Lecture :- Trees-2

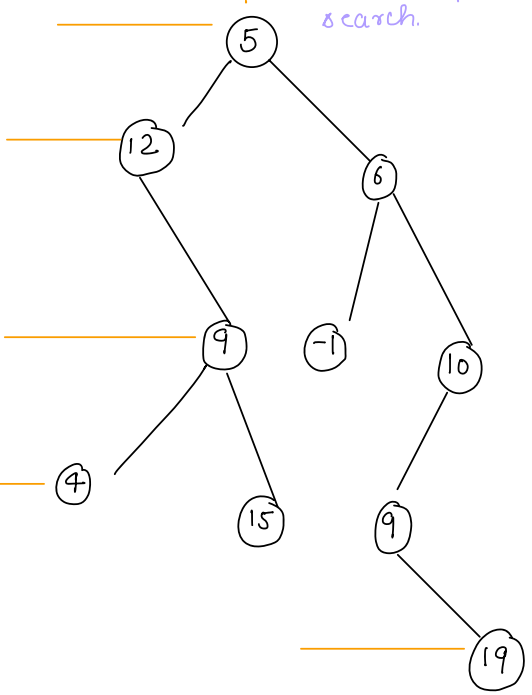
Agenda

- Level order traversal →
- Left and right view.
- Vertical level order →
- Top and bottom view
- Types of Binary tree
- Is tree height balanced? →

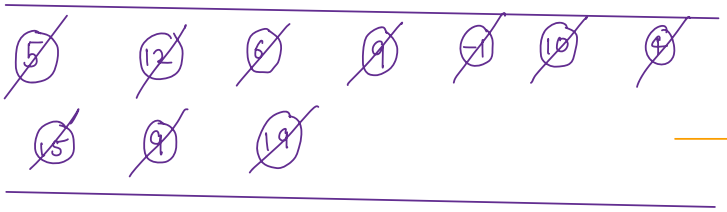
Qu1 Level order traversal.

o/p :- 5 12 6 9 -1 10
4 15 9 19

bfs : Breadth first search



Data structure: Queue.



5 12 6 9 -1 10 4 15
9 19

Code:

```
void levelOrder(Node root) {
    Queue<Node> q = new LinkedList<>();
    q.add(root);
    while(!q.isEmpty()) {
        Node curr = q.poll();
        print(curr.val);
        if(curr.left!=null) {
            q.add(curr.left);
        }
        if(curr.right!=null) {
            q.add(curr.right);
        }
    }
}
```

TC: O(n)

SC: O(n)

Qu2: Level order traversal 2.

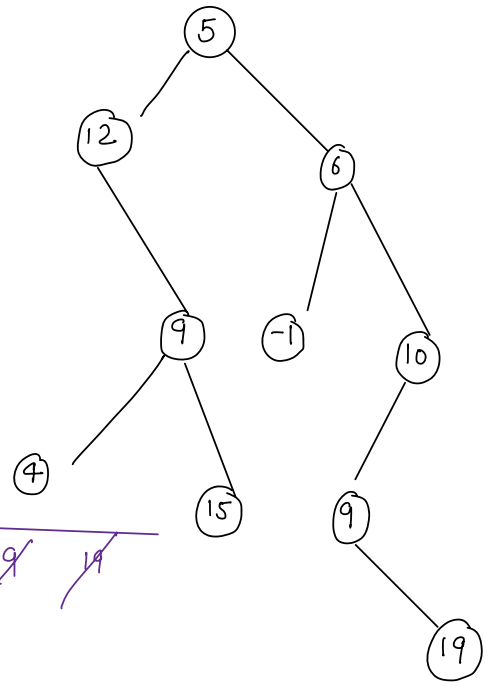
5

12 6

9 -1 10

4 15 9

19



5 12 6 9 -1 10 4 15 9 19

currLevel = 1 0 2 1 0 3 2 1 0 3 2 1 0 1 0

↑ queue.size()

Print:

5 /n

12 6 /n

9 -1 10 /n

4 15 9 /n

19

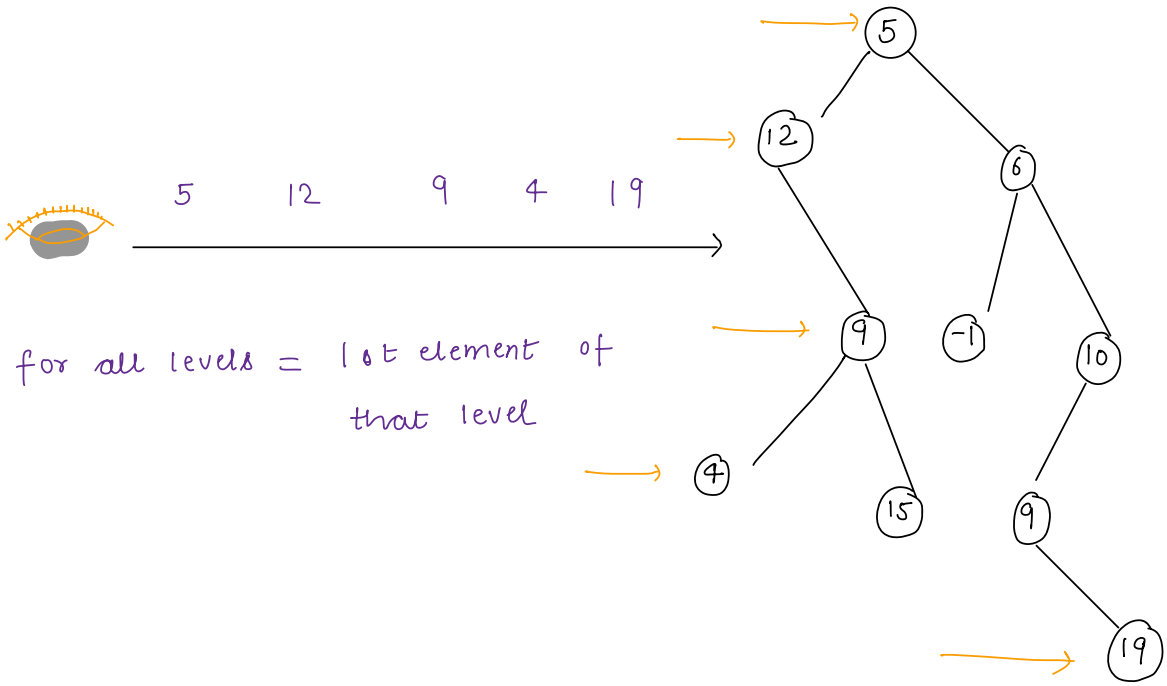
Code:

```
void levelorder( Node root) {
    Queue < Node > q = new LinkedList<>();
    q. add(root);
    while( ! q. isEmpty()) {
        currLevel = q. size();
                          currlevel
        for(i=1; i<= n; i++) {
            Node curr = q. poll();
            print (curr. val);
            if( curr. left ! =null) {
                q. add( curr. left);
            }
            if( curr. right ! =null) {
                q. add ( curr. right);
            }
        }
        System. out. println(); ← Java
        print ("\n"); → May be some other language
    }
}
```

TC: O(n)

SC: O(n)

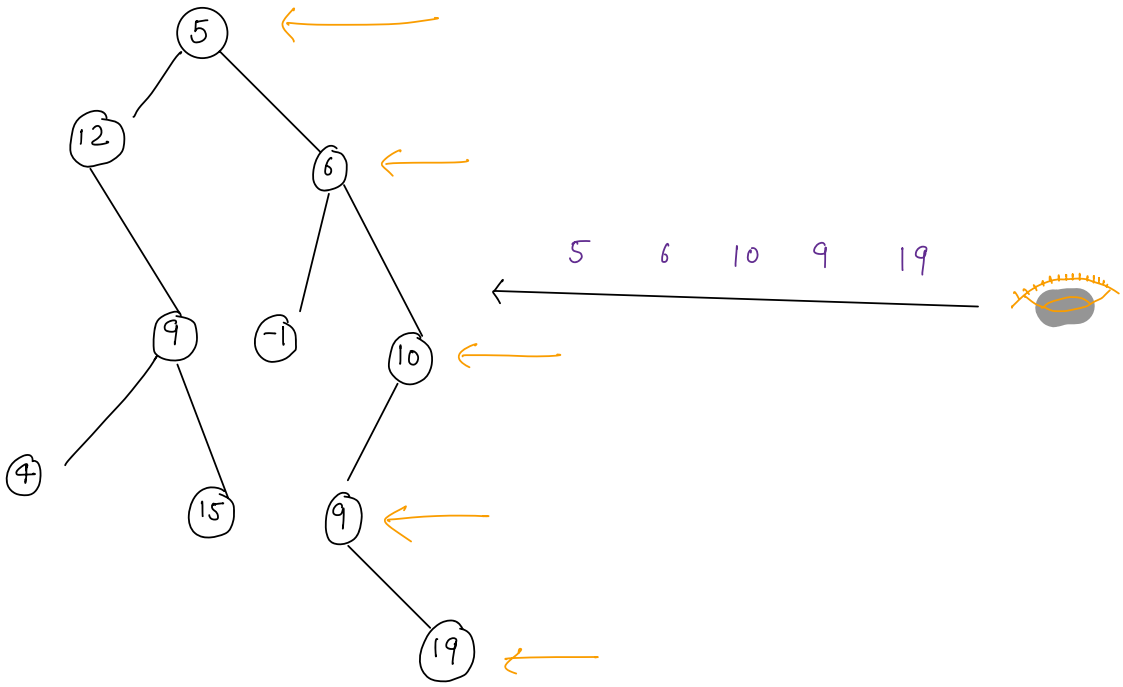
Qu3 Left view of BT.



Code:

```
void leftview (Node root) {

    Queue < Node > q = new LinkedList<>();

    q.add(root);

    while( !q.isEmpty()) {
        currLevel = q.size();
                              currLevel
        for(i=1; i<= n; i++) {
            Node curr = q.poll();
            if(i==1) {
                print (curr.val);
            }
            if( curr.left !=null) {
                q.add(curr.left);
            }
            if( curr.right !=null) {
                q.add(curr.right);
            }
        }
        System.out.println(); ← Java
        print ("\n"); → may be some other language
    }
}
```

Qu4: Right view of B.T.

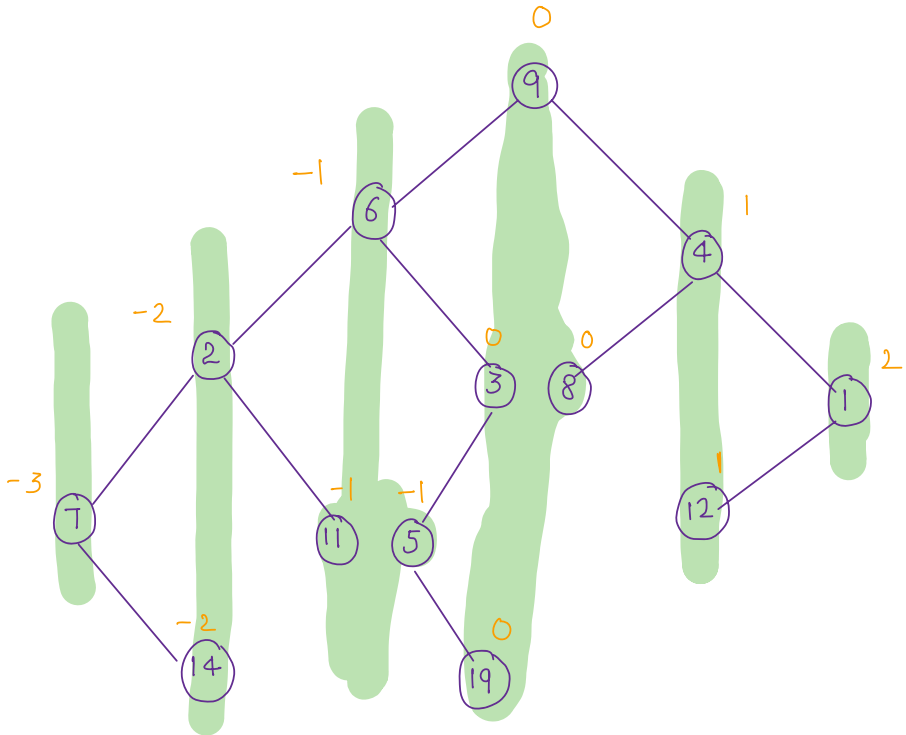


curr level = Last el of that level.

Code:-

```
void rightview ( Node root) {

    Queue < Node > q = new LinkedList<>();

    q. add(root);

    while( ! q. isEmpty()) {
        currLevel = q. size();
                                  currlevel
        for(i=1; i<= n; i++) {
            Node curr = q. poll();
            if ( i == n) {
                                  currlevel
                print (curr. val);
            }
            if( curr.left ! =null) {

                q. add ( curr.left);

            }

            if( curr. right ! =null) {

                q. add ( curr. right);

            }
        }
        system. out. println(); ← Java
        print ( "\n"); → may be some other language
    }
}
```

Qu Vertical level traversal



7

2 14

6 11 5

9 3 8 19

4 12

1

0 : 9 , 3 , 8 , 19

-1 : 6 , 11 , 5

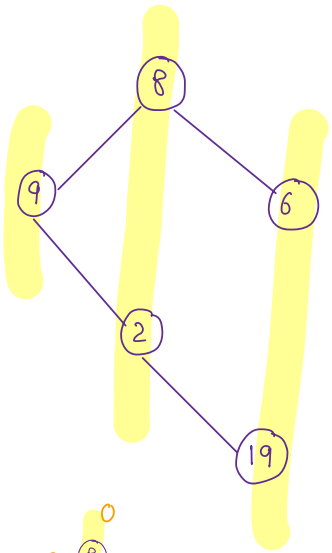
1 : 4 , 12

-2 : 2 , 14

2 : 1

-3 : 7

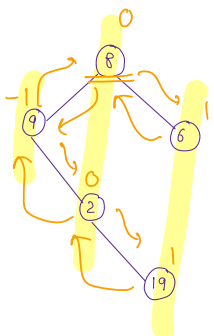
Example:



9
8 2
6 19

Preorder:

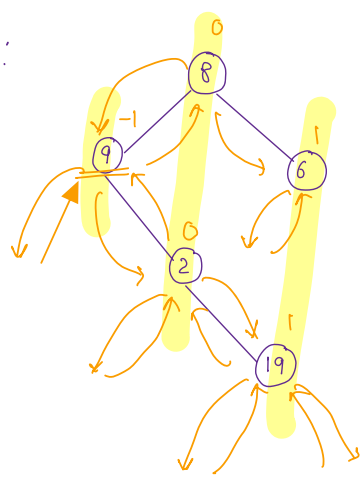
D L R



0 : 8 , 2
-1 : 9
1 : 19 , 6

X

Inorder:

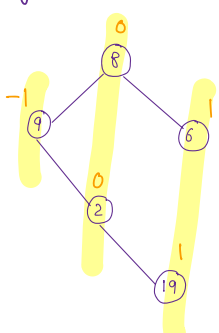


-1 : 9
0 : 2 , 8
1 : 19 , 6

X

Postorder: Try out [Aloo wrong]

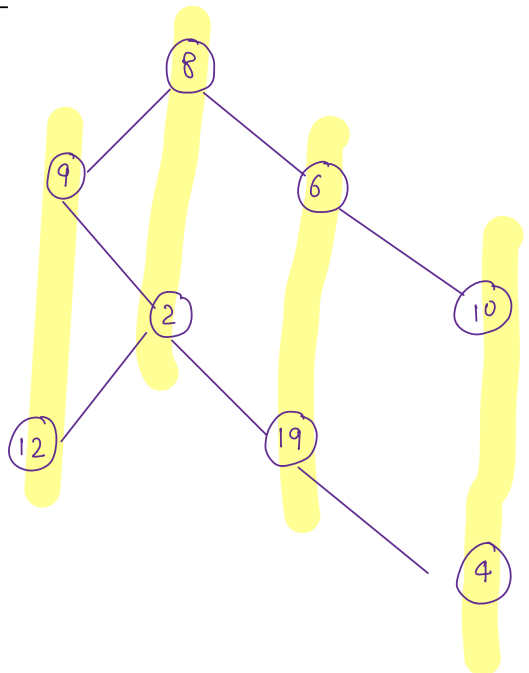
Level order:



0 : 8 , 2
-1 : 9 ,
1 : 6 , 19

✓

Logic



9 12

8 2

6 19

10 4

Map:

0 : 8, 2

-1 : 9, 12

1 : 6, 19

2 : 10, 4

Queue: | (8,0) (9,-1) (6,1) (2,0) (10,2) (12,-1) (19,1) (4,2)

minLevel = ~~0~~ -1

maxLevel = ~~0~~ 2

```
class Pair {
    Node node;
    int lvl;
    Pair (x, y) {
        node = x;
        lvl = y;
    }
}
```

Code:

```
void verticalorder(Node root) {
    Queue<Pair> q;
    Map<Integer, List<Integer>> map;
    minL=0, maxL=0;
    q.add(root, 0);
    while(!q.isEmpty()) {
        Pair f = q.poll();
        Node curr = f.node;
        int lvl = f.lvl;
        minL = min(lvl, minL);
        maxL = max(lvl, maxL);
        addToMap(map, curr.data, lvl);
        if(curr.left!=null){
            q.add(new Pair(curr.left, lvl-1));
        }
        if(curr.right!=null){
            q.add(new Pair(curr.right, lvl+1));
        }
    }
    for(i=minL; i<=maxL; i++){
        List<Integer> list = map.get(i);
        for(int el: list){
            sout(el);
        }
        soutln();
    }
}
```

Map:

0 : 8, 2
-1 : 9, 12
1 : 6, 19
2 : 10, 4

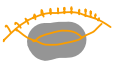
// minL = -1
// maxL = 2

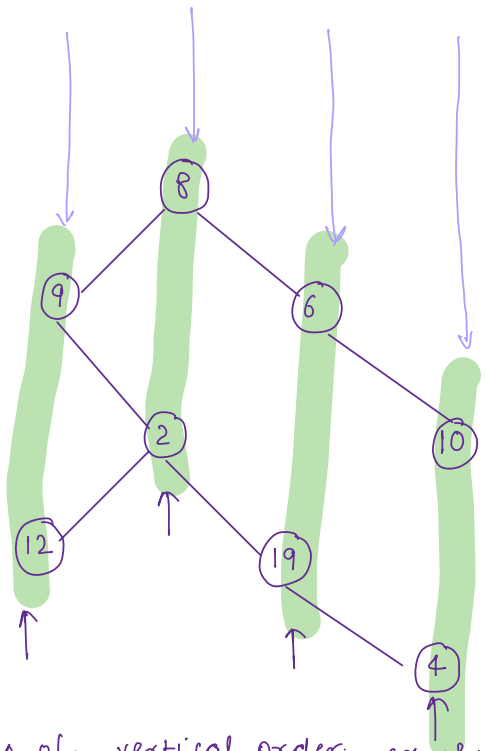
9 12
8 2
6 19
10 4

TC: O(n)
SC: O(n)

Break: 8:53

Qu Top order of B.T.





Op
9
8
6
10

Map:

0	8, 2
-1	9, 12
1	6, 19
2	10, 4

BV = 12 2 19 4

print (list.get (n-1);
)

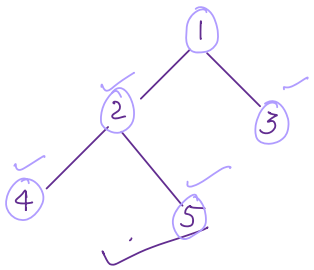
Code: Same code as of vertical order except

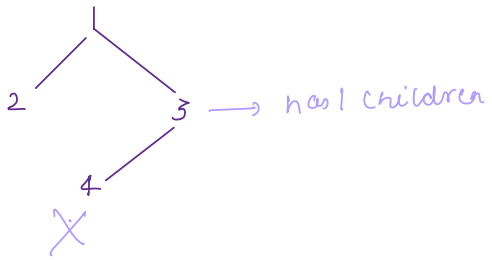
```
for ( i = minL; i <= maxL; i++ ) {
    List<Integer> list = map.get(i);
    print ( list.get(0));
}
```

* Bottom view:- TODO

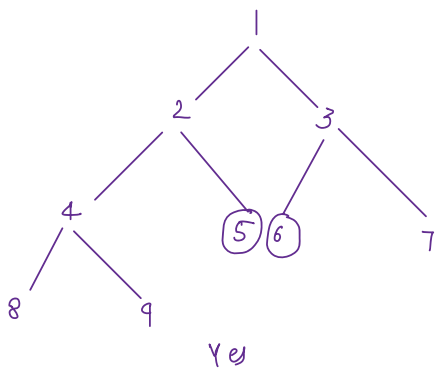
Types of BT [optional]

1.) Proper BT. : Every node should have either 0 or 2 children

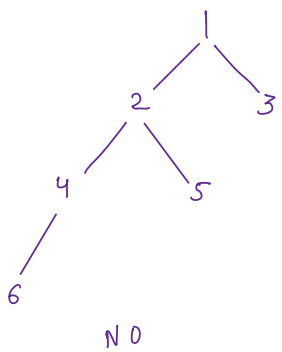




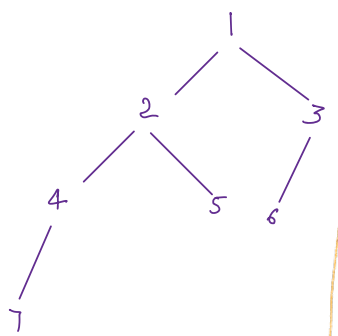
2.) Complete BT. : Nodes are arranged from left to right & top to bottom



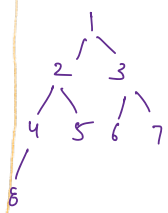
Yes



NO

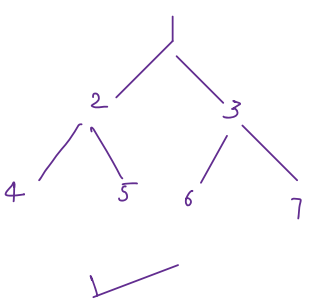


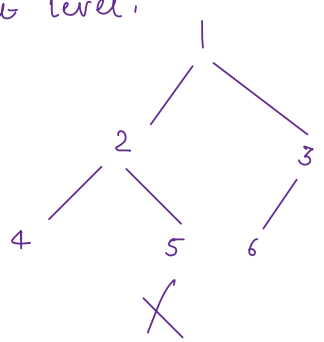
NO



Yes

3.) Perfect BT. (h/w) : All nodes are completely filled except for last level. (2 children)



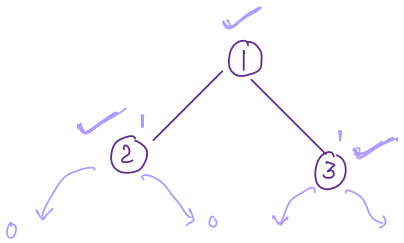


Perfect = X

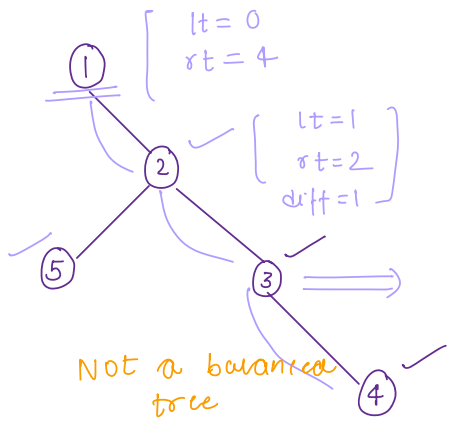
complete = ✓

proper : X

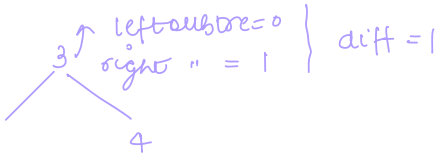
Balanced B.T. : A tree in which height b/w left and right subtree for every node is <=1.

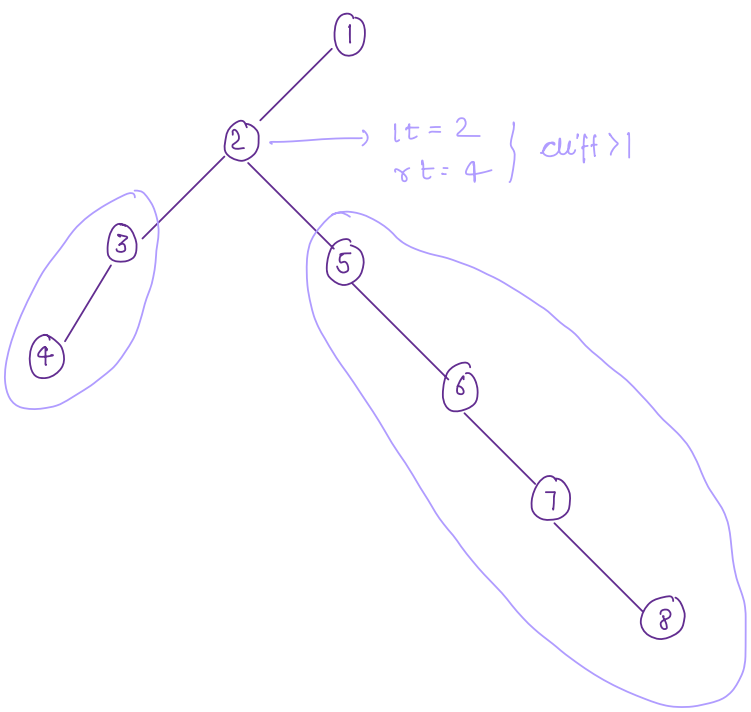


→ Balanced B.T.

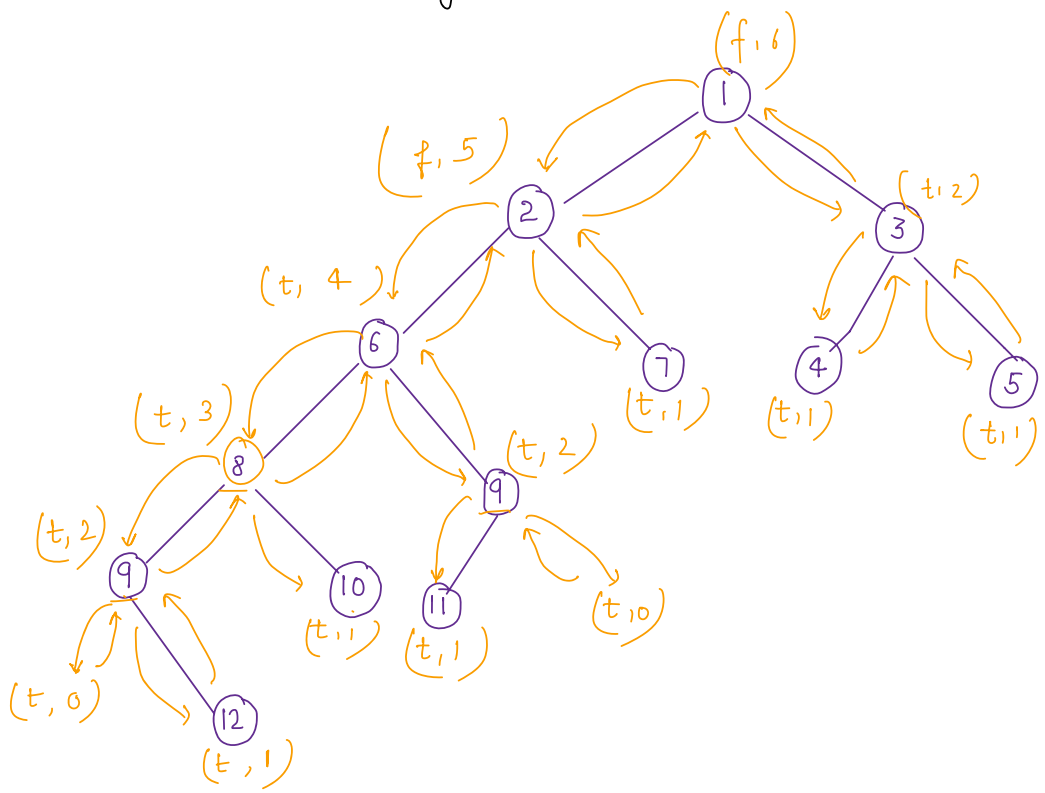


Not a balanced tree



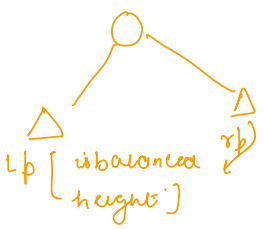


Qu: Check if tree is height balanced?




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

    Pair(x, y) {
        isBalanced = x
        height = y;
    }
}

boolean isBalanced(Node root) {
    Pair p = helper(root);
    return p.isBalanced;
}
```

```
Pair helper(Node root) {
    if(root == null) {
        return new Pair(t,0);
    }
    if(root.left == null && root.right == null) {
        return new Pair(t,1);
    }
    Pair lp = helper(root.left);
    Pair rp = helper(root.right);
    int height = max(lp.height, rp.height)+1;
    if(lp.isBalanced == false ||
       rp.isBalanced == false) {
        return new Pair(false, height);
    }
    if(Math.abs(lp.height - rp.height) > 1) {
        return new Pair(false, height);
    }
    return new Pair(true, height);
}
```



TC: O(n)

SC: O(height)

thankyou ☺