**Name: Ravi Prakash Singh Uid: 22BCS14299**

**Section : FL\_IOT\_602/A**

[**94. Binary Tree Inorder Traversal**](https://leetcode.com/problems/binary-tree-inorder-traversal/)

Sol: class Solution {

    List<Integer> res = new ArrayList<>();

    public List<Integer> inorderTraversal(TreeNode root) {

        if(root==null)return new ArrayList<>();

        inorderTraversal(root.left);

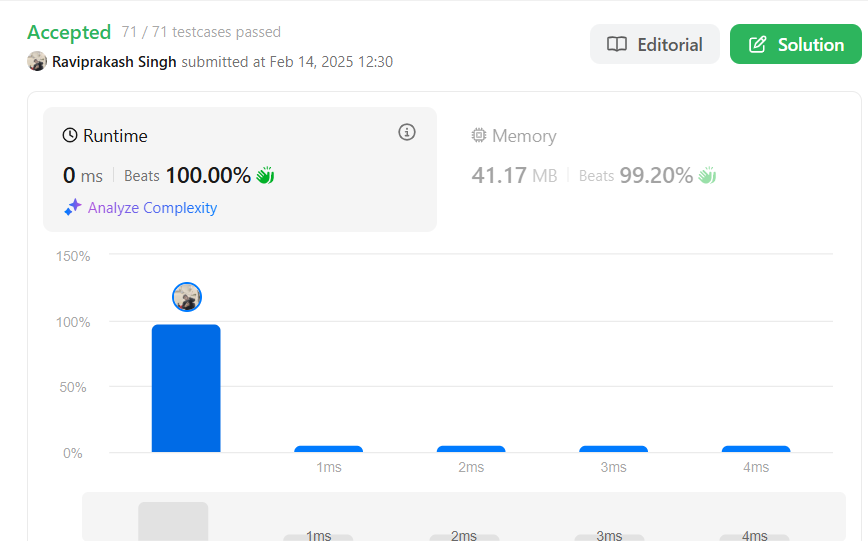
        res.add(root.val);

        inorderTraversal(root.right);

        return res;

    }

}

****

[**101. Symmetric Tree**](https://leetcode.com/problems/symmetric-tree/)

Sol: class Solution {

    boolean check(TreeNode left , TreeNode right){

        if(left==null && right==null)return true;

        if(left==null || right == null) return false;

        if(left.val != right.val) return false;

        return check(left.left,right.right) && check(left.right,right.left);

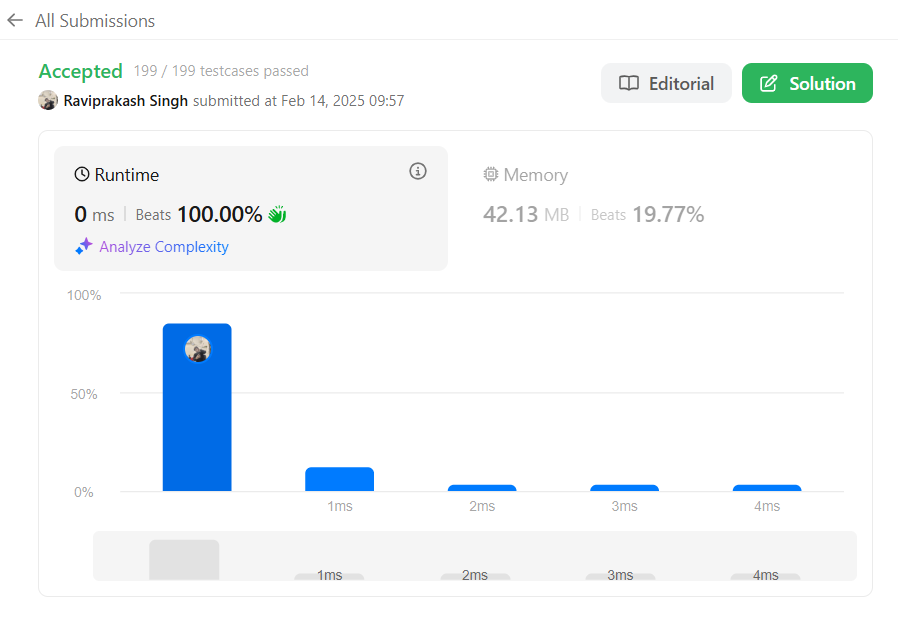
    }

    public boolean isSymmetric(TreeNode root) {

        return check(root,root);

    }

}



[**104. Maximum Depth of Binary Tree**](https://leetcode.com/problems/maximum-depth-of-binary-tree/)

Sol:

class Solution {

    public int maxDepth(TreeNode root) {

        if(root==null) return 0;

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

    }

}

[**98. Validate Binary Search Tree**](https://leetcode.com/problems/validate-binary-search-tree/)

Sol:

class Solution {

    public void inorder(TreeNode root, ArrayList<Integer> ans){

        if(root==null) return;

        inorder(root.left,ans);

        ans.add(root.val);

        inorder(root.right,ans);

        return;

    }

    public boolean isValidBST(TreeNode root) {

        ArrayList<Integer> ans = new ArrayList<>();

        inorder(root,ans);

        for(int i =0 ; i<ans.size()-1;i++){

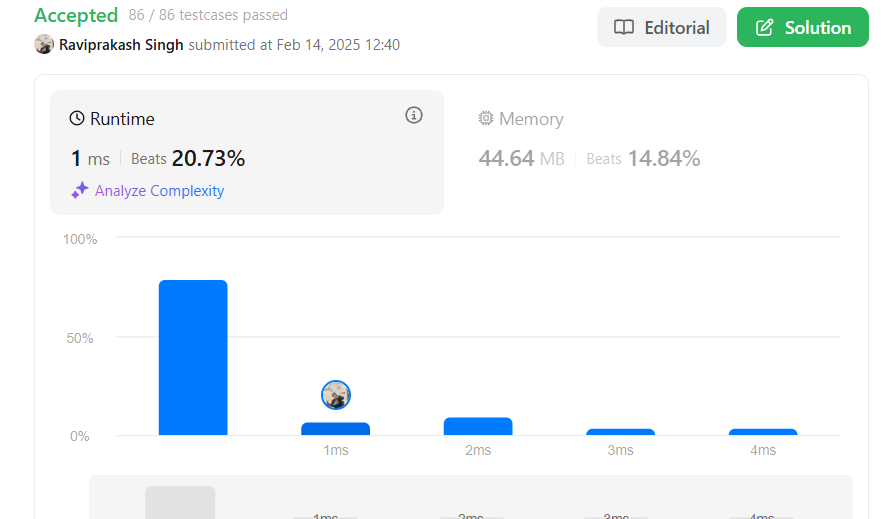
            if(ans.get(i)>=ans.get(i+1)) return false;

        }

        return true;

    }

}

****

[**230. Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

Sol: class Solution {

    public ArrayList<Integer> inorderTraversal(ArrayList<Integer> result , TreeNode root){

        if(root==null) return result;

        inorderTraversal(result,root.left);

        result.add(root.val);

        inorderTraversal(result,root.right);

        return result;

    }

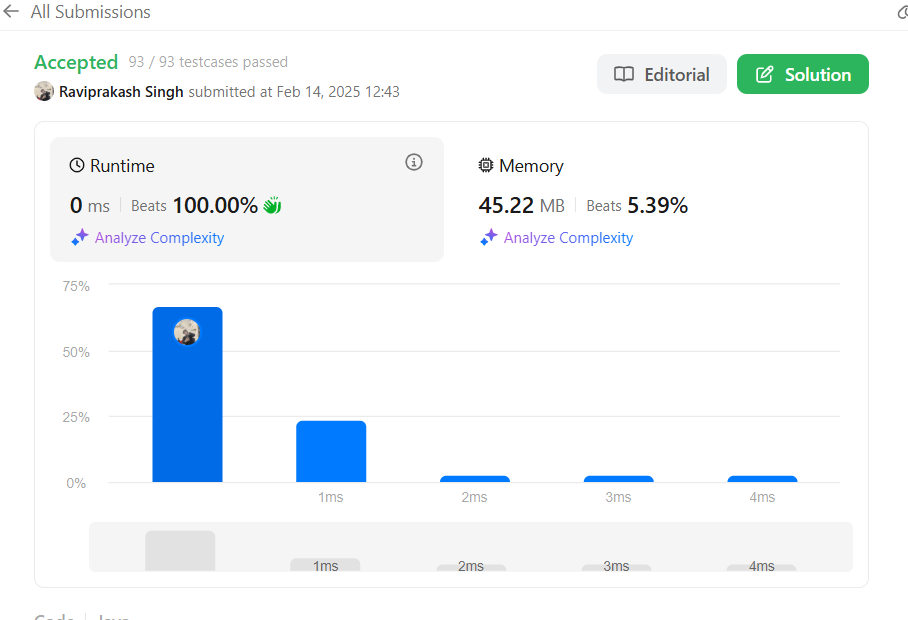
    public int kthSmallest(TreeNode root, int k) {

        ArrayList<Integer> result = inorderTraversal(new ArrayList<>() , root);

        return result.get(k-1);

    }

}



[**102. Binary Tree Level Order Traversal**](https://leetcode.com/problems/binary-tree-level-order-traversal/)

Sol: class Solution {

    public List<List<Integer>> levelOrder(TreeNode root) {

        if(root==null) return new ArrayList<>(new ArrayList<>());

        List<List<Integer>> res = new ArrayList<>();

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

        q.add(root);

        while(!q.isEmpty()) {

        List<Integer> l = new ArrayList<>();

         int count = q.size();

          for(int i= 0 ; i<count ; i++){

            TreeNode x = q.remove();

            l.add(x.val);

            if(x.left!=null) q.add(x.left);

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

          }

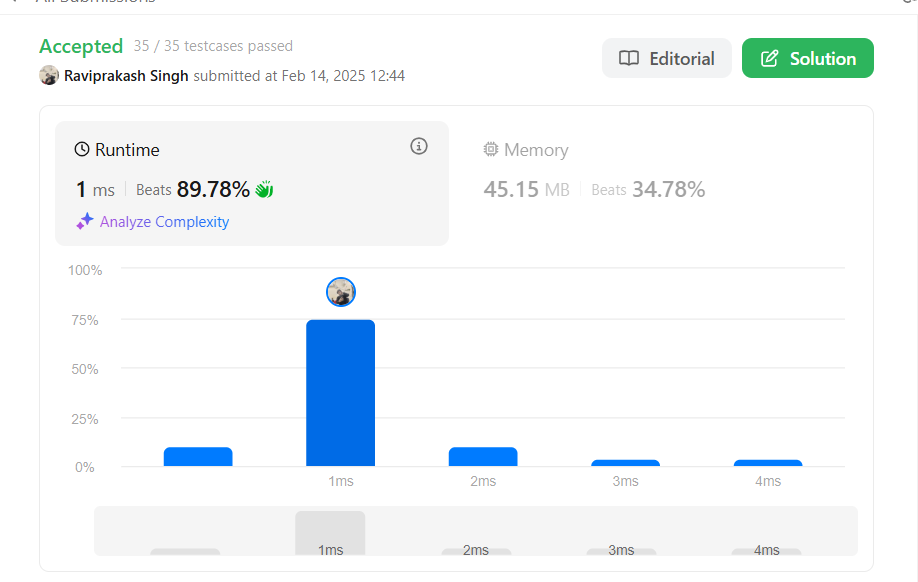
          res.add(l);

        }

        return res;

    }

}



[**103. Binary Tree Zigzag Level Order Traversal**](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/)

Sol:

class Solution {

    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

        List<List<Integer>> ans = new ArrayList<>();

        if(root==null) return ans;

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

        int level = 0;

        q.add(root);

        while(!q.isEmpty()){

            int size = q.size();

            List<Integer> lev = new ArrayList<>();

            for(int i = 0 ; i<size ; i++){

                TreeNode node = q.poll();

                lev.add(node.val);

                if(node.left!=null)q.add(node.left);

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

            }

            if(level%2!=0)Collections.reverse(lev);

            ans.add(lev);

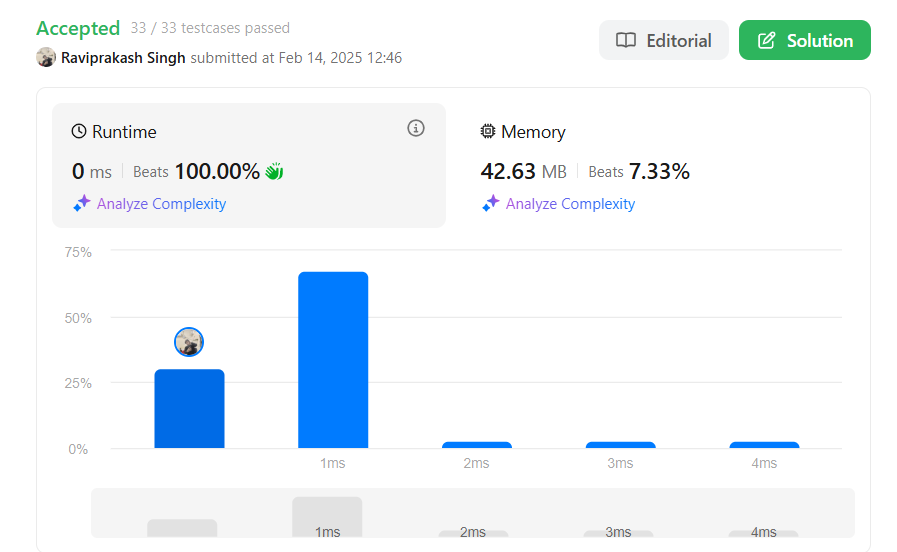
            level++;

        }

        return ans;

    }

}



[**199. Binary Tree Right Side View**](https://leetcode.com/problems/binary-tree-right-side-view/)

class Solution {

    public void view(TreeNode root , List<Integer> ans , int level){

       if(root==null) return;

       if(ans.size()==level) ans.add(root.val);

       view(root.right,ans,level+1);

       view(root.left,ans,level+1);

    }

    public List<Integer> rightSideView(TreeNode root) {

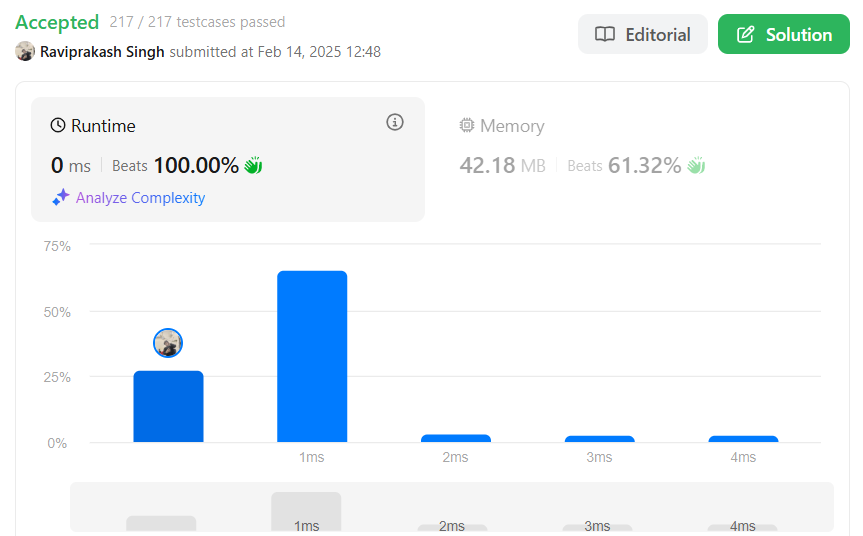
        List<Integer> ans = new ArrayList<>();

        view(root,ans,0);

        return ans;

    }

}



[**124. Binary Tree Maximum Path Sum**](https://leetcode.com/problems/binary-tree-maximum-path-sum/)

Sol: class Solution {

    int max = Integer.MIN\_VALUE;

    public int path(TreeNode root){

        if(root==null) return 0;

        int left= Math.max(0,path(root.left));

        int right =Math.max(0,path(root.right));

        max = Math.max(max,left+right+root.val);

        return root.val + Math.max(left,right);

    }

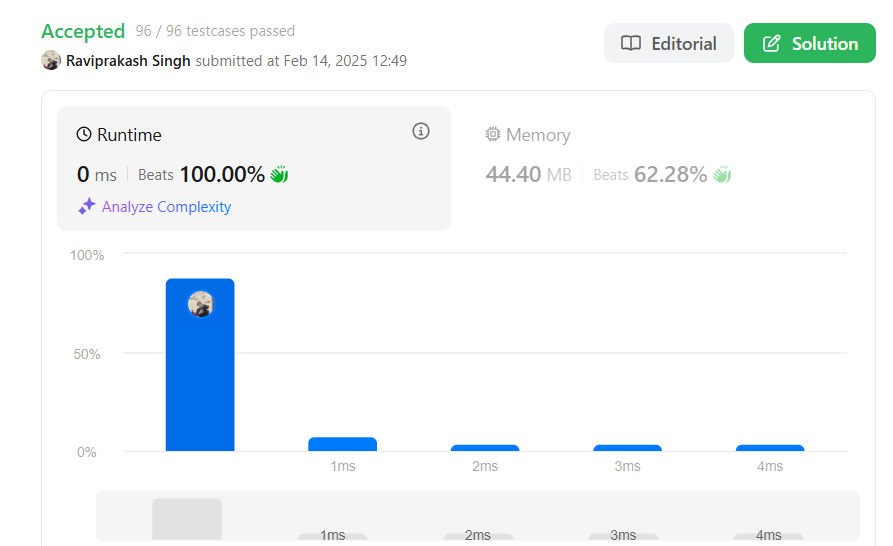
    public int maxPathSum(TreeNode root) {

        path(root);

        return max;

    }

}



[**106. Construct Binary Tree from Inorder and Postorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/)

Sol:

import java.util.HashMap;

class TreeNode {

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int x) { val = x; }

}

class Solution {

    public TreeNode buildTree(int[] inorder, int[] postorder) {

        HashMap<Integer, Integer> rec = new HashMap<>();

        for (int i = 0; i < inorder.length; i++) {

            rec.put(inorder[i], i);

        }

        return helper(inorder, postorder, 0, inorder.length - 1, 0, postorder.length - 1, rec);

    }

    private TreeNode helper(int[] inorder, int[] postorder,

                            int inStart, int inEnd,

                            int postStart, int postEnd,

                            HashMap<Integer, Integer> rec) {

        if (inStart > inEnd || postStart > postEnd) return null;

        int val = postorder[postEnd];

        TreeNode root = new TreeNode(val);

        int idx = rec.get(val);

        int leftSubtreeSize = idx - inStart;

        root.left = helper(inorder, postorder,

                            inStart, idx - 1,

                            postStart, postStart + leftSubtreeSize - 1,

                            rec);

        root.right = helper(inorder, postorder,

                             idx + 1, inEnd,

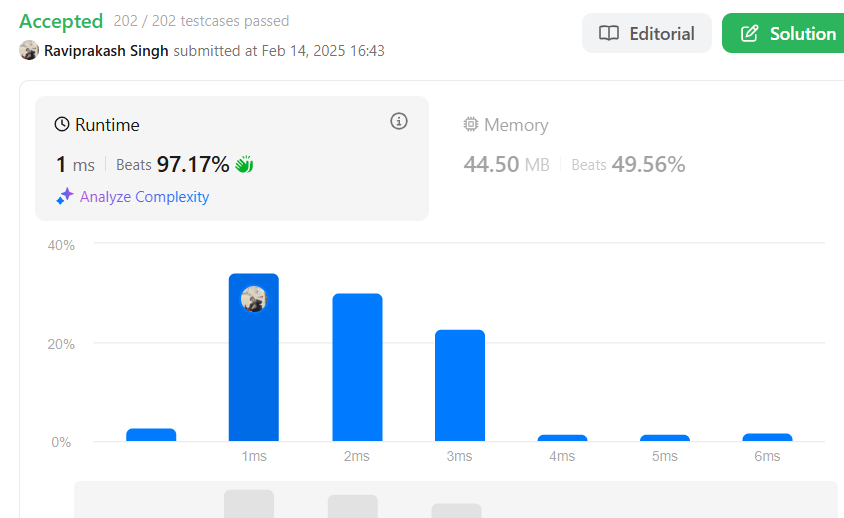
                             postStart + leftSubtreeSize, postEnd - 1,

                             rec);

        return root;

    }

}



[**513. Find Bottom Left Tree Value**](https://leetcode.com/problems/find-bottom-left-tree-value/)

Sol:

public class Solution {

public int findBottomLeftValue(TreeNode root) {

int last = 0;

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

q.add(root);

while (!q.isEmpty()) {

int count = q.size();

for (int i = 0; i < count; i++) {

TreeNode curr = q.poll();

if (i == 0)

last = curr.val; // last leftMost val

if (curr.left != null)

q.add(curr.left);

if (curr.right != null)

q.add(curr.right);

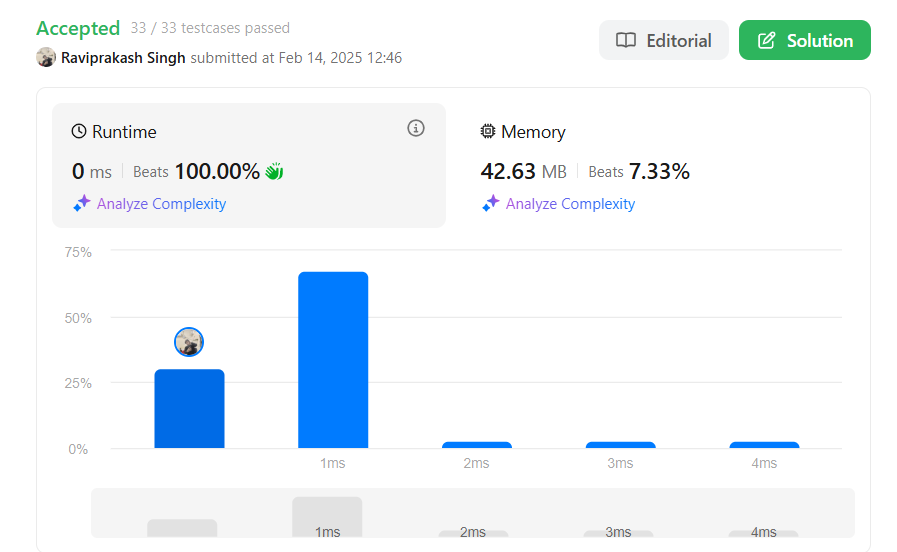
}

}

return last;

}

}



[**107. Binary Tree Level Order Traversal II**](https://leetcode.com/problems/binary-tree-level-order-traversal-ii/)

Sol:

class Solution {

    public List<List<Integer>> levelOrderBottom(TreeNode root) {

        LinkedList<List<Integer>> ans = new LinkedList<>();

        if(root==null)return ans;

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

        q.add(root);

        while(!q.isEmpty()){

          int size = q.size();

          LinkedList<Integer> l = new LinkedList<>();

          for(int i = 0 ; i<size ; i++){

            TreeNode x = q.poll();

            l.add(x.val);

            if(x.left!=null)q.add(x.left);

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

          }

          ans.addFirst(l);

        }

        return ans;

    }

}

