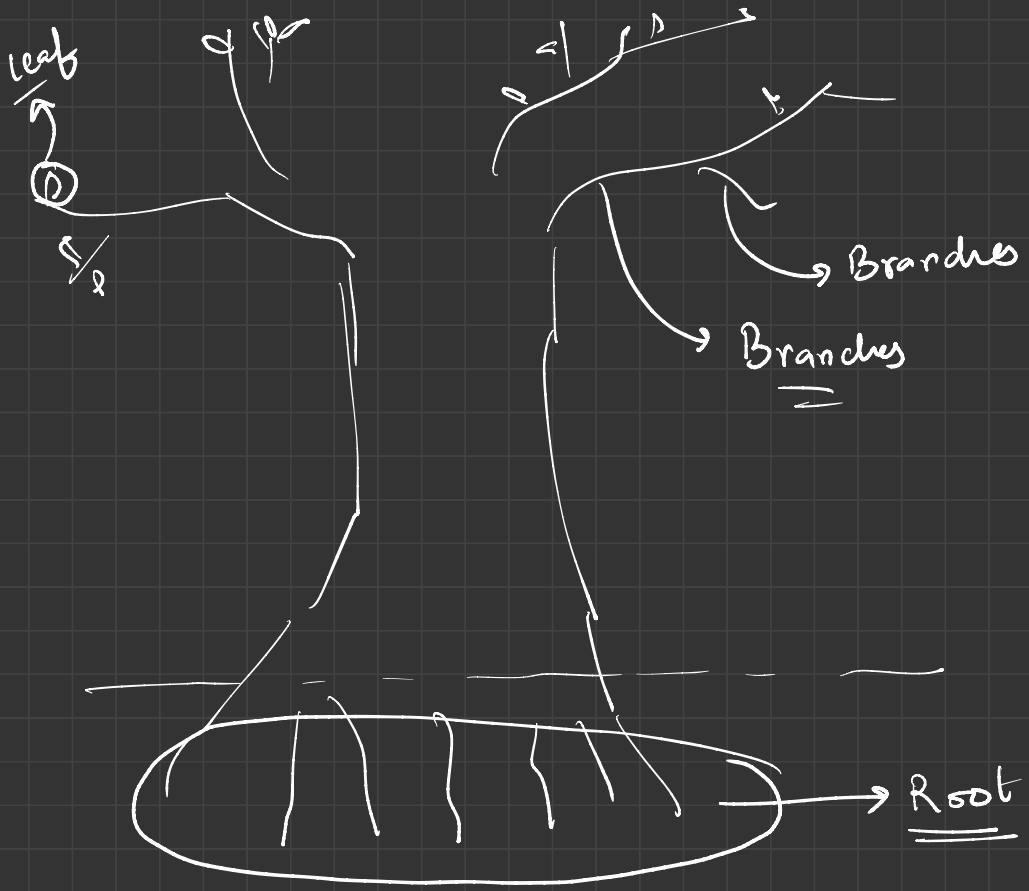
Binary Trees

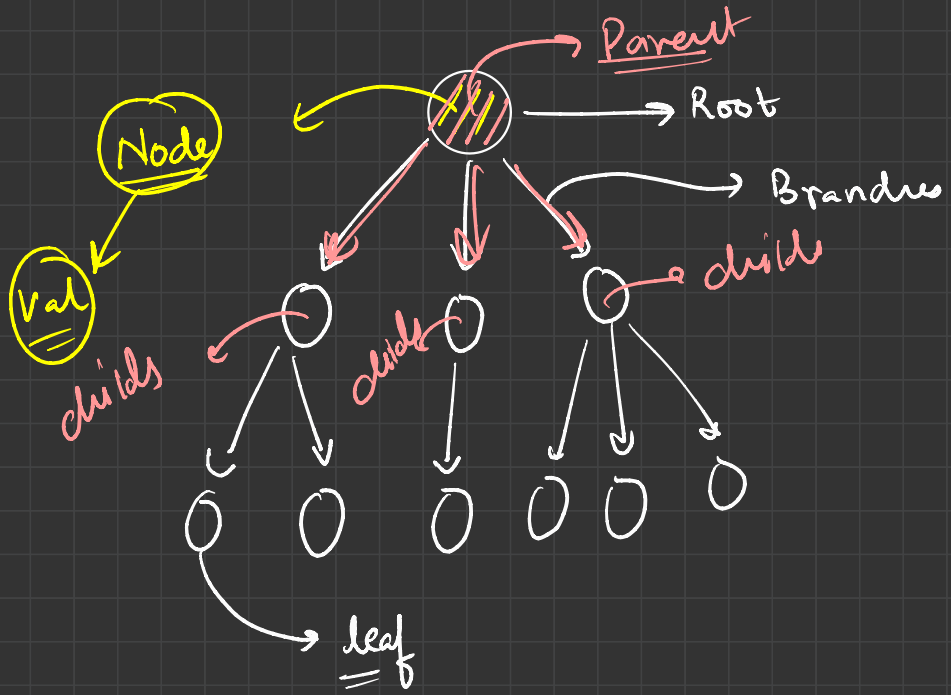


Data

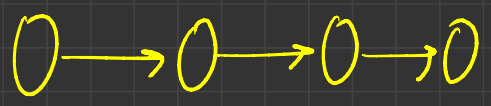
↳ tree data structure


AccioJob
M3
M2
M1
Stacks
Q
BS
Trapping
Rotten Oranges
Agency
} Tree data structure

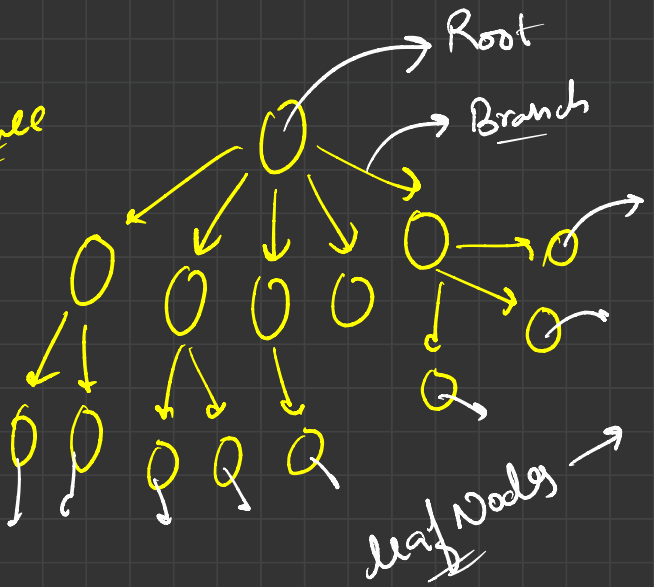

Leaf
Branches
Branches
Root

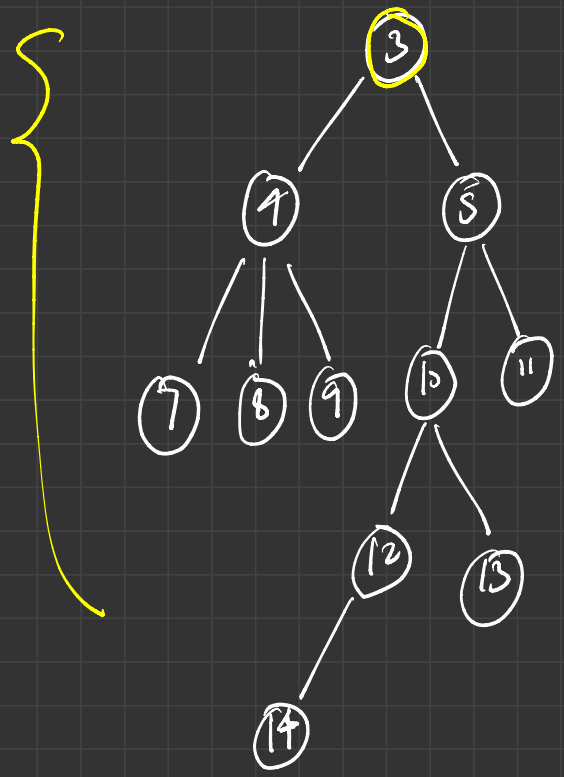

Parent
Root
Node
Val
Branches
child
childs
child
leaf

Linklist



Tree





Root : 3

leaf Nodes : 7, 8, 9, 14, 13, 11

childs of 4 : 7, 8, 9

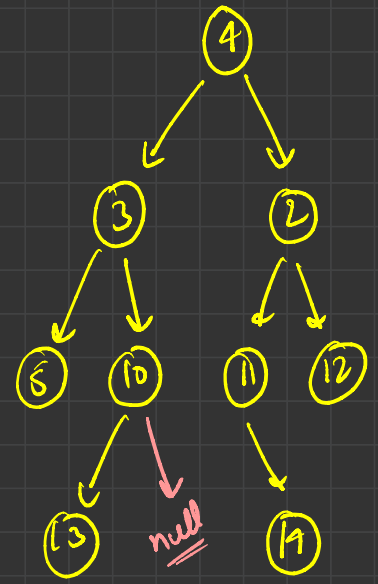
0, N

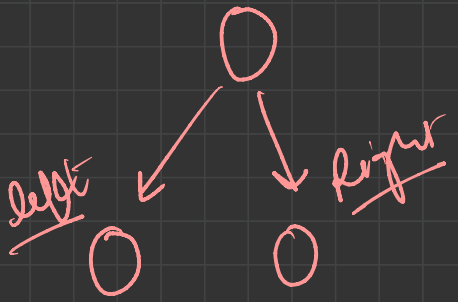
0, N childs

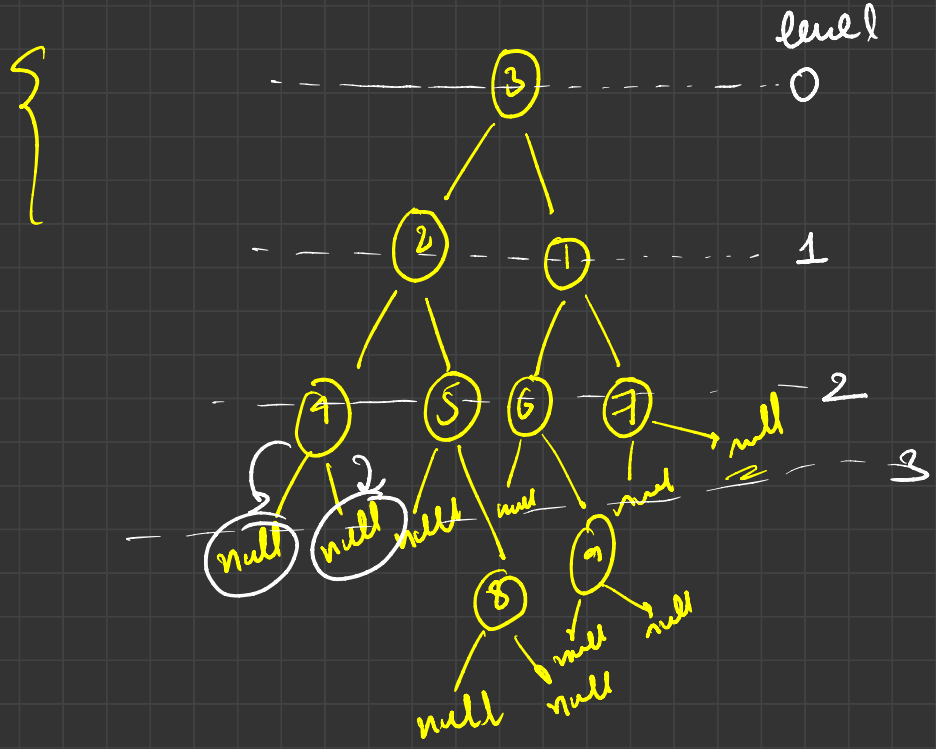
generic tree !

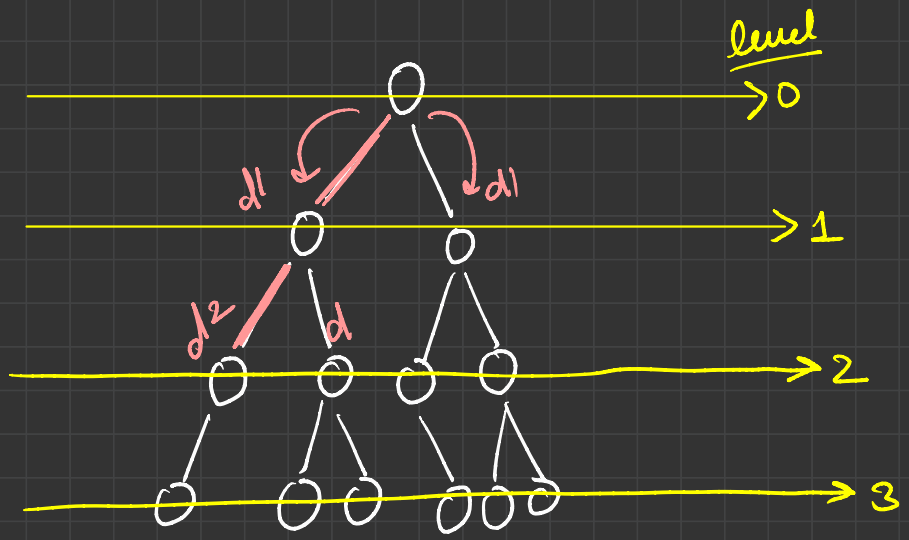
Binary Tree!

Each Parent at max has 2 children!



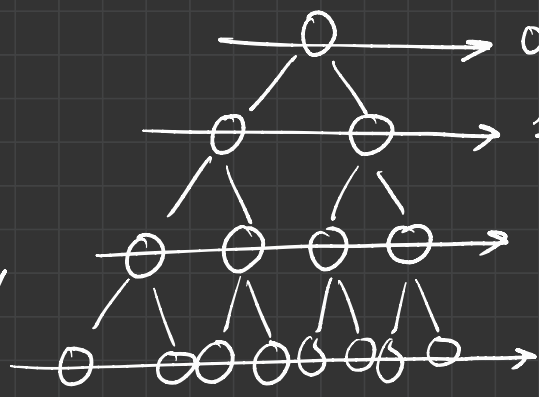



level
0
1
2
3
3
2
1
4
5
6
7
null
null
null
null
null
null
8
9
null
null
null
null


level
0
1
2
3
d1
d1
d2
d

perfect Binary tree

When Each level is Completely filled



level

0 → 1 ⇝ 2^0

1 → 2 ⇝ 2^1

2 → 4 ⇝ 2^2

3 → 8 ⇝ 2^3

⋮

h → 2^h → terms

Total Number of terms in Perfect BT of h levels

$= 2^0 + 2^1 + 2^2 + \cdots\cdots + 2^h$

$(h+1)$ terms

$r = 2$

$a = 2^0 = 1$

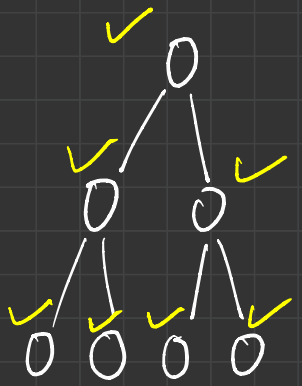
$S_n = \frac{a(r^n - 1)}{(r-1)}$

$S_{h+1} = \frac{1(2^{h+1} - 1)}{(2-1)}$

$= 2^{h+1} - 1$

$2^{6+1} - 1 = 2^6 - 1$

$= 63$



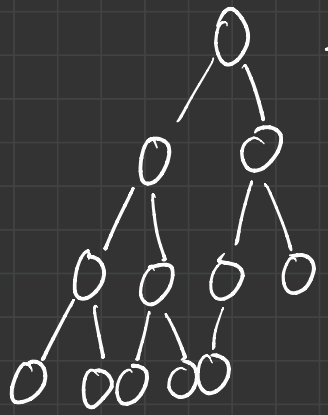
h=2

$$2^{h+1} - 1$$

$$= 2^3 - 1$$

$$= 8 - 1 = 7$$

Complete Binary tree

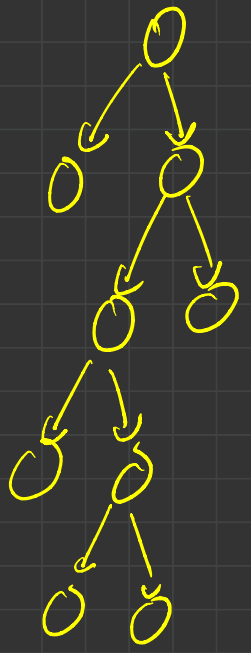


+ leaf Node prioritize left Most Parents

⇒ Every level Except last is Completely filled!

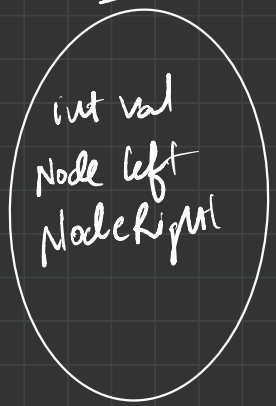
full binary tree

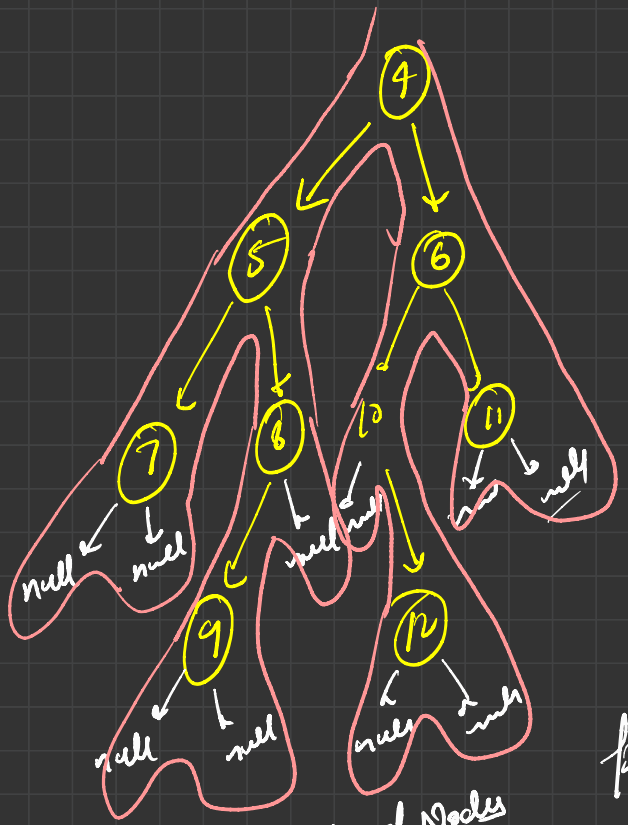
Each Node will have zero / two childs



Construct A Binary tree

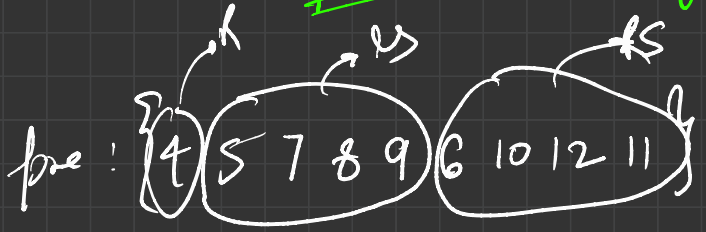
```
class Node
{
  int val;
  Node left;
  Node right;
}
```





pre Order traversal
In Order traversal
post Order traversal

print left Right

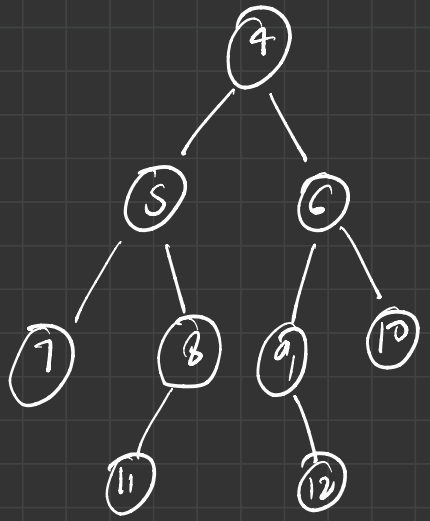


TC : O(N) — No. of Nodes
SC : O(h)

{ 4, 5, 7, 8, 9, 6, 10, 12, 11 }

In Order Traversal

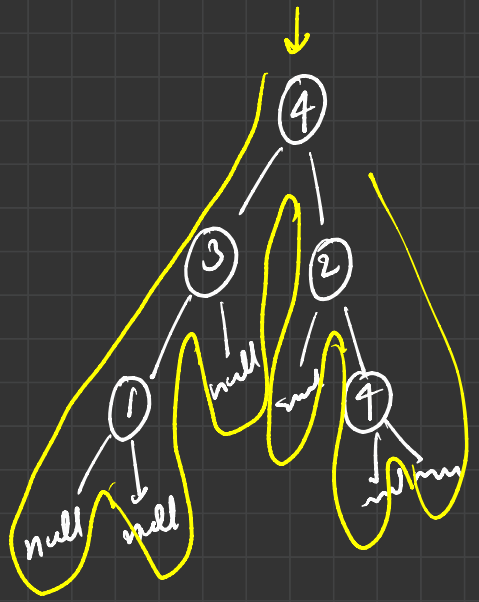
Left Print Right



{7, 5, 11, 8, 4, 9, 12, 6, 10}

{ 7, 5, 11, 8, 4, 9, 12, 6, 10 }

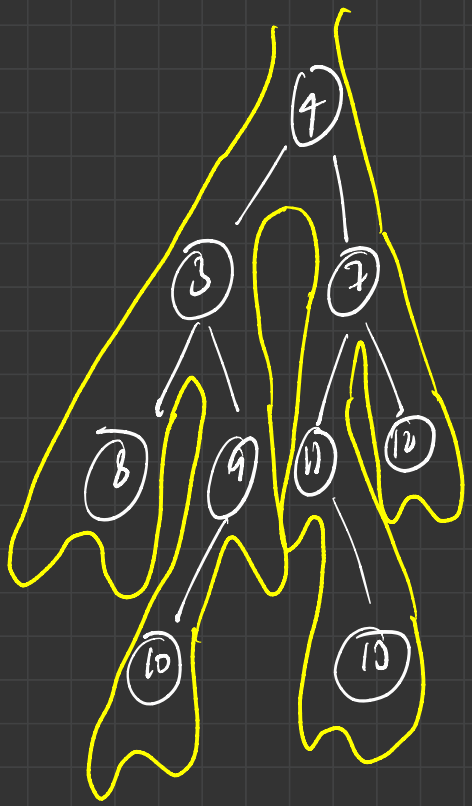
lsr Root rs

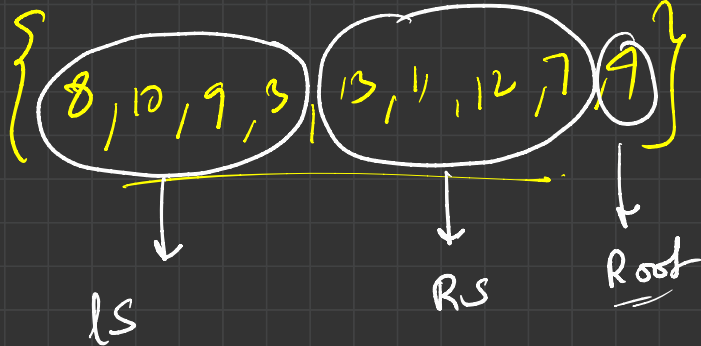


```java
class Solution {
    public void ioTraversal(TreeNode root, List<Integer> ans) {
        if (root == null) {
            return;
        }

        ioTraversal(root.left, ans);

        ans.add(root.val);

        ioTraversal(root.right, ans);
    }
}
```

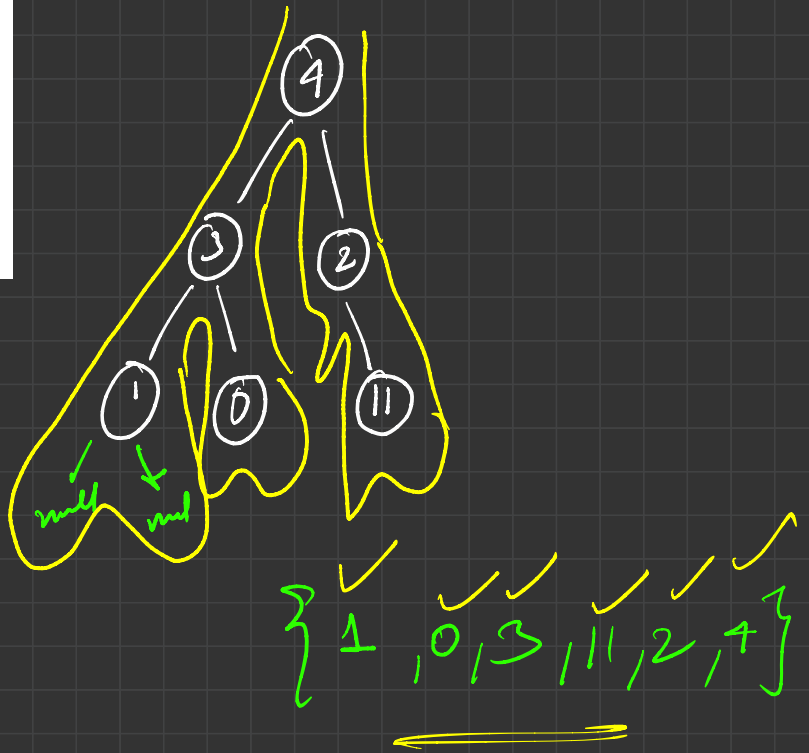
{1, 3, 4, 2, 4}

post Order traversal : left Right from



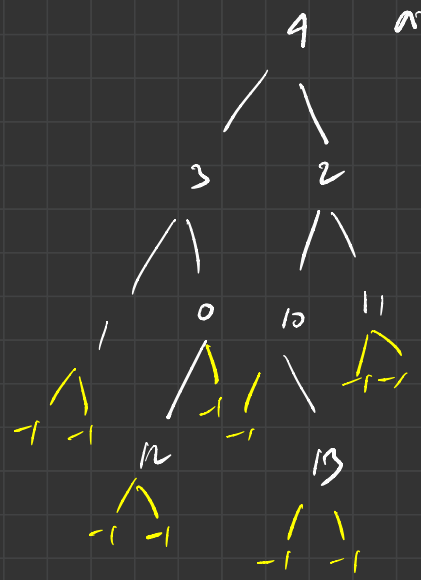


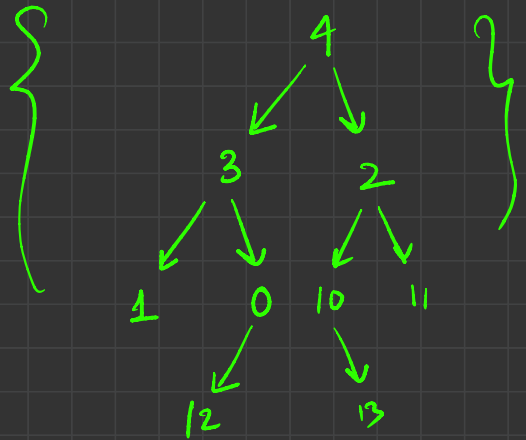
```java
class Solution {
    public void postTraversal (TreeNode root, List<Integer> ans)
        if (root == null) {
            return;
        }

//          call left subtree
        postTraversal(root.left, ans);

//          call right subtree
        postTraversal(root.right, ans);

//          add myself
        ans.add(root.val);
    }
```



Construct A Binary tree

arr = [4, 3, 1, -1, -1, 0, 12, -1, -1, -1, 2, 10, -1, 13, -1, -1, 11, -1, -1]

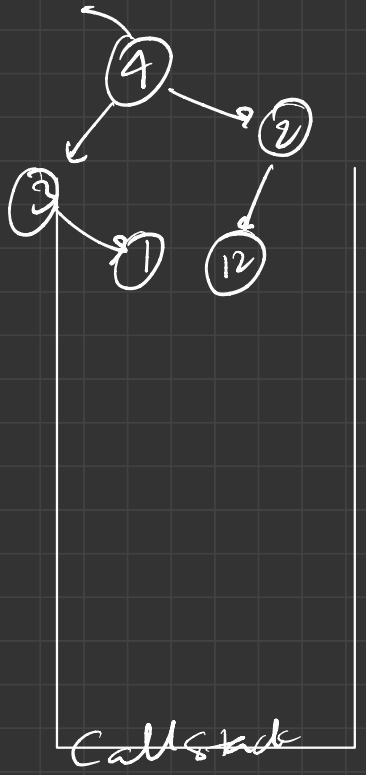


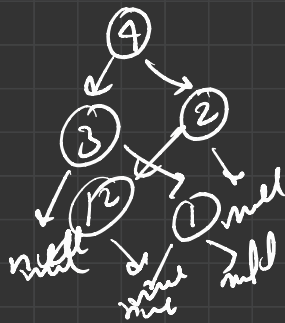


Amzon

4, 3, -1, 1, -1, -1, 2, 12, -1, -1, -1.

```java
static TreeNode constructBinaryTree() {
    Scanner scn = new Scanner(System.in);

    System.out.println(x: "Print value of the node");
    int val = scn.nextInt();
    TreeNode root = new TreeNode(val);

    if (val == -1) {
        return null;
    }

    System.out.println(x: "Enter my left tree root");
    root.left = constructBinaryTree();

    System.out.println(x: "Enter my right tree root");
    root.right = constructBinaryTree();

    return root;
}
```

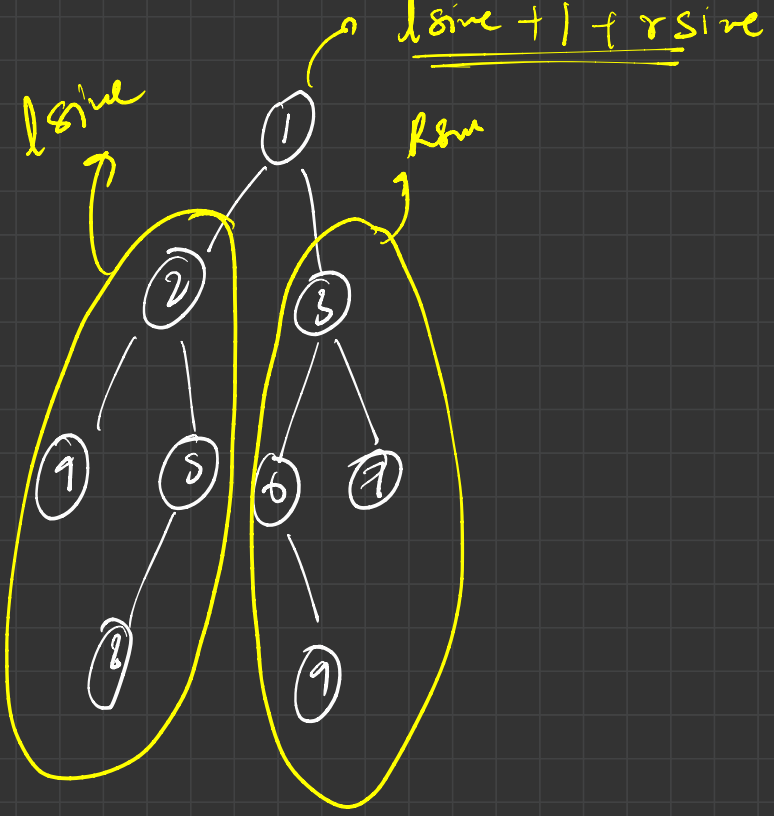




Binary tree

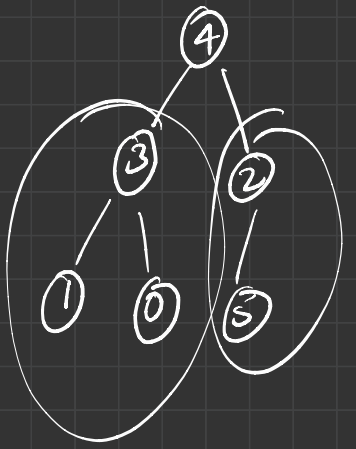
Size = Number of Nodes

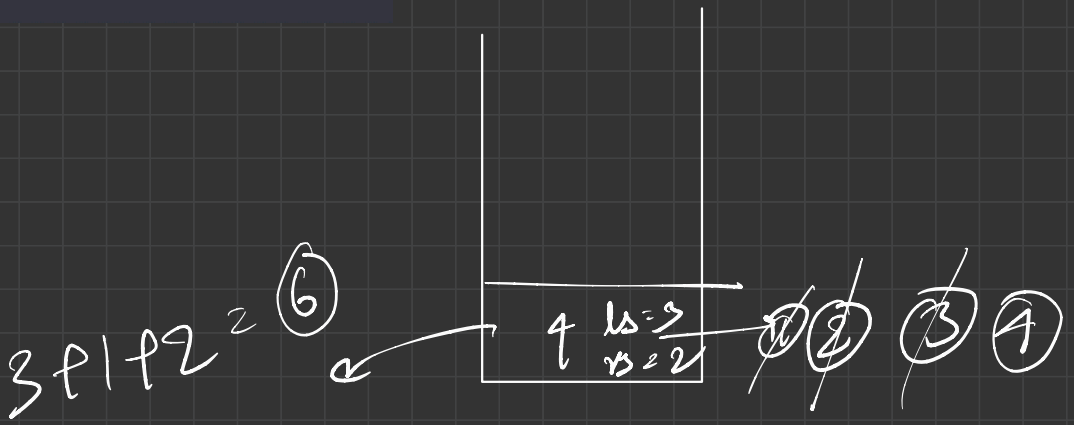
9

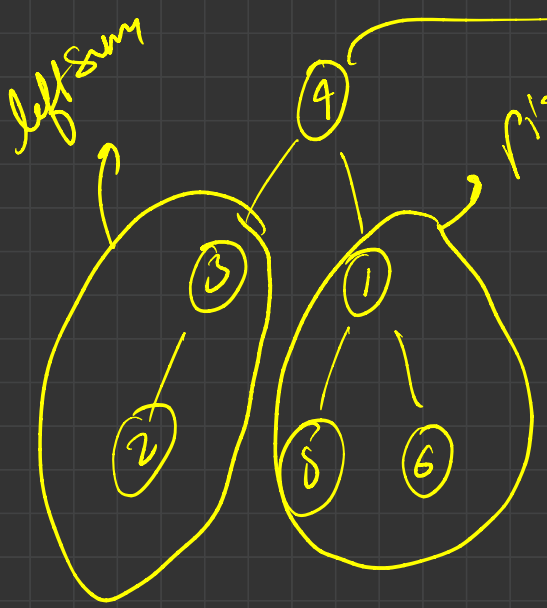



```
int Size(TreeNode root)
{
    if (root == null)
        return 0;
    int ls = size(root.left);
    int rs = size(root.right);
    return ls + rs + 1;
}
```

```java
public int sizeOfTree(Node root) {
    if (root == null) {
        return 0;
    }

    // get size of left subTree
    int ls = sizeOfTree(root.left);

    // get size of right subTree
    int rs = sizeOfTree(root.right);

    // total size is left size + myselft + right size
    return ls + 1 + rs;
}
```



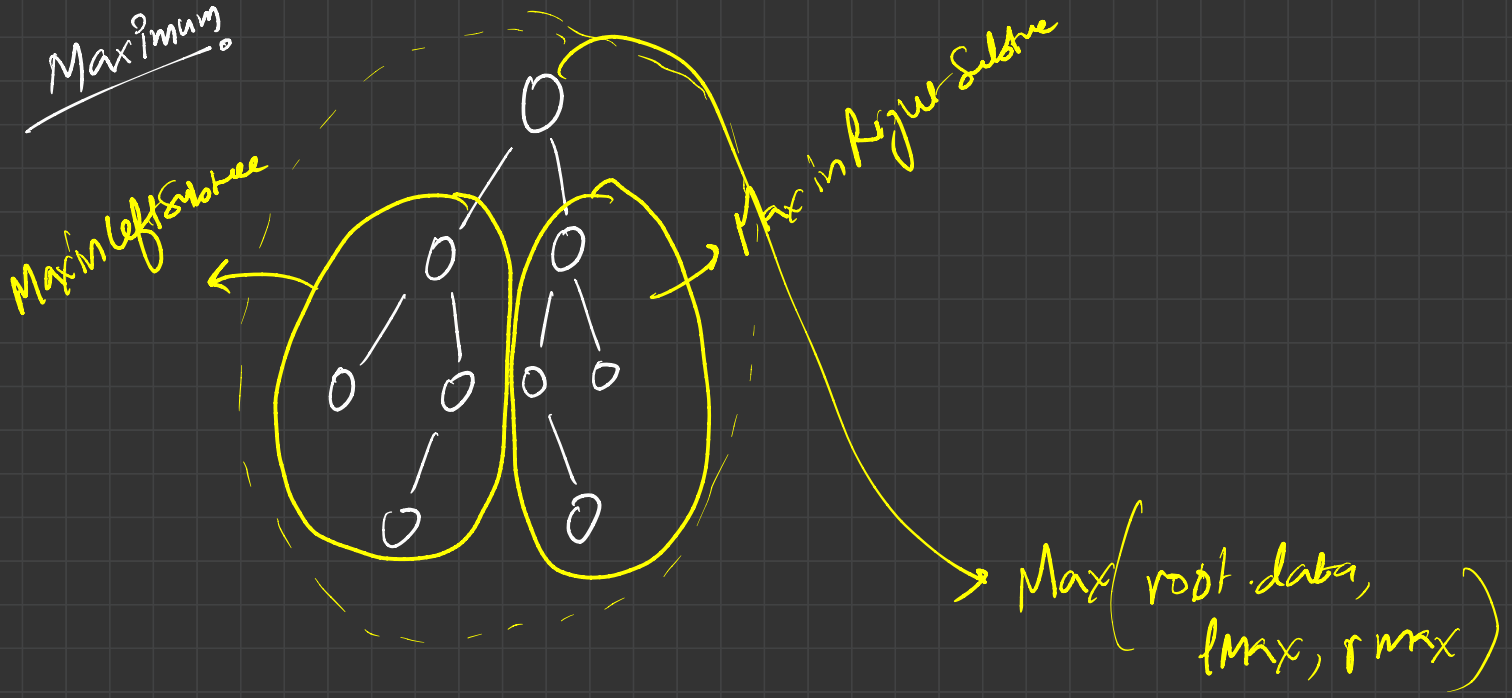


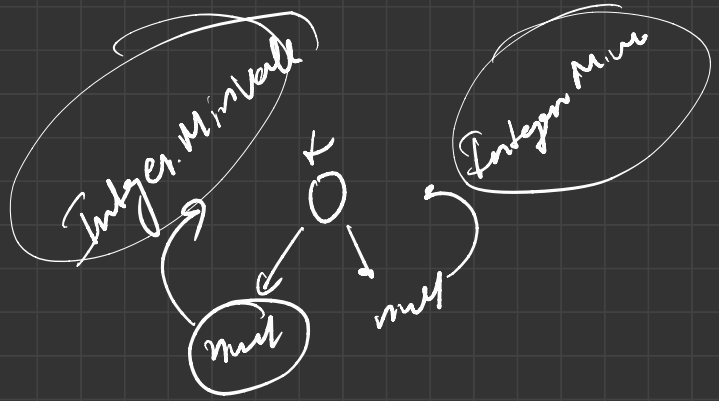


leftSum + root.data + RightSum

faith: it gives Sum of subtree

Maximum

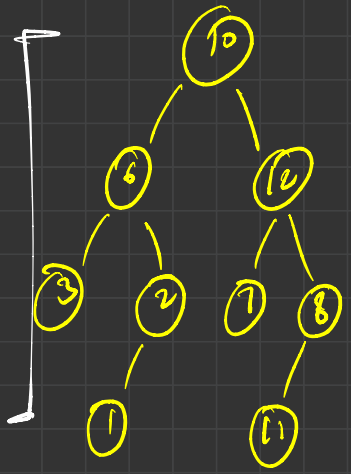

Max in Left Subtree
Max in Right Subtree
Max(root.data,
lmax, rmax)

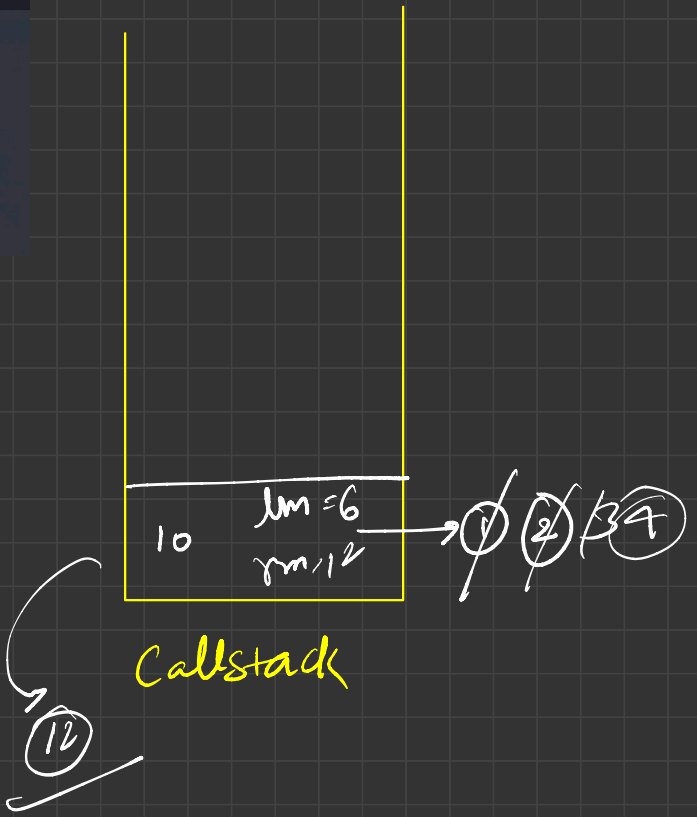

Integer.MinValue
k
Integer.MIN
null
null

```java
public int maxOfTree(Node root) {
    if (root == null) return Integer.MIN_VALUE;

    int leftMax = maxOfTree (root.left);

    int rightMax = maxOfTree (root.right);

    return Math.max(root.data, Math.max(leftMax, rightMax));
}
```

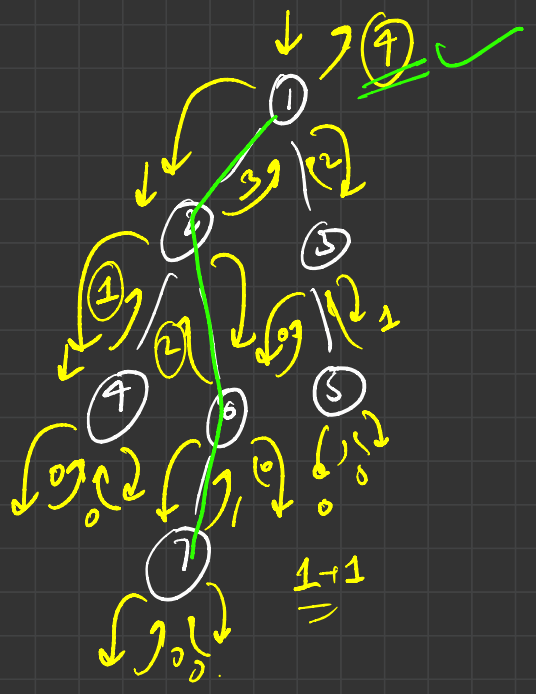
10 , 12 (12 , 6)

TC: O(N)
SC: O(H)





```java
public int heightOfTree(Node root) {
    if (root == null) {
        return 0;
    }

    // get height of left subTree
    int lh = heightOfTree(root.left);

    // get height of right subTree
    int rh = heightOfTree(root.right);

    // total height = Max(lh, rh) + 1
    return Math.max(lh, rh) + 1;
}
```

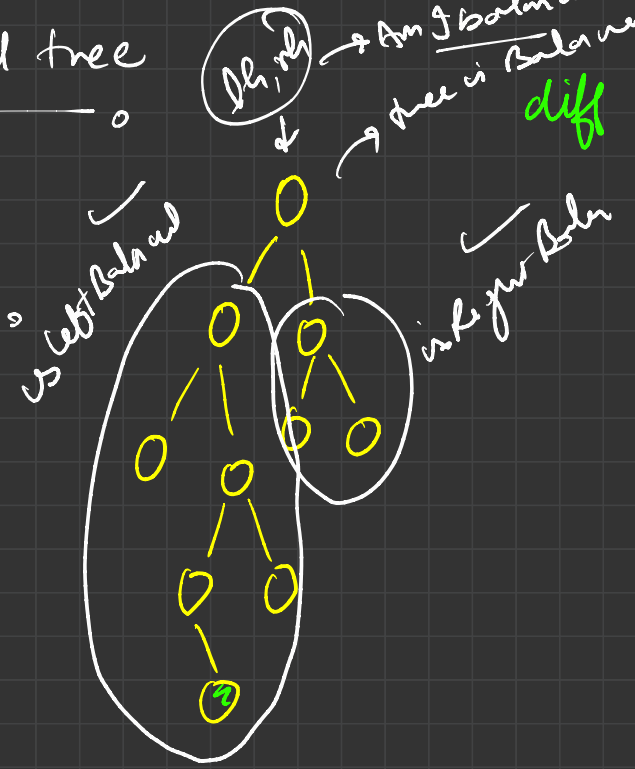


TC: O(N)

SC: O(h)

↳ height of the tree

Balanced tree



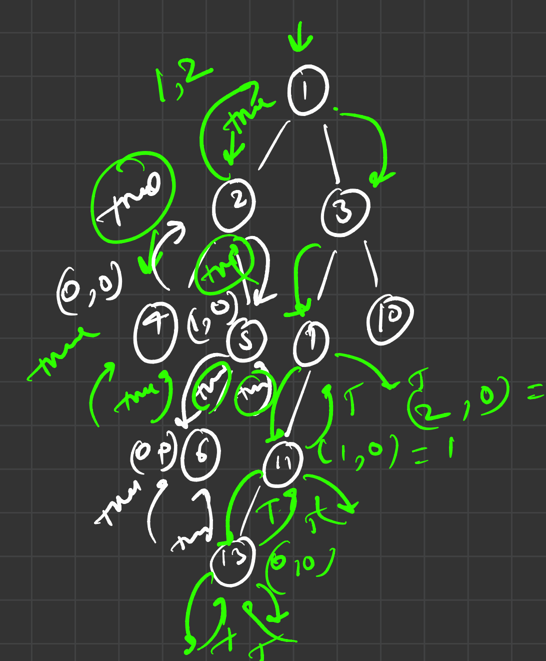
diff between $|lh - rh|$ of Subtrees should not exceed 1

```java
class Solution {
    public int height (TreeNode root) {
        if (root == null) return 0;

        int lh = height(root.left);
        int rh = height(root.right);

        return Math.max(lh, rh) + 1;
    }

    public boolean isBalanced(TreeNode root) {
        if (root == null) {
            return true;
        }

//          is left sub-tree balanced
        boolean isLeftBalanced = isBalanced(root.left);

//          is right sub-tree balanced
        boolean isRightBalanced = isBalanced(root.right);

//          check am I balanced
        int lh = height(root.left);
        int rh = height(root.right);
        int diff = Math.abs(lh - rh);

        boolean ImBalanced = false;
        if(diff <= 1) {
            ImBalanced = true;
        }

        if (isLeftBalanced == false || isRightBalanced == false || ImBalanced == false) {
            return false;
        } else {
            return true;
        }
    }
}
```

O(h)

O(h)

O(2N) ≈ O(N)

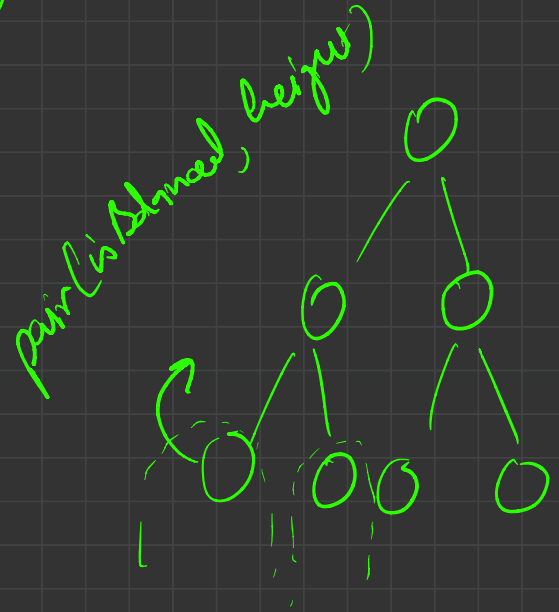


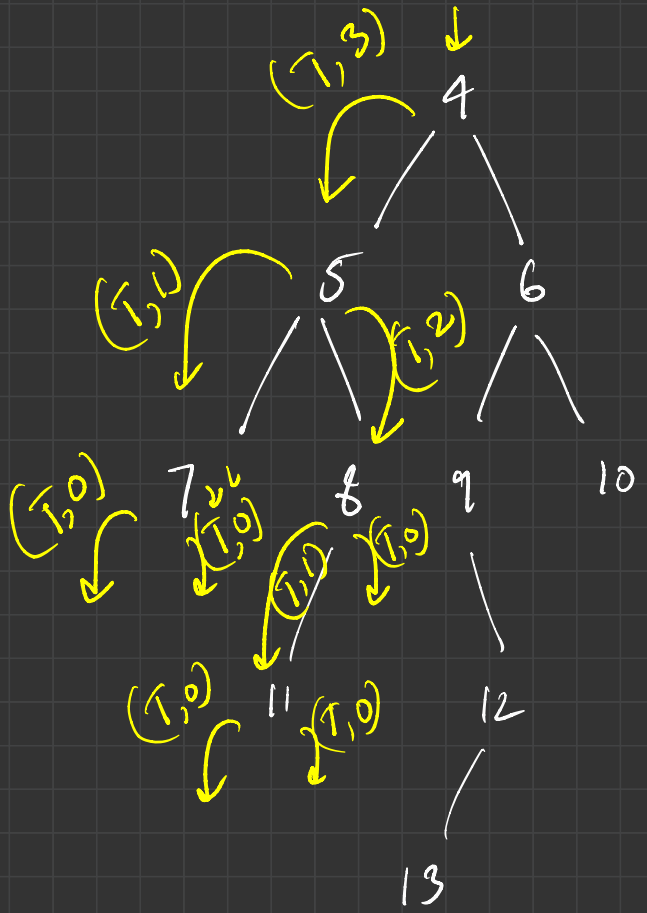
TC: O(NXN)

$O(N \times N) = O(N^2)$

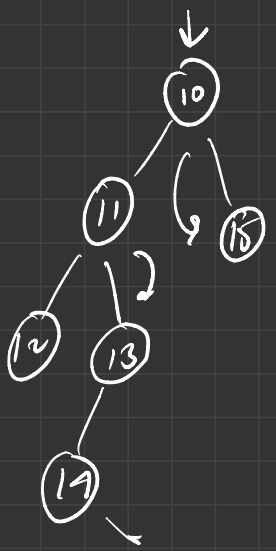
SC: O(h)

TC: O(N) ?

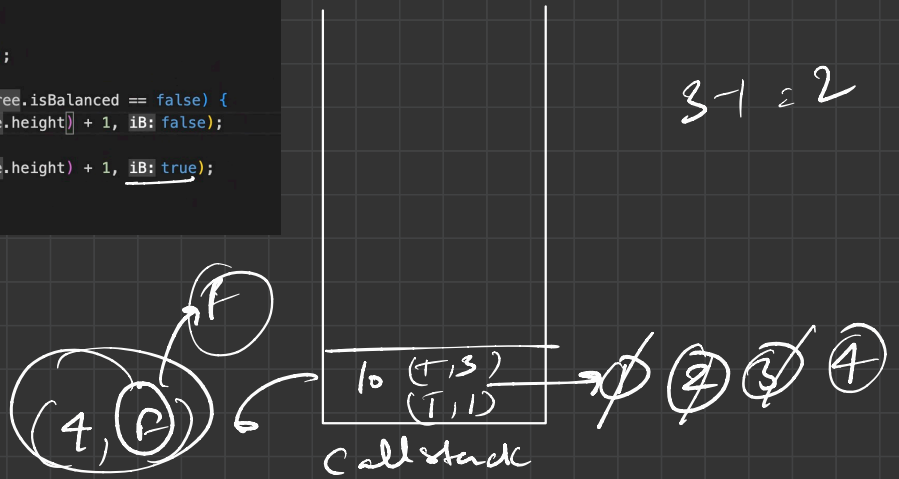



(T,3)
4
(T,1)
5
6
(T,2)
(T,0)
7
(T,0)
8
9
10
(T,1)
(T,0)
(T,0)
11
(T,0)
12
13

```java
static class Pair {
    boolean isBalanced;
    int height;

    Pair() {
        isBalanced = true;
        height = 0;
    }

    Pair(int h, boolean iB) {
        isBalanced = iB;
        height = h;
    }
}

public static Pair isBalanced_(Node root) {
    // your code here
    if (root == null) {
        return new Pair();
    }

    Pair leftSubTree = isBalanced_(root.left);

    Pair rightSubTree = isBalanced_(root.right);

    int diff = Math.abs(leftSubTree.height - rightSubTree.height);

    if (diff > 1 || leftSubTree.isBalanced == false || rightSubTree.isBalanced == false) {
        return new Pair(Math.max(leftSubTree.height, rightSubTree.height) + 1, iB: false);
    } else {
        return new Pair(Math.max(leftSubTree.height, rightSubTree.height) + 1, iB: true);
    }
}
```



3-1 = 2



diameter of tree