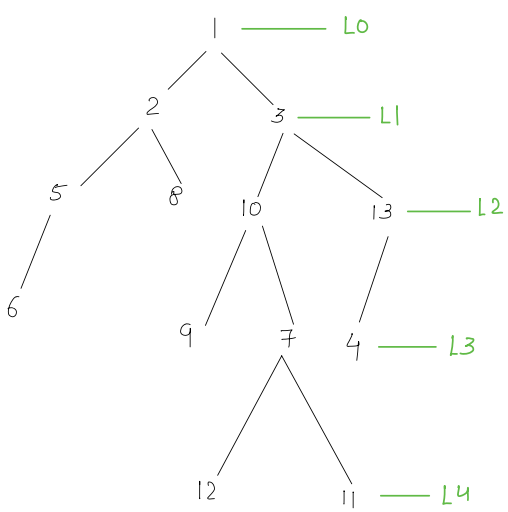
Lecture 20 Trees - 2

Agenda

- Level order traversal
- Right view of binary tree
- Vertical order traversal
- Top view of binary tree
- Height balanced binary tree.

Level order traversal | Bfs |



output

1

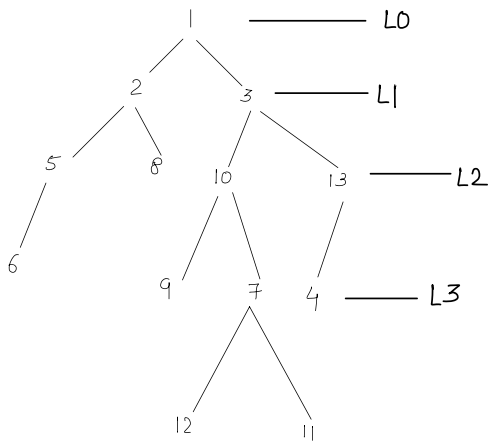
2 3

5 8 10 13

6 9 7 4

12 11

Approach



Output

1

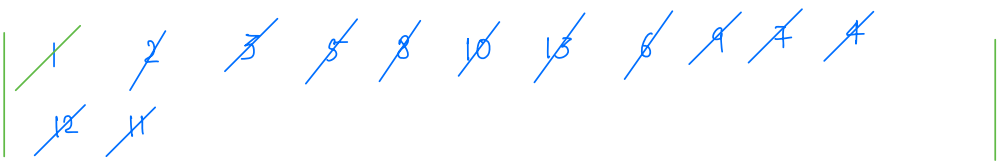
2 3

5 8 10 13

6 9 7 4

12 11

Queue



size = 1 0 2 1 0 1 3 2 1 0 1 3 2 1 0 2 1 0

1

2 3

5 8 10 13

6 9 7 4

12 11

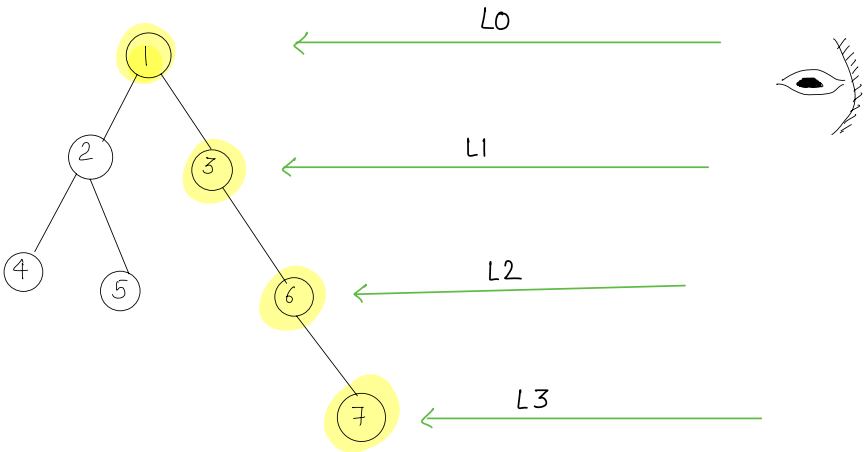
Pseudocode

```
void levelOrder(Node root) {
    if(root==null){
        return
    }
    Queue<Node> q = new LinkedList<>();
    q.add(root);

    while(!q.isEmpty()) {
        int size = q.size();
        for(i=1; i<=size; i++) {
            Node curr = q.poll();
            print(curr.data);
            if(curr.left!=null) {
                q.add(curr.left);
            }
            if(curr.right!=null) {
                q.add(curr.right);
            }
        }
        print() go to next line. [ system.out.println() ]
    }
}
```

TC: O(n)

SC: O(n)

Right view of a tree



Ob 1 3 6 7

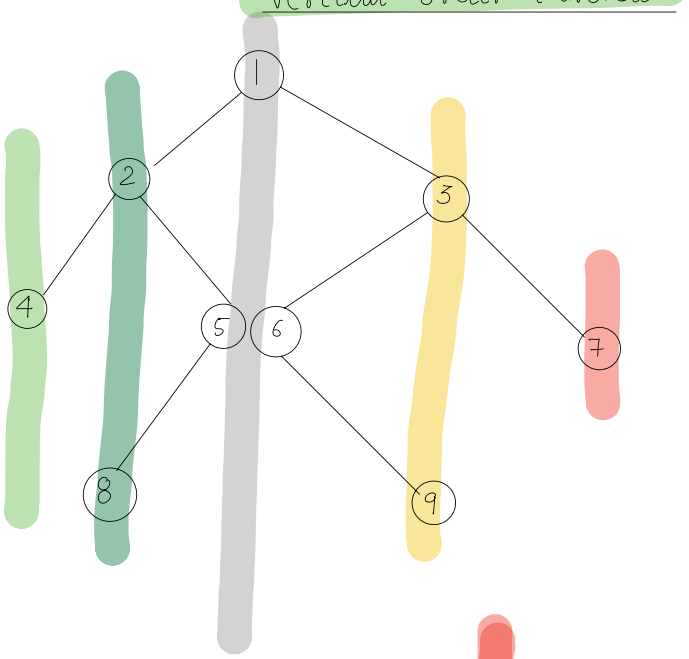
Right view :— Last el of each level

Left view :— first el of each level

Pseudocode

```
void rightView(Node root) {
    if(root==null){
        return
    }
    Queue<Node> q = new LinkedList<>();
    q.add(root);

    while( !q.isEmpty()) {
        int size = q.size();
        for(i=1; i<=size; i++) {
            Node curr = q.poll();
            if(i == size) {
                print(curr.data);
            }
            if(curr.left != null) {
                q.add(curr.left);
            }
            if(curr.right != null) {
                q.add(curr.right);
            }
        }
        print( go to next line. [ system.out.println() ]
    }
}
```

Vertical order traversal



output

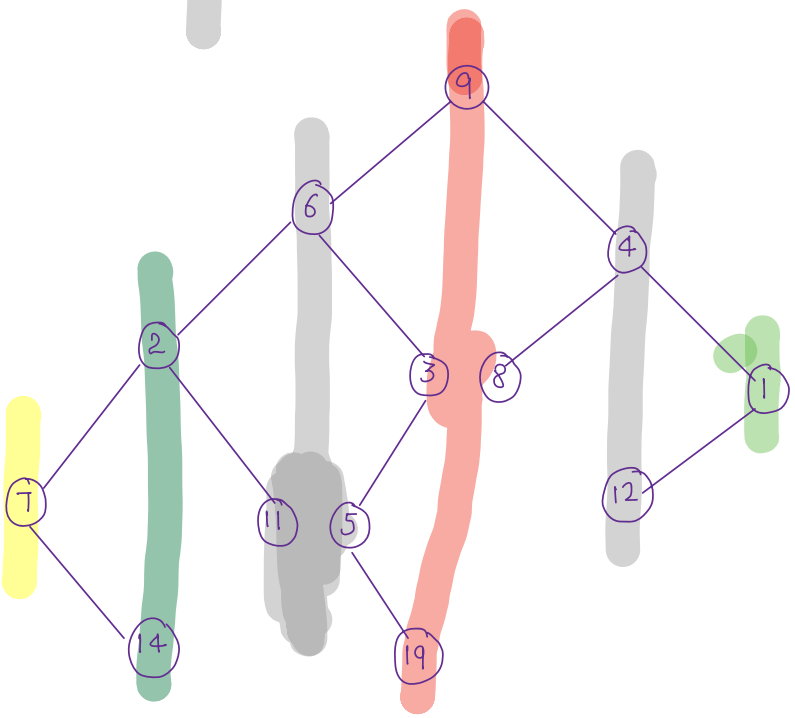
4

2 8

1 5 6

3 9

7



op

7

2 14

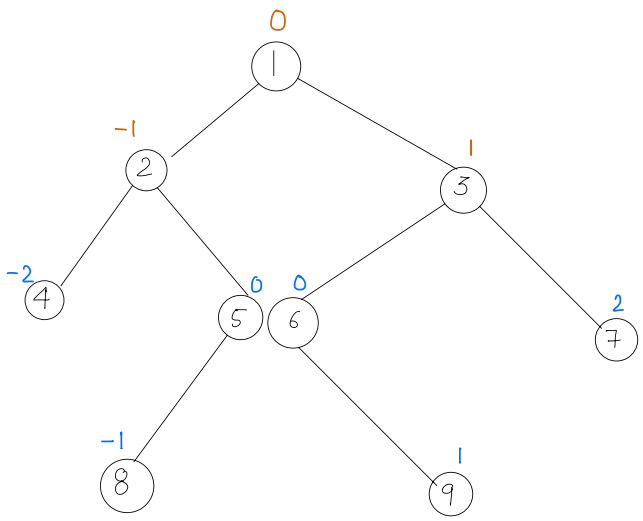
6 11 5

9 3 8 19

4 12

1

Approach

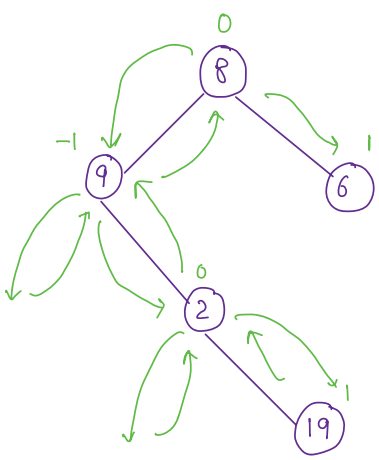


0 : 1 , 5 , 6
-1 : 2 , 8
1 : 3 , 9
-2 : 4
2 : 7

output:

4
2 8
1 5 6
3 9
7

```
class Pair {
    Node node;
    int col;
}
```



9
8 2
6 19

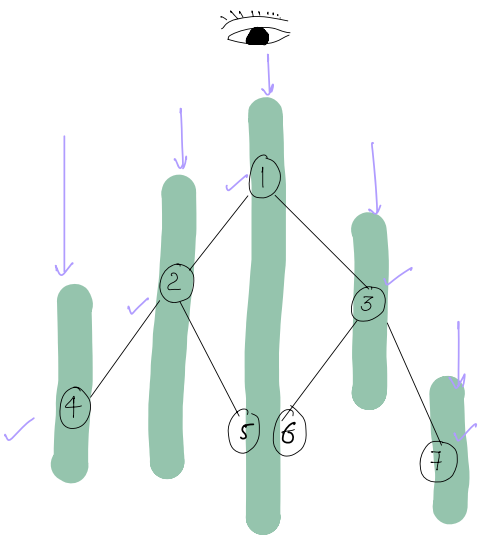
0 : 8 , 2
-1 : 9
1 : 19 , 6

Pre-order wont work

Pseudocode

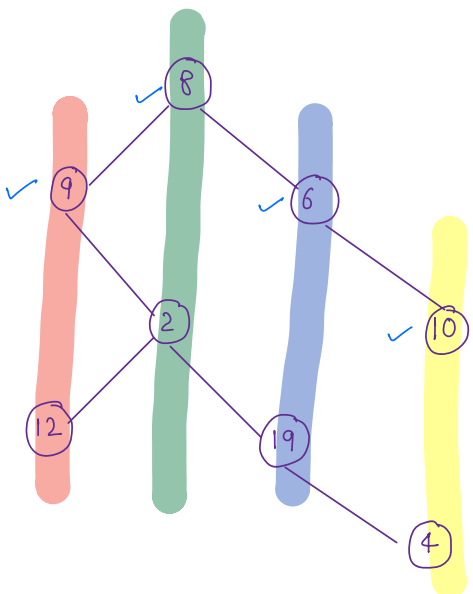
```
void verticalOrderTraversal (Node root) {
    if(root==null){
        return
    }
    Map < Integer. List<Integer>> map = new HashMap<>();
    Queue< Pair > q = new LinkedList<>();
    q. add ( new Pair (root, 0 ));
    minCol = 0 . maxCol=0.
    while ( ! q. isEmpty ()) {
        int size = q. size();
        for(i=1; i<=size ; i++) {
            Pair curr = q.poll();
            minCol = min(minCol, curr.col);
            maxCol = max(maxCol, curr.col);
            addToMap ( map, curr.col, curr. node).
            Node left = curr.node.left;
            Node right  curr.node. right;
            if( left ! = null ) {
                q.add( new Pair ( left , curr.col - 1 ));
            }
            if( right ! = null ) {
                q.add( new Pair (right, curr.col + 1 ));
            }
        }
        print| go to next line. [ system.out. println() ]
    }

    for(i=minCol; i<=maxCol ; i++) {
        List<Integer> els = map. get(i);
        for(int el: els){
            print(el);
        }
        println();
    }
}
```

Top view of binary tree



output

4
2
1
3
7



output

9
8
6
10

Pseudocode

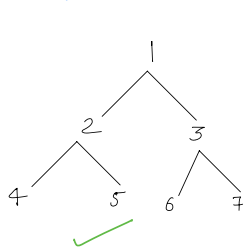
```
void topView(Node root) {
    if(root==null){
        return
    }
    Map<Integer, List<Integer>> map = new HashMap<>();
    Queue<Pair> q = new LinkedList<>();
    q.add(new Pair(root, 0));
    minCol = 0, maxCol=0.
    while( !q.isEmpty()) {
        int size = q.size();
        for(i=1; i<=size; i++) {
            Pair curr = q.poll();
            minCol = min(minCol, curr.col);
            maxCol = max(maxCol, curr.col);
            addToMap(map, curr.col, curr.node);
            Node left = curr.node.left;
            Node right = curr.node.right;
            if( left != null ) {
                q.add(new Pair(left, curr.col-1));
            }
            if( right != null ) {
                q.add(new Pair(right, curr.col+1));
            }
        }
        print(go to next line. [ system.out.println() ]
    }

    for(i=minCol; i<=maxCol; i++) {
        List<Integer> els = map.get(i);
        print(els.get(0));
        println();
    }
}
```

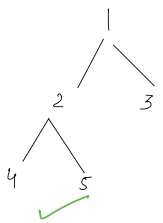
Break: 8:34 - 8:44

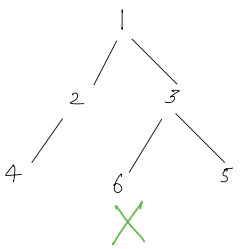
Types of binary tree

1> **Proper binary tree** | strict binary tree

Every node has 0 | 2 children.

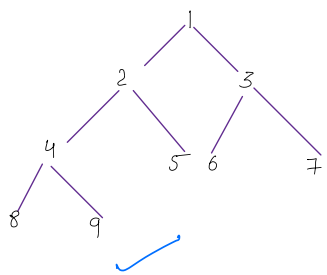


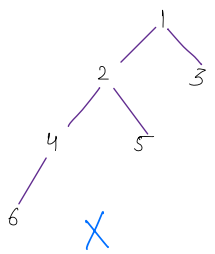


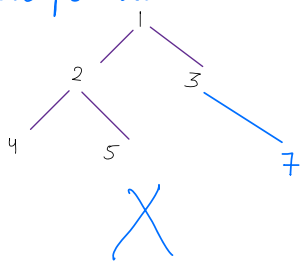


2> **Complete binary tree**

All levels must be completely filled except last level. | fill - left to ~~right~~



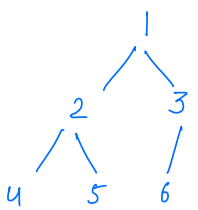


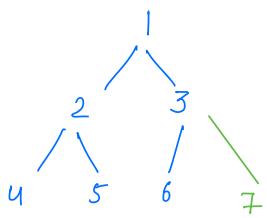


3> **Perfect binary tree**

All internal node has exactly 2 children, and all leaf nodes should be at same level [~~left to right~~]

[proper + complete]

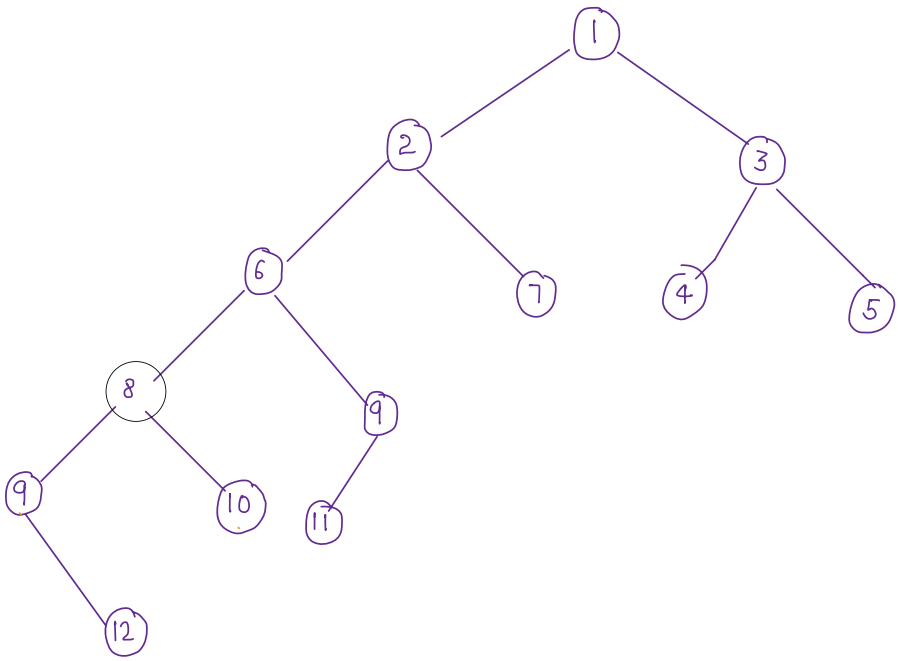


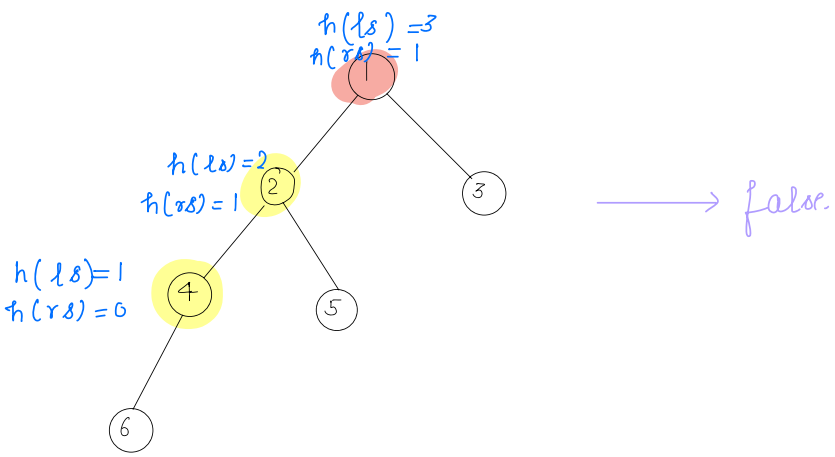


Proper	X	✓
Complete	✓	✓
Perfect	X	✓

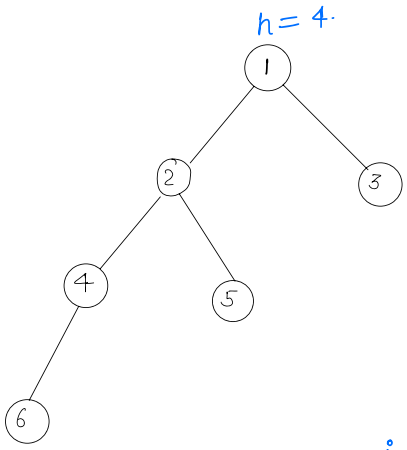
Qu Given a binary tree, check whether it is height balanced?

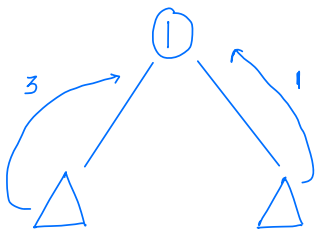
Height balanced :- for all nodes, abs(height of left subtree − height of right subtree) <= 1





→ false.

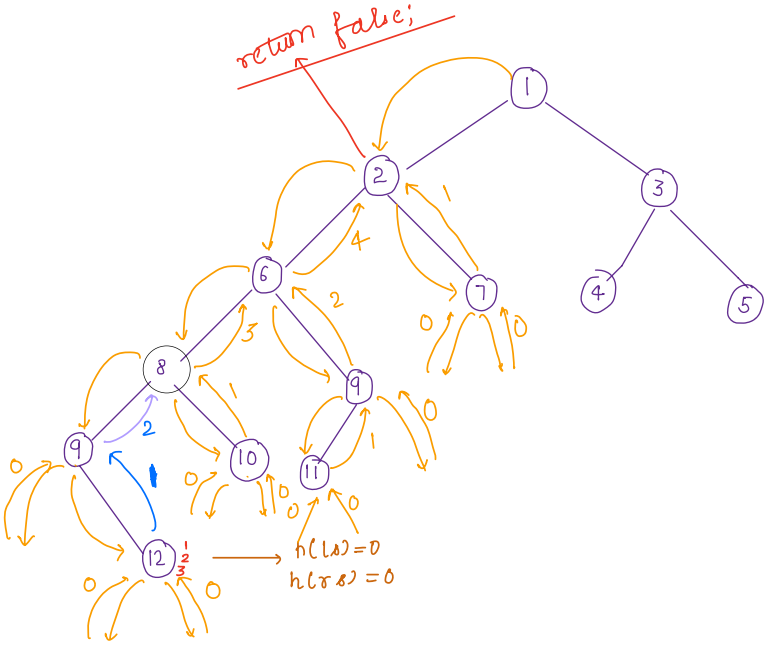




ans = max(3,1) + 1
= 4.

```
int height(Node root) {
    if(root == null) {
        return 0;
    }
    int ls = height(root.left);
    int rs = height(root.right);
    int h = max(ls, rs)+1;
    return h;
}
```

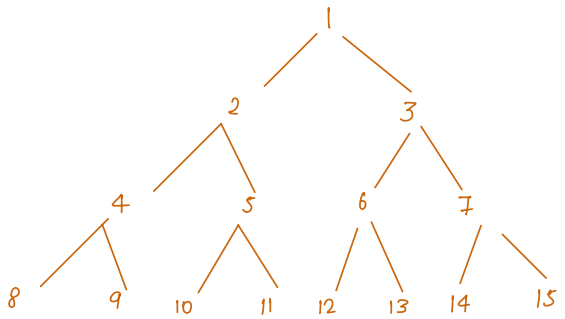
TC: O(n)




```
int height(Node root) {
    1 if(root == null) {
        return 0;
      }
    2 int ls = height(root.left);
    3 int rs = height(root.right);
    4 int h = max(ls, rs)+1
    5 return h;
}
```

```
int height(Node root) {
    if(root == null) {
        return 0;
    }
    int ls = height(root.left);
    if(ls == -1){
        return -1;
    }
    int rs = height(root.right);
    if(rs == -1){
        return -1;
    }
    int diff = abs(ls - rs);
    if(diff > 1) {
        return -1;
    }
    int h = max(ls, rs) + 1;
    return h;
}

boolean isheightBalanced(Node root) {
    int h = height(root);
    if(h == -1) {
        return false;
    }
    return true;
}
```

Doubts



In perfect tree. —

no of nodes in last level = $\frac{\text{total node}}{2}$.