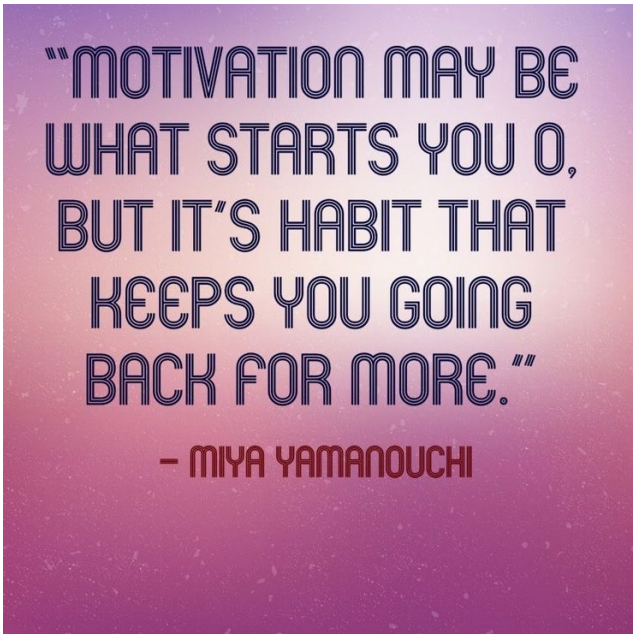
TODAY:

Trees 2

DON'T FORGET

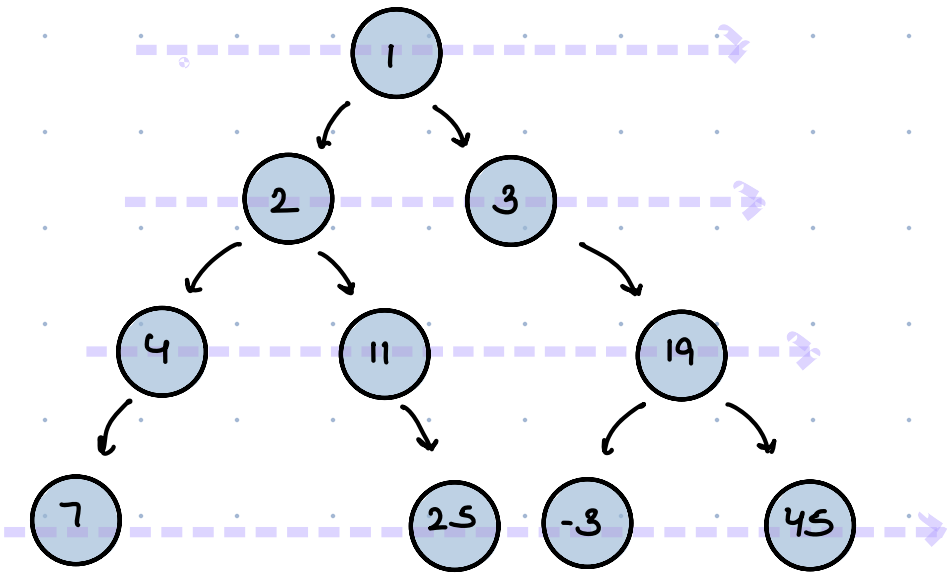


Good Evening

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 (Breadth first Search)



Queues

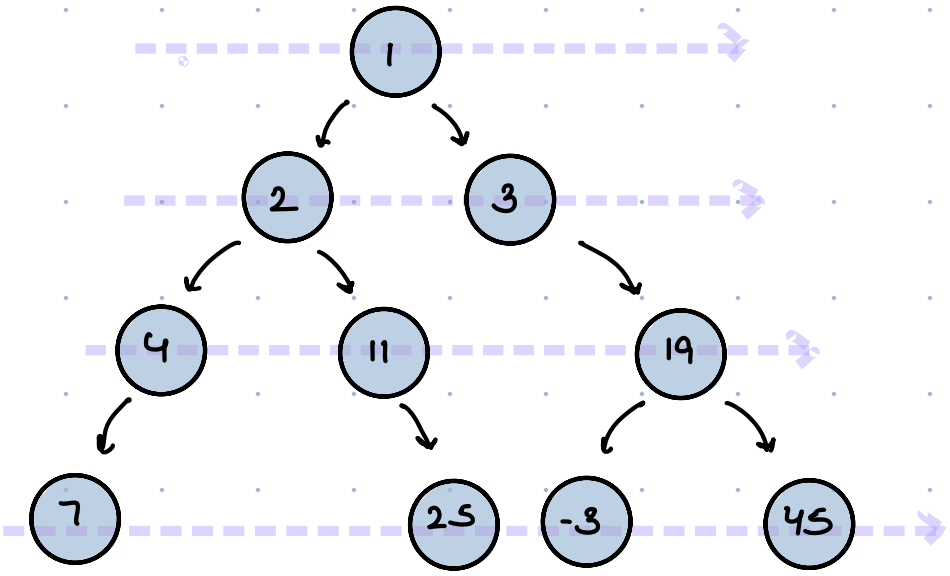
01. Remove
02. Work
03. Add the children

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

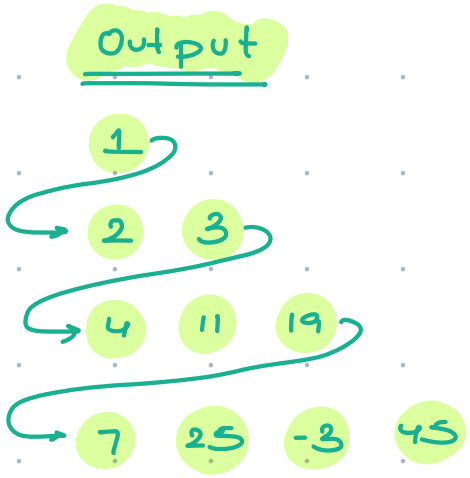
1	2	3	4	11	19	7	25	-3	45

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

q.add(root);

while(q.size()>0){
    Node rem = q.remove();
    print(rem.val);

    if(rem.left != null) q.add(rem.left)
    if(rem.right != null) q.add(rem.right);
}
```

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



Output



1	2	3	4	11	19	7	25	-3	45

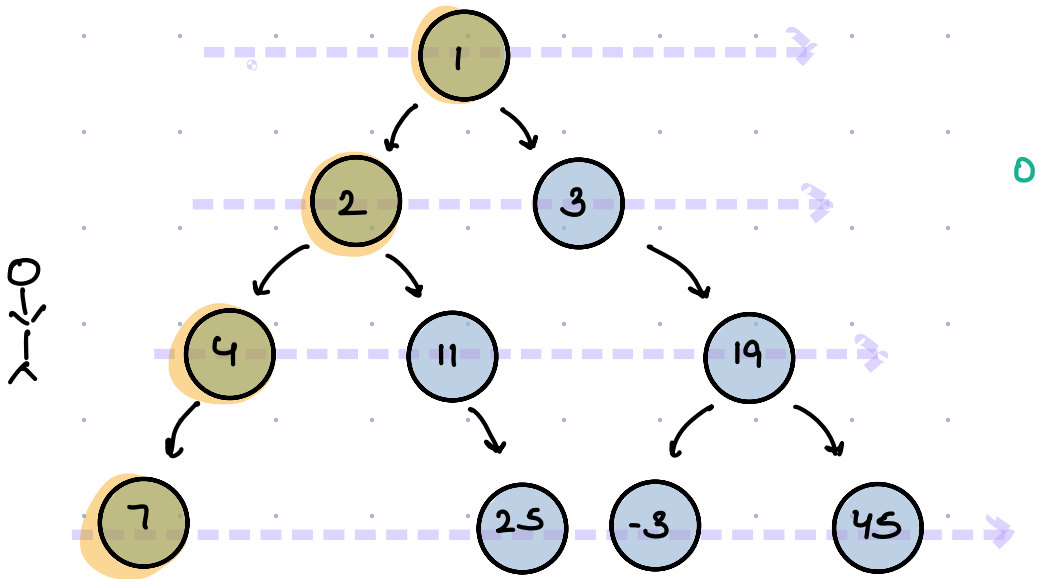
sz : ~~1~~ 0 → ~~2~~ + 0 → ~~3~~ ~~2~~ + 0 → 4 ~~3~~ ~~2~~ + 0

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

q.add(root);

while(q.size()>0){
    int sz=q.size();
    for(i=1; i<=sz; i++){
        Node rem = q.remove();
```

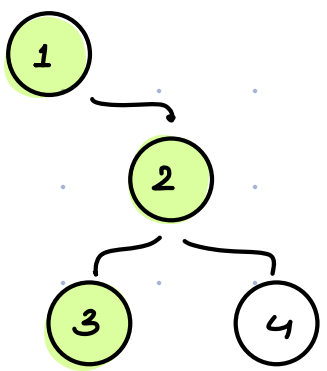
TC: O(n)
SC: O(n)

```
        print (ram.val);
        if (ram.left != null)  q.add(ram.left)
        if (ram.right != null)  q.add(ram.right);
    }
    System.out.println();
}
```

* **Left view of Binary Tree**



Output = 1 2 4 7

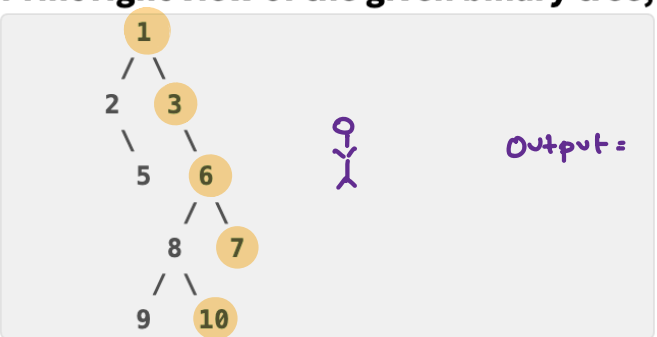


Output = 1 2 3

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

q.add(root);

while(q.size()>0){

    int sz=q.size();

    for(i=1; i<=sz; i++){

        Node rem = q.remove();

        if(i==1){ print(rem.val); }

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

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

    }

    System.out.println();

}
```

* Right View → {code by yourself}

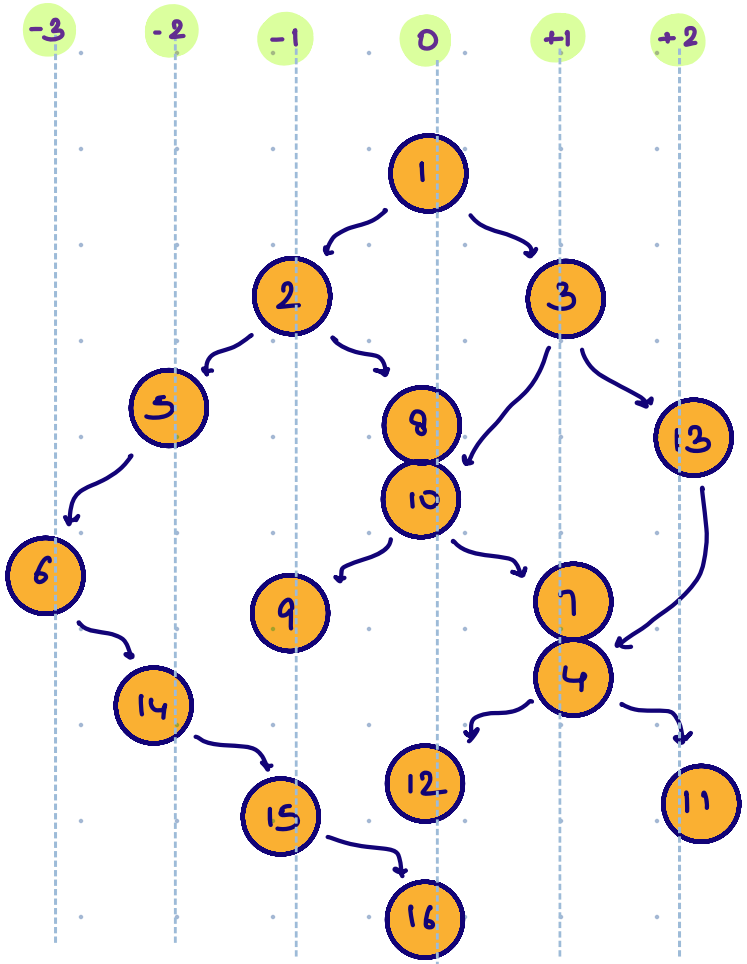
Print right view of the given binary tree,



Output = {1, 3, 6, 7, 10}

10:29 AM → 10:39 AM

Vertical Order Traversal



Output

-3 → 6

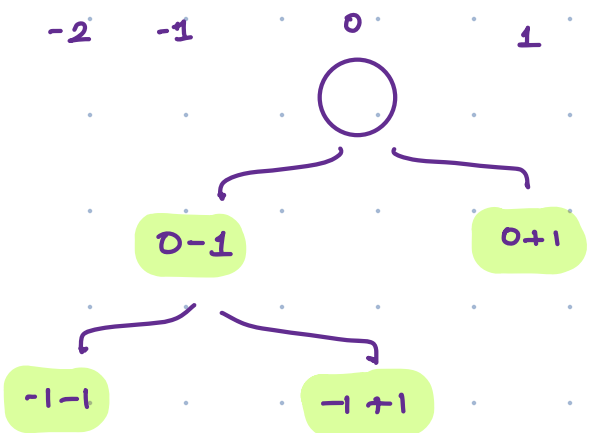
-2 → 5 14

-1 → 2 9 15

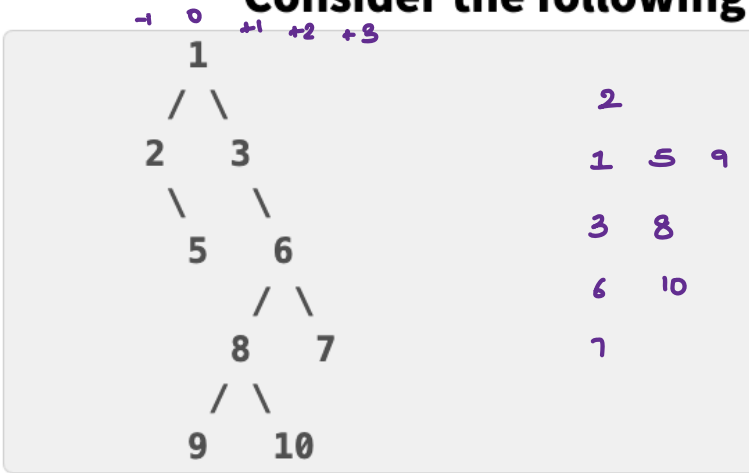
1 8 10 12 16

3 7 4

13 11

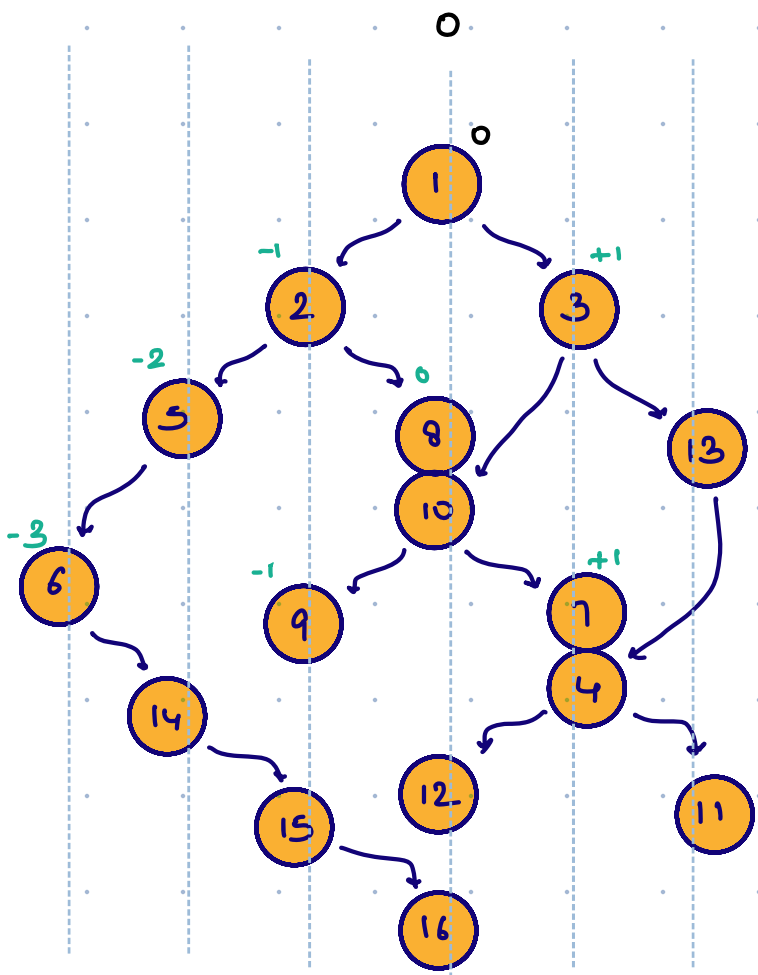


Consider the following binary tree:



Pick the vertical order traversal of the given Binary Tree.

BFS



HM

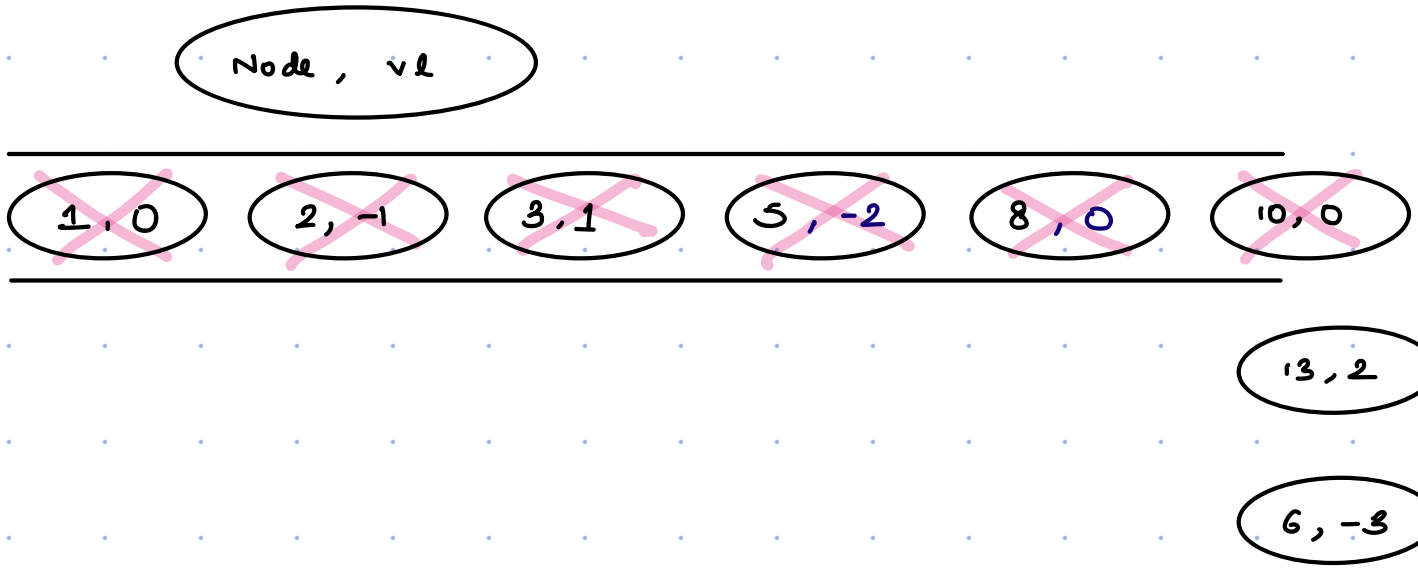
vl → AL<Node>

-2 → 5

-1 → 2

0 → 1 , 8 , 10

1 → 3

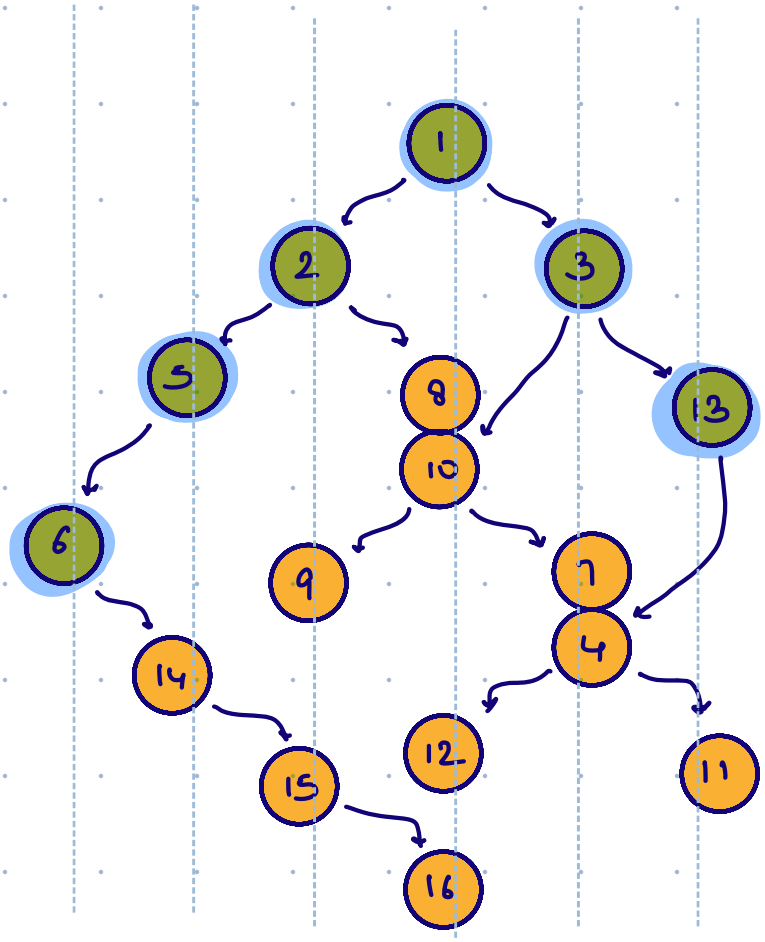



```
class pair {
    Node node
    int vl;

    pair(Node n, int v){
        node = n;
        vl = v;
    }
}
```

```
Queue < pair> q = new LinkedList <>();

HashMap < Integer, AL<Node>> hm = new HashMap<>();

minvl = 0

maxvl = 0

Pair rp = new pair(root, 0);

q.add(rp);
```

```
while (q.size() > 0) {
    pair rem = q.remove();
    Node remN = rem.node;
    int remL = rem.vl;
    minvl = Math.min(minvl, remL);
    maxvl = Math.max(maxvl, remL);
    hm.putIfAbsent(remL, new ArrayList<>());
    hm.get(remL).add(remN);

    if (remN.left != null) {
        pair lp = new pair(remN.left, remL - 1);
        q.add(lp);
    }

    if (remN.right != null) {
        pair rip = new pair(remN.right, remL + 1);
        q.add(rip);
    }
}
for (i = minvl; i <= maxvl; i++) {
    AL<Node> ans = hm.get(i)
    for (Node a : ans) {
        print(a.val);
    }
    println();
}
```

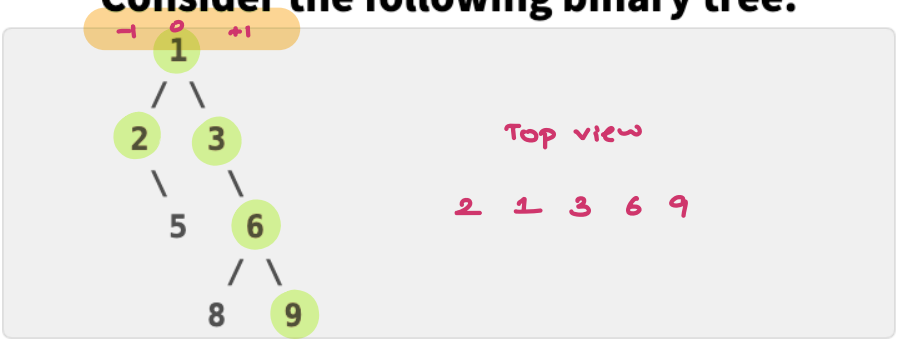
Top View of Binary Tree




```
for ( i = minvl ; i ≤ maxvl ; i++) {
    Al<Node> ans = hm.get(i)
    for (j=0; j< ans.size(); j++) {
        if (i==0) print (ans.get(j));
    }
}
```

* Bottom view of Binary Tree { code by yourself }

Consider the following binary tree:

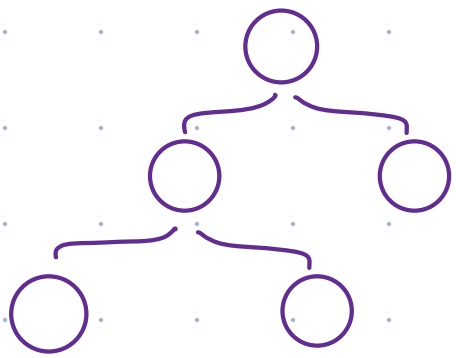


What is the top view of the given binary tree.

* Types of Binary Tree

01. Proper / Full Binary Tree

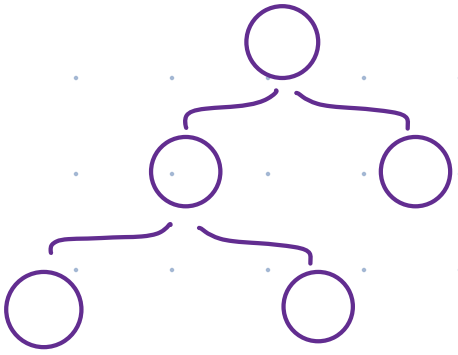
→ Every node will have either 0 child or 2 children



02. Complete Binary Tree

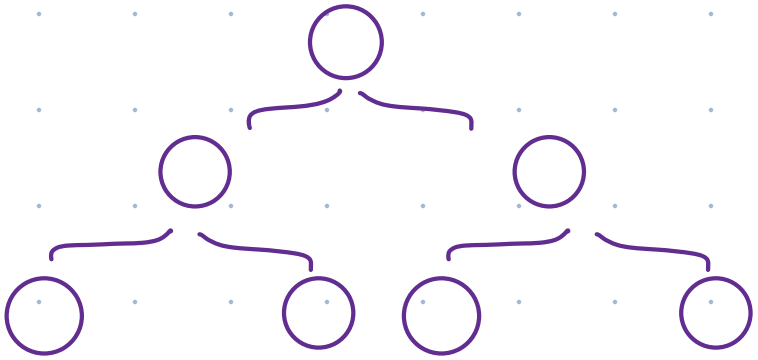
→ Every level of BT should be completely filled except the last level which may or may not be completely filled.

→ In the last level, nodes should be filled from LHS to RHS



Q3 **Perfect Binary Tree**

→ All the internal nodes have exactly two children, all the leaf nodes are on same level.



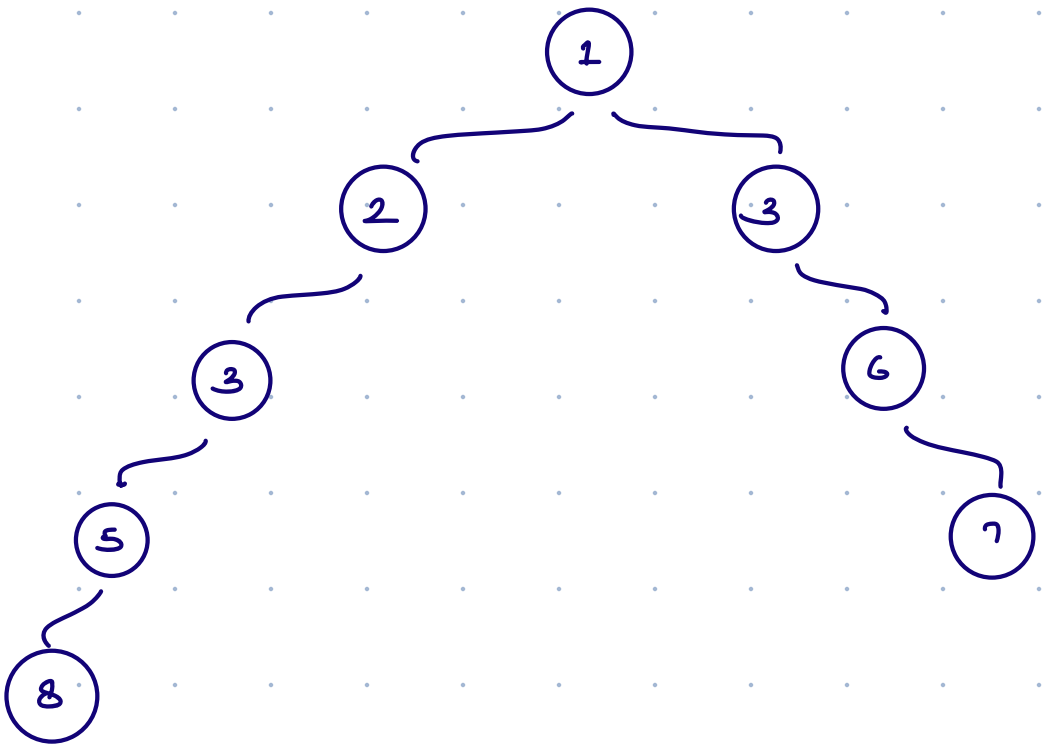
Perfect BT → Proper BT

↳ Complete BT

* Given a Binary tree, check if it is balanced or not.

For all nodes,

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



* Height of tree
* Balance or not (boolean)
  ↳ check while traversing at each & every node.