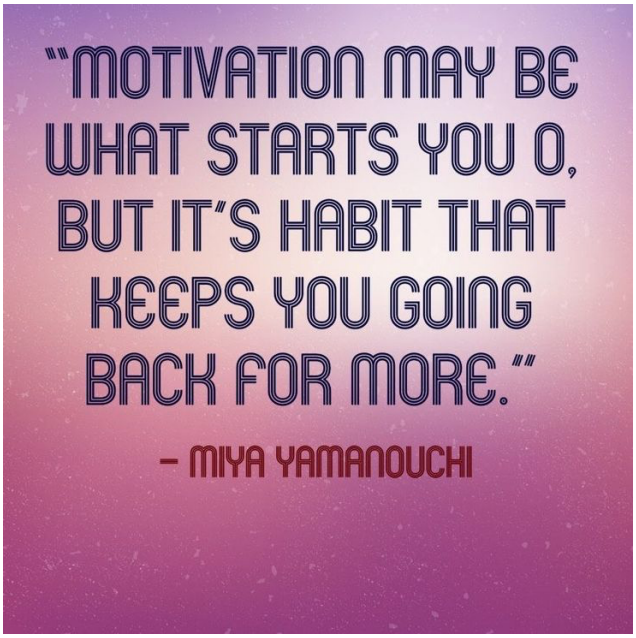
TODAY:

Trees 2

DON'T FORGET



Good Evening

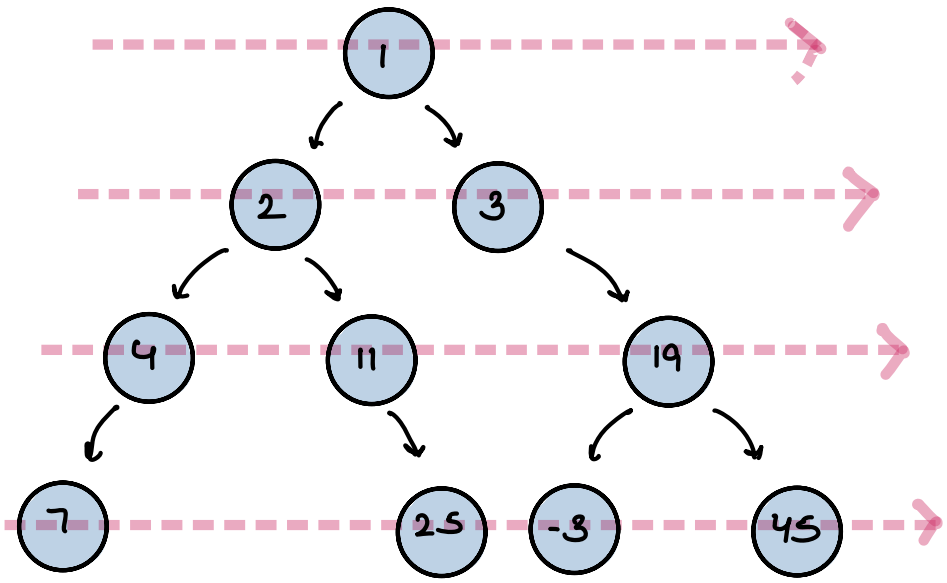
AGENDA:

Topics for Today

01. Level Order Traversal
02. Left view & Right view
03. Vertical level order Traversal
04. Top view & Bottom view
05. Types of B.T
06. Check height balanced

Level Order Traversal (BFS)

Breadth first Search



Ans = 1 2 3 4 11 19 7 25 -3 45

Queue

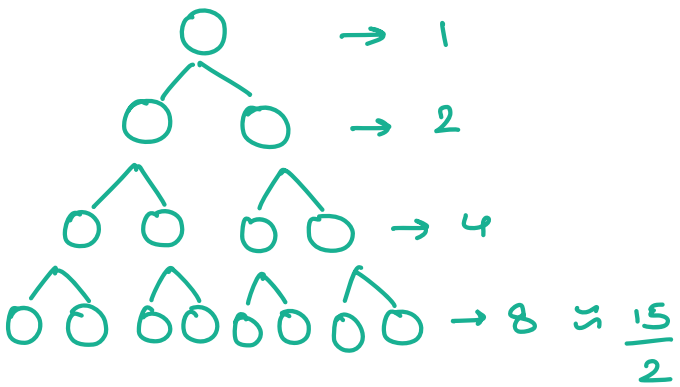
- remove
- work
- push the children

```
Queue <Node> q ;

q. enque (root);

while (q.size () > 0) {

    Node rem = q. deque ();

      print (rem.data);

      if (rem. left != null)   q. enque (rem. left)

      if (rem. right != null)   q. enque (rem. right);

}
```

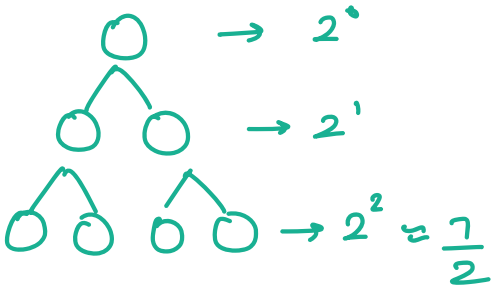
TC : O(N)

SC : O(N)

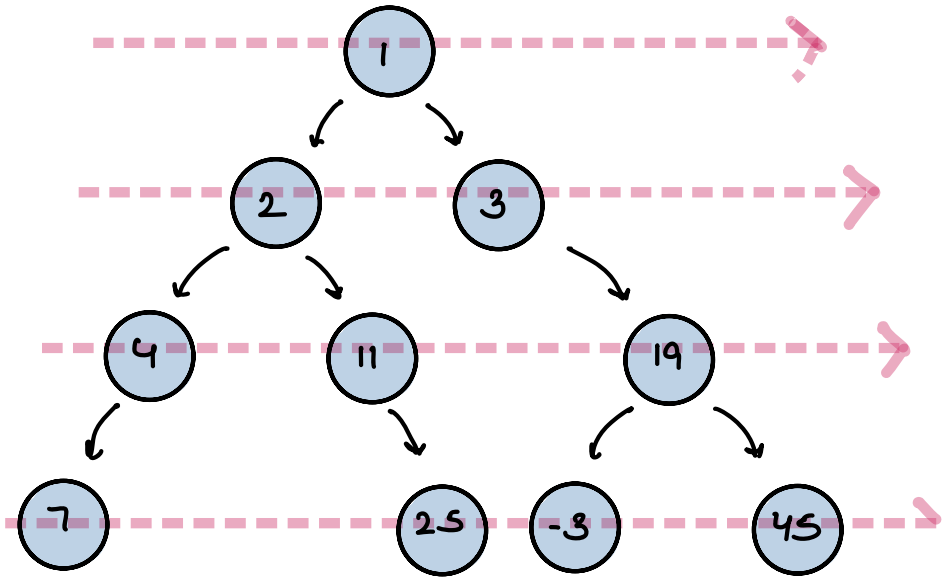
N = 15 nodes



N = 7



(2)



Output

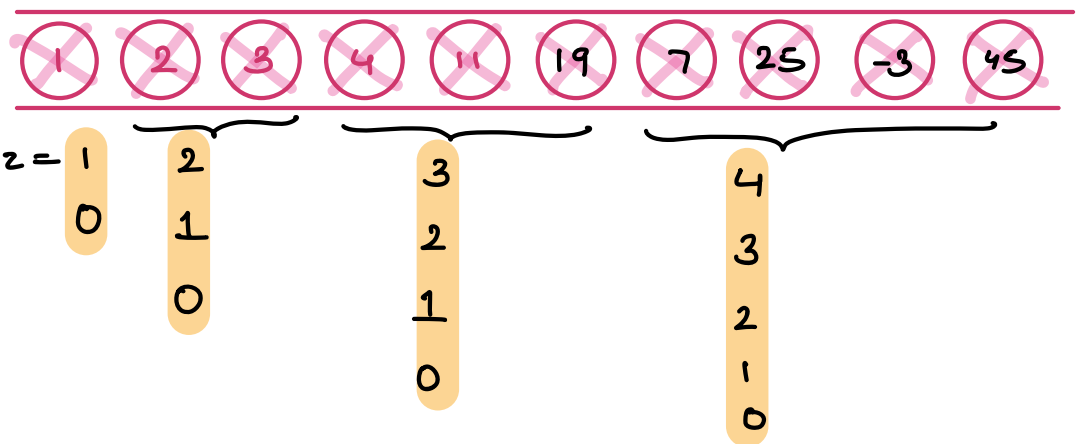
1

2 3

4 11 19

7 25 -3 45

Queue



1

2 3

4 11 19

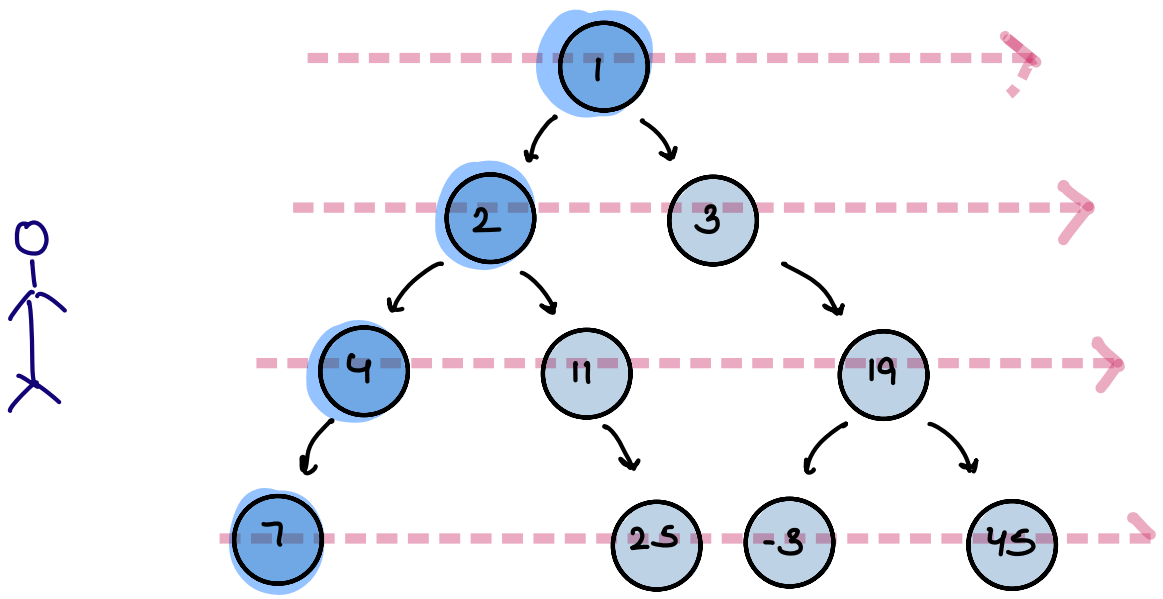
7 25 -3 45

```
Queue <Node> q ;

q. enque (root);

while (q.size() > 0) {
    int sz = q.size();
    for (i=1; i ≤ sz; i++) {
        Node rem = q.deque();
        print (rem.data);
        if (rem.left != null)  q. enque (rem.left)
        if (rem.right != null)  q. enque (rem.right);
    }
    println();
}
```

TC: O(n)

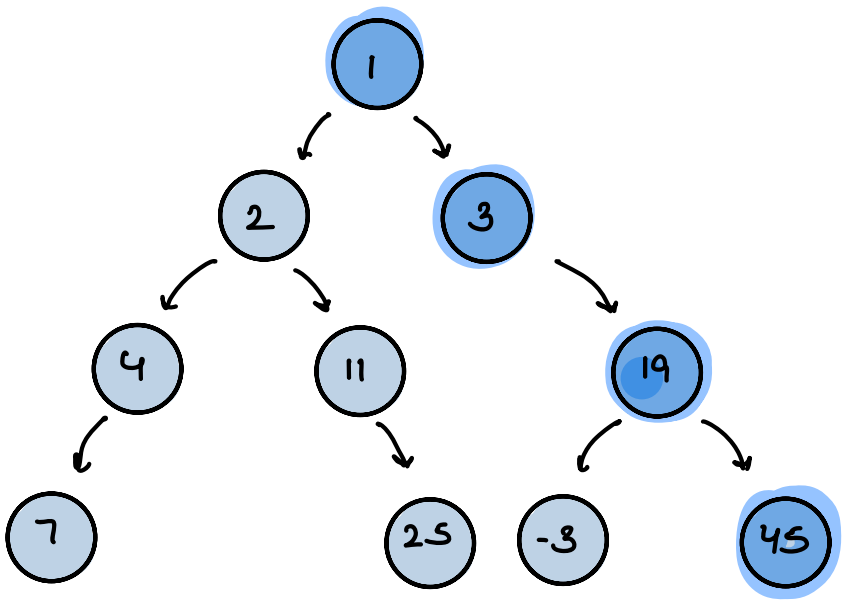
SC: O(n)

* Left view of Tree



```
Queue <Node> q ;

q. enque (root);

while (q.size() > 0) {
    int sz = q.size();
    for (i=1; i ≤ sz; i++) {
        Node rem = q.deque();
        if (i==1) print (rem.data);
        if (rem.left != null) q. enque (rem.left)
        if (rem.right != null) q. enque (rem.right);
    }
    println();
}
```

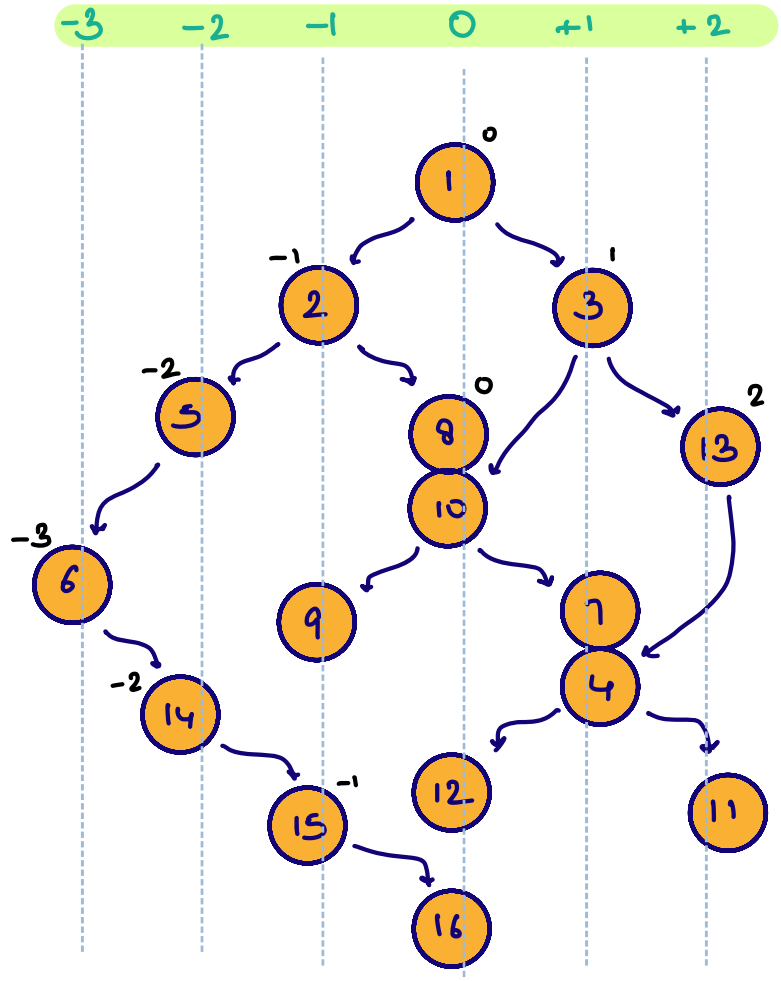
* Right view of Tree




```
if (i == sz) {
    print (rem.data)
}
```

{ TODO }

* Vertical Order Traversal (Microsoft)



Output

6

5 14

2 9 15

1 8 10 12 16

3 7 4

13 11

Recursion → Order may vary from Top to Bottom

Level order Traversal

(Iteratively)

HM

vl	Nodes
0	→ 1, 8, 10
-1	→ 2
+1	→ 3
-2	→ 5

Queue →

(1,0) (2,-1) (3,+1) (5,-2) (8,0) (10,0)

pair = Node, vl

13, 2

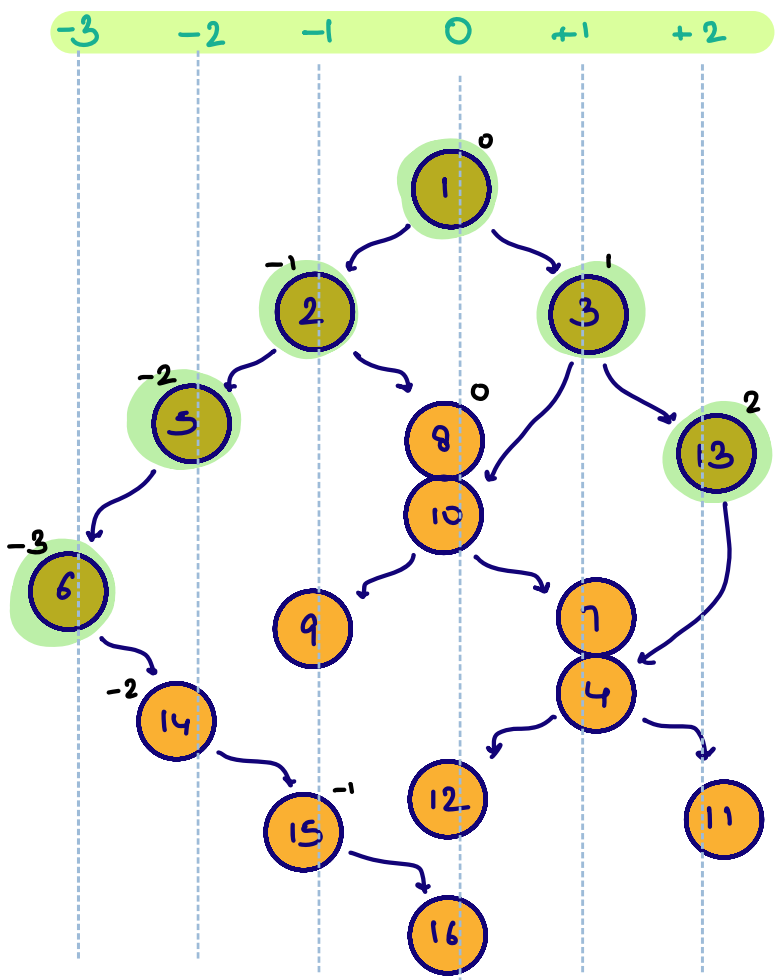
6, -3

```
class pair {
    Node node;
    int vl;
}

Queue < Pair> q ;

pair rpair = new pair(root, 0);

   HM < int, list > map;

     int minvl, maxvl = 0

    q. enque (rpair);
    while ( q.size () > 0)
        pair rp = q.deque ();
        minvl = Math.min( minvl, rp.vl)
        maxvl = Math.max(maxvl, rp.vl);
        hm.get(rp.vl) . add (rp.node);        // Handle this part carefully
        if (rp.node.left != null) {
            q. enque ( new pair (rp.node. left, rp.vl - 1);
        }
        if (rp.node.right != null) {
```

```
        q.enque(new pair(rp.node.right, rp.vl+1);
    }
}
```

```
for(i=minvl; i<=maxvl; i++){
    AL<Node> p = hm.get(i);
    for(Node n: p){
        print(n.data);
    }
    println();
}
```

* Top view of tree & Bottom view of Tree



Output

6

5 14

2 9 15

1 8 10 12 16

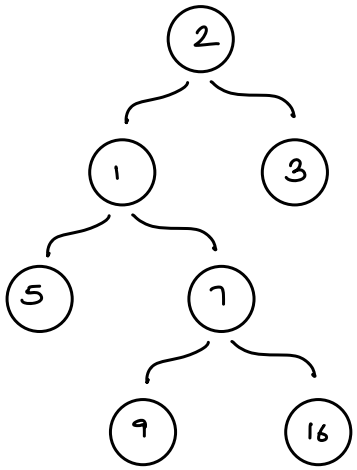
3 7 4

13 11

* Types of Binary Tree

01. **Proper / Full Binary Tree**

↳ Either 0 children or 2 children

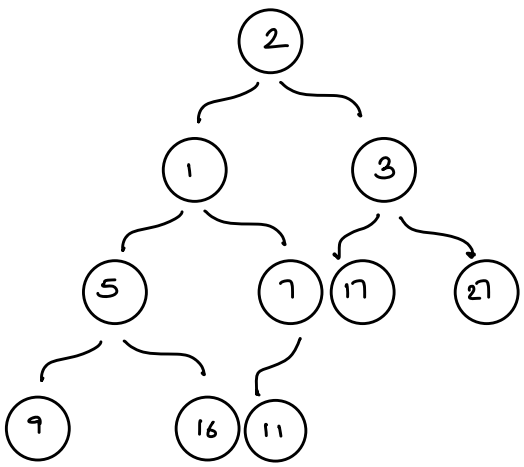


02. **Complete Binary Tree**

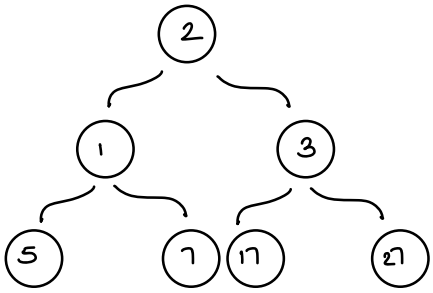
↳ All levels should be completely filled except the last level

↓

nodes should be filled from Left to Right

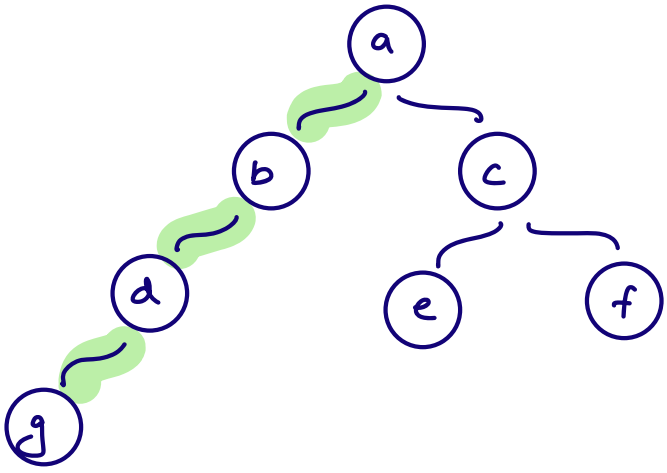


* **Perfect Binary Tree** → All levels are completely filled



* Perfect binary Tree is also a complete BT & proper BT

* Height of BT = Maximum distance b/w root node to leaf node



→ in terms of edges

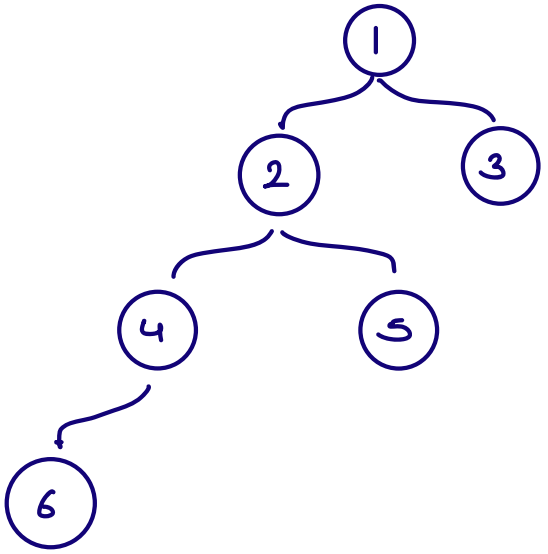
```
int height (Node root)
{
    if (root == null) return -1;
    int lh = height (root.left)
    int rh = height (root.right)
    return max(lh, rh) + 1;
}
```

* Balanced Binary Tree

For all nodes,

$$\left| \text{height of lsub tree} - \text{height of rsubtree} \right| \leq 1$$

Q Given a BT, check if it is balanced or not.




```
boolean isheightBal(root){
    if(root == null) return true;
    int lh = height(root.left);
    int rh = height(root.right);
    if(abs(lh-rh) > 1)
        return false;
    return isheightBal(root.left) &&
           isheightBal(root.right);
}
```

TC: $O(n^2)$

SC: $O(n)$

* For the type of question, where we have to calculate something that is dependend upon some other parameter, then those questions can be solved by a technique known as **Travel & change**

```
static boolean isbalanced = true;

int height (Node root)
{
    if (root == null) return -1;
    int lh = height (root.left)
    int rh = height (root.right)
    if ( abs (lh - rh) > 1) isbalanced = false;
    return max(lh, rh) + 1;
}
```

```
class pair {
    boolean isbalanced
    int ht;
}

boolean isBalanced (root) {
    isBala (root). isbalanced == true ) return T
    else return false
}

pair isBala ( root)
{
    if (root == null) {
        return new pair ( true, -1);
    }
    pair l = isBala (root.left)
```

```
pair r = isBala (root.right);

if ( l.isbalanced == false || r.isbalanced == false)
    return new pair (false, -1);
else if ( abs (l.ht - r.ht) > 1){
    return new pair (false, -1);
else{
    return new pair (true, Math.max(l.ht, r.ht) +1);
}
```