Tree Traversal Questions

1. **Print TopView and Bottom View by Iterative Way (Hint : Queue)**

CODE:

```
class Solution
{ static ArrayList<Integer> topView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new
        TreeMap<>(); Queue<Pair> q = new
        LinkedList<Pair>(); q.add(new Pair(root,
        0)); while(!q.isEmpty()) { Pair it =
        q.remove(); int hd = it.hd; Node temp =
        it.node;
            if(map.get(hd) == null) map.put(hd, temp.data);
            if(temp.left != null) {

                q.add(new Pair(temp.left, hd -
            1)); } if(temp.right != null) {

                q.add(new Pair(temp.right, hd +
        1)); } }

        for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
            ans.add(entry.getValue());

   } return ans;



class Solution
{ public ArrayList <Integer> bottomView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new
        TreeMap<>(); Queue<Node> q = new
        LinkedList<Node>(); root.hd = 0;
        q.add(root);
        while(!q.isEmpty()) { Node
            temp = q.remove(); int hd
            = temp.hd; map.put(hd,
            temp.data); if(temp.left
```

```
            != null) { temp.left.hd =
            hd - 1; q.add(temp.left);
            } if(temp.right != null) {
            temp.right.hd = hd + 1;
            q.add(temp.right);
            }
        }

        for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
            ans.add(entry.getValue());
        } return
        ans;

    }

   }
```

2.Diagonal View of a Tree

CODE:

```
    public static void diagonalPrint(Node root){
        if (root == null) return;

        TreeMap<Integer, List<Integer> > map = new TreeMap<Integer,
List<Integer> >();

        Queue<TNode> q = new LinkedList<TNode>();
        q.add(new TNode(root, 0));

        while (!q.isEmpty()) { TNode curr = q.poll();
            map.putIfAbsent(curr.level, new
            ArrayList<>());
            map.get(curr.level).add(curr.node.data);

            if (curr.node.left != null)
                q.add(new TNode(curr.node.left, curr.level + 1));

            if (curr.node.right != null)
                q.add(new TNode(curr.node.right, curr.level));
        }

        for (Map.Entry<Integer, List<Integer> > entry : map.entrySet()) {
            int k = entry.getKey();

            List<Integer> l = map.get(k);
            int size = l.size();
```

```
            for (int i = 0; i < l.size(); i++) {
                System.out.print(l.get(i));
                System.out.print(" ");
            }
            System.out.println("");
        }
        return;
}
```

3. Boundary Traversal of a Binary Tree

CODE:

```
static Boolean isLeaf(Node root) { return (root.left ==
     null) && (root.right == null);
 }
 static void addLeftBoundary(Node root, ArrayList < Integer > res) {
     Node cur = root.left; while (cur != null) { if (isLeaf(cur) ==
     false) res.add(cur.data); if (cur.left != null) cur = cur.left;
     else cur = cur.right;
     } }
 static void addRightBoundary(Node root, ArrayList < Integer > res) { Node
     cur = root.right;
     ArrayList < Integer > tmp = new ArrayList < Integer > ();
     while (cur != null) { if (isLeaf(cur) == false)
     tmp.add(cur.data); if (cur.right != null) cur = cur.right;
     else cur = cur.left;
     } int i; for (i = tmp.size() - 1; i >= 0;
     --i) {
         res.add(tmp.get(i));
     }
 } static void addLeaves(Node root, ArrayList < Integer > res) {
 if (isLeaf(root)) { res.add(root.data); return;
     } if (root.left != null) addLeaves(root.left, res);
     if (root.right != null) addLeaves(root.right, res);
 }

 static ArrayList < Integer > printBoundary(Node node) {
     ArrayList < Integer > ans = new ArrayList < Integer > ();
     if (isLeaf(node) == false) ans.add(node.data);
     addLeftBoundary(node, ans); addLeaves(node, ans);
     addRightBoundary(node, ans); return ans;

}
```

4. Convert a Binary Tree into Sum Tree & Check Tree is Balanced or Not?

CODE:

```
public static int toSumTree(Node root)
{
     if (root != null) { int l =
        toSumTree(root.left); int r =
        toSumTree(root.right); int
        temp = root.data; root.data =
        l + r; return temp + l + r;
    } else
    return 0;
}
class Solution { boolean ans = true; public
    boolean isBalanced(TreeNode root) {
    solve(root); return ans;
    }

    public int solve(TreeNode root) {
        if (root == null) return 0;

        int left = solve(root.left); int right =

        solve(root.right); if (Math.abs(left -

        right) > 1) ans = false;

        return Math.max(left, right) + 1;
    }

}
```