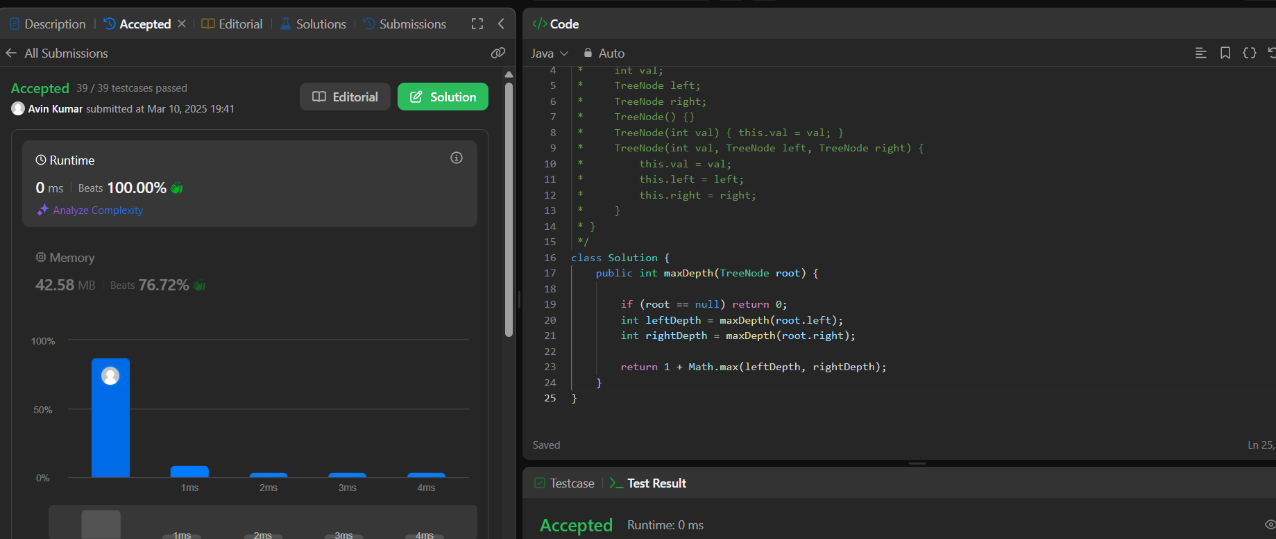
**Assignment 5**

**Student Name: Avin Kumar UID: 22BCS10600 Section/Group: IOT-614 B Semester: 6th**

**Subject Name: Advanced Programming Subject Code:** **22CSP-351**

**Lab-2**

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



**Code:**

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int leftDepth = maxDepth(root.left);

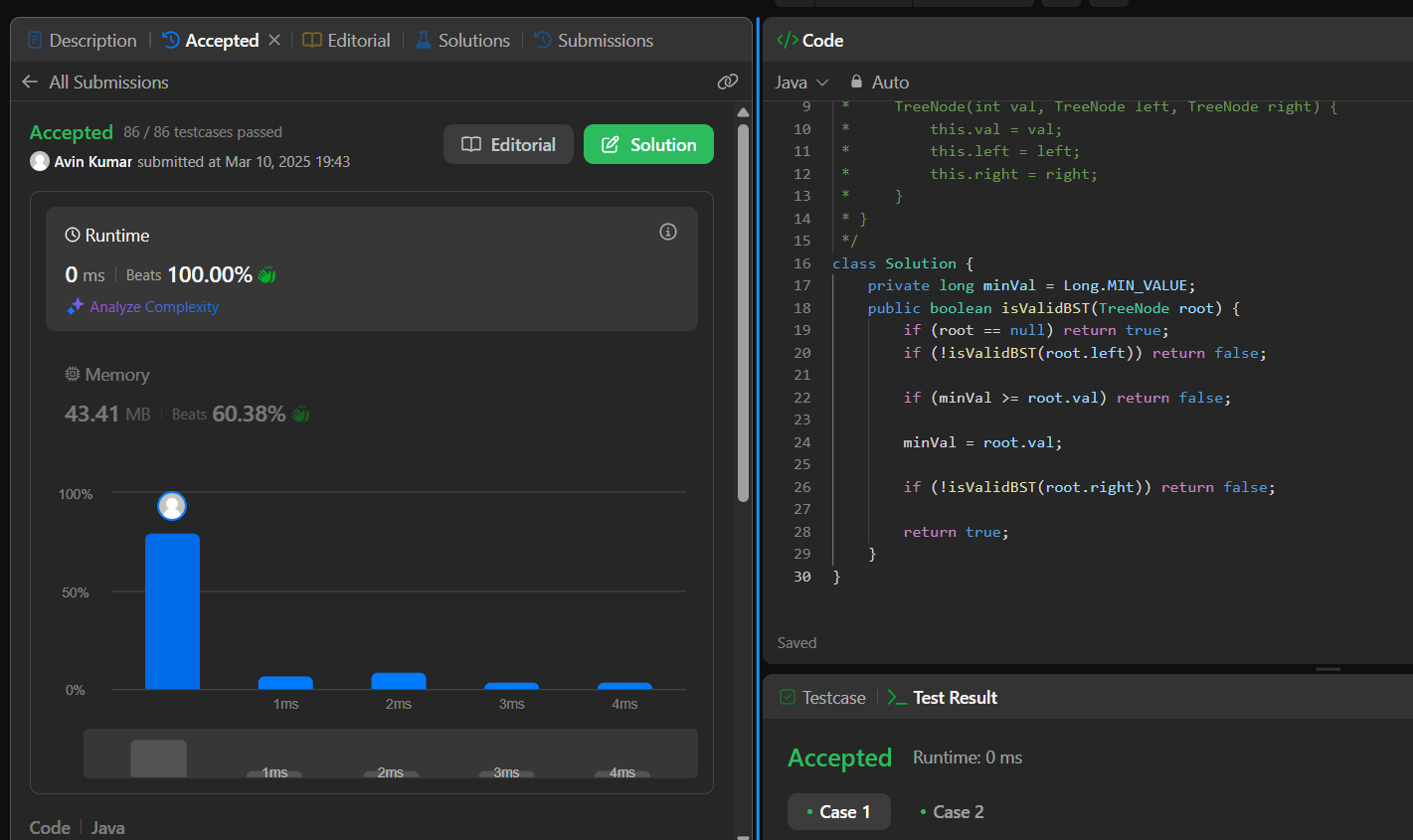
int rightDepth = maxDepth(root.right);

return 1 + Math.max(leftDepth, rightDepth);

}

}

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



**Code:**

class Solution {

private long minVal = Long.MIN\_VALUE;

public boolean isValidBST(TreeNode root) {

if (root == null) return true;

if (!isValidBST(root.left)) return false;

if (minVal >= root.val) return false;

minVal = root.val;

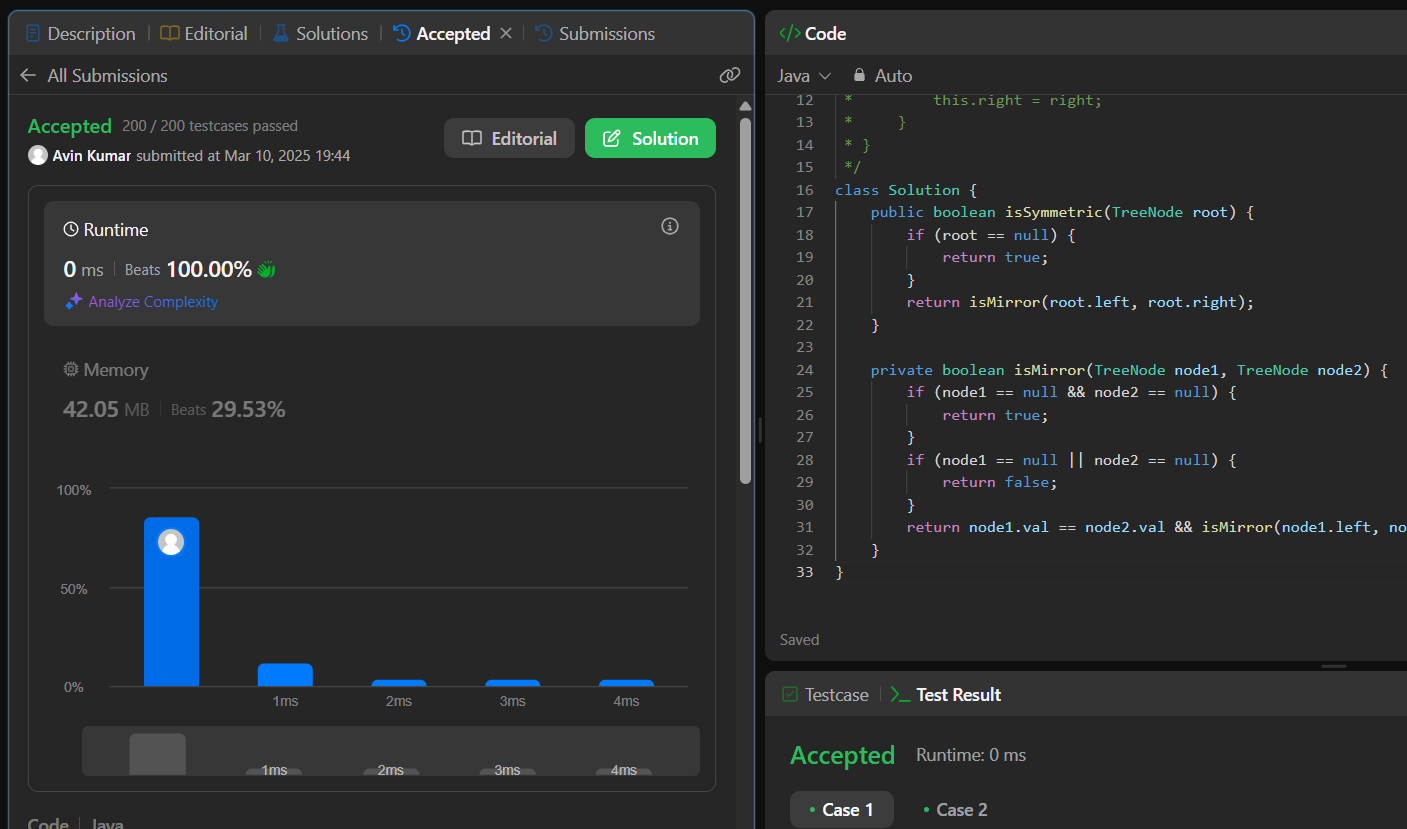
if (!isValidBST(root.right)) return false;

return true;

}

}

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



**Code:**

class Solution {

public boolean isSymmetric(TreeNode root) {

if (root == null) {

return true;

}

return isMirror(root.left, root.right);

}

private boolean isMirror(TreeNode node1, TreeNode node2) {

if (node1 == null && node2 == null) {

return true;

}

if (node1 == null || node2 == null) {

return false;

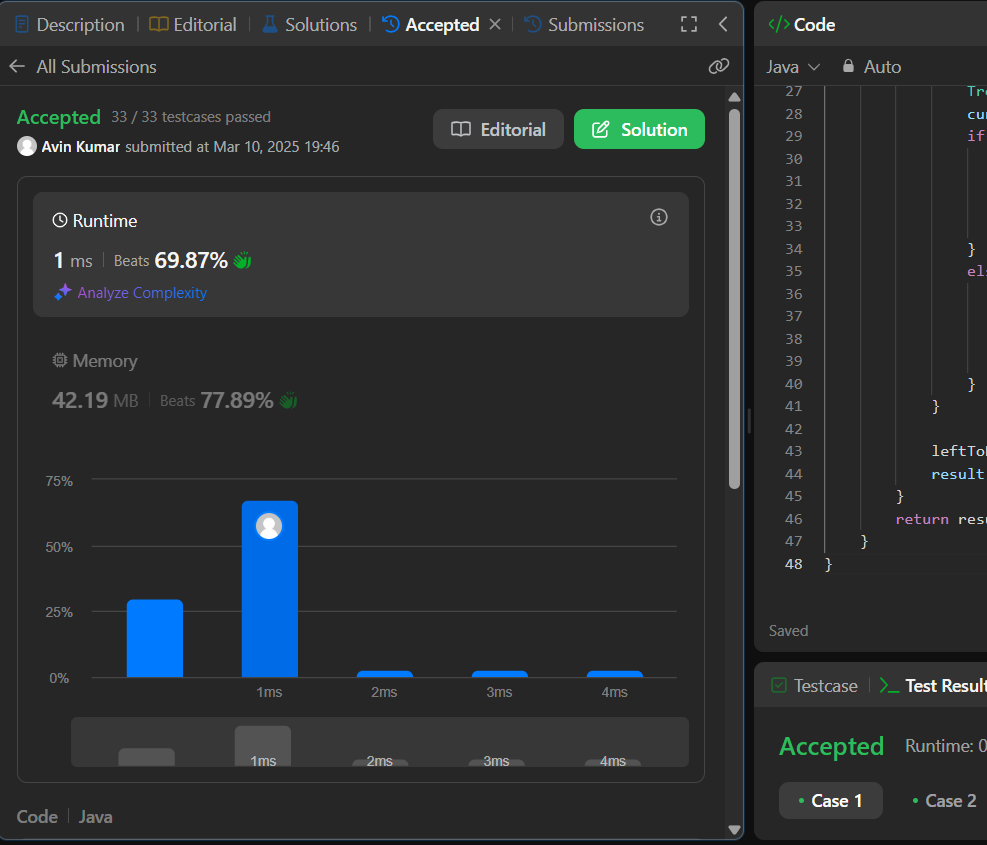
}

return node1.val == node2.val && isMirror(node1.left, node2.right) && isMirror(node1.right, node2.left);

}

}

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



**Code:**

class Solution {

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

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

ArrayDeque<TreeNode> dq = new ArrayDeque<>();

dq.offer(root);

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

boolean leftToRight = true;

while(!dq.isEmpty()){

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

for(int i = dq.size(); i > 0; i--){

TreeNode curr = (leftToRight)?dq.pollFirst():dq.pollLast();

currLevel.add(curr.val);

if(leftToRight){

if(curr.left != null)

dq.offerLast(curr.left);

if(curr.right != null)

dq.offerLast(curr.right);

}

else{

if(curr.right != null)

dq.offerFirst(curr.right);

if(curr.left != null)

dq.offerFirst(curr.left);

}

}

leftToRight = !leftToRight;

result.add(currLevel);

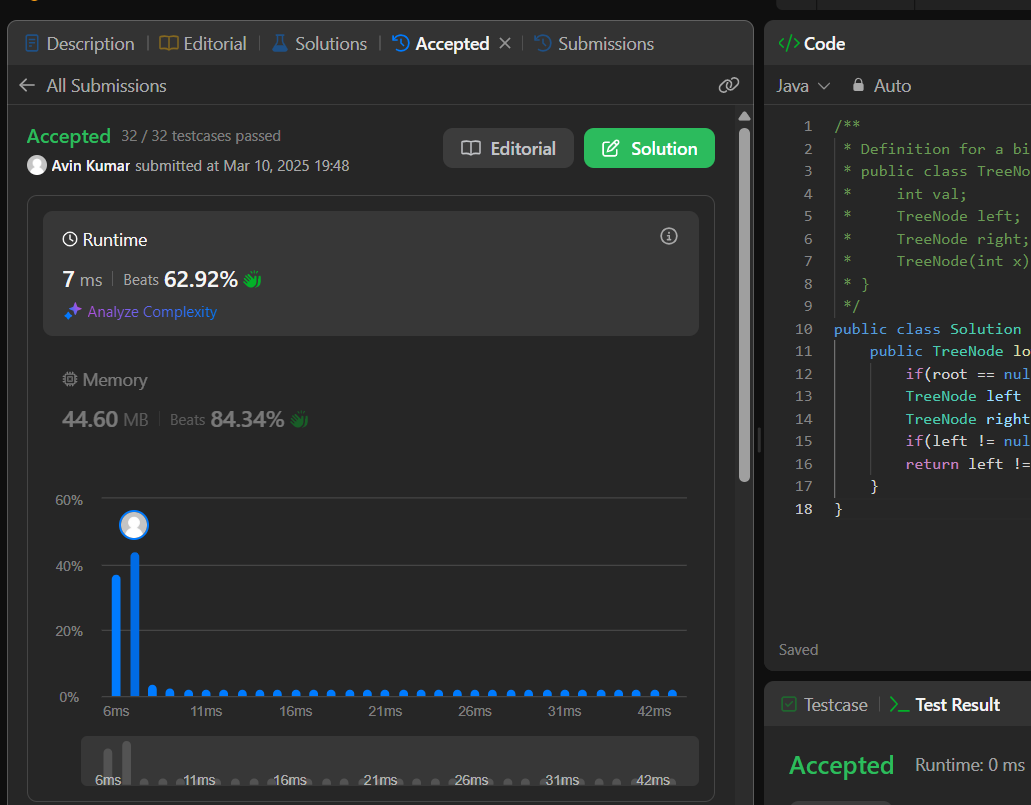
}

return result;

}

}

[**236. Lowest Common Ancestor of a Binary Tree**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)



**Code:**

public class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(root == null || root == p || root == q) return root;

TreeNode left = lowestCommonAncestor(root.left, p, q);

TreeNode right = lowestCommonAncestor(root.right, p, q);

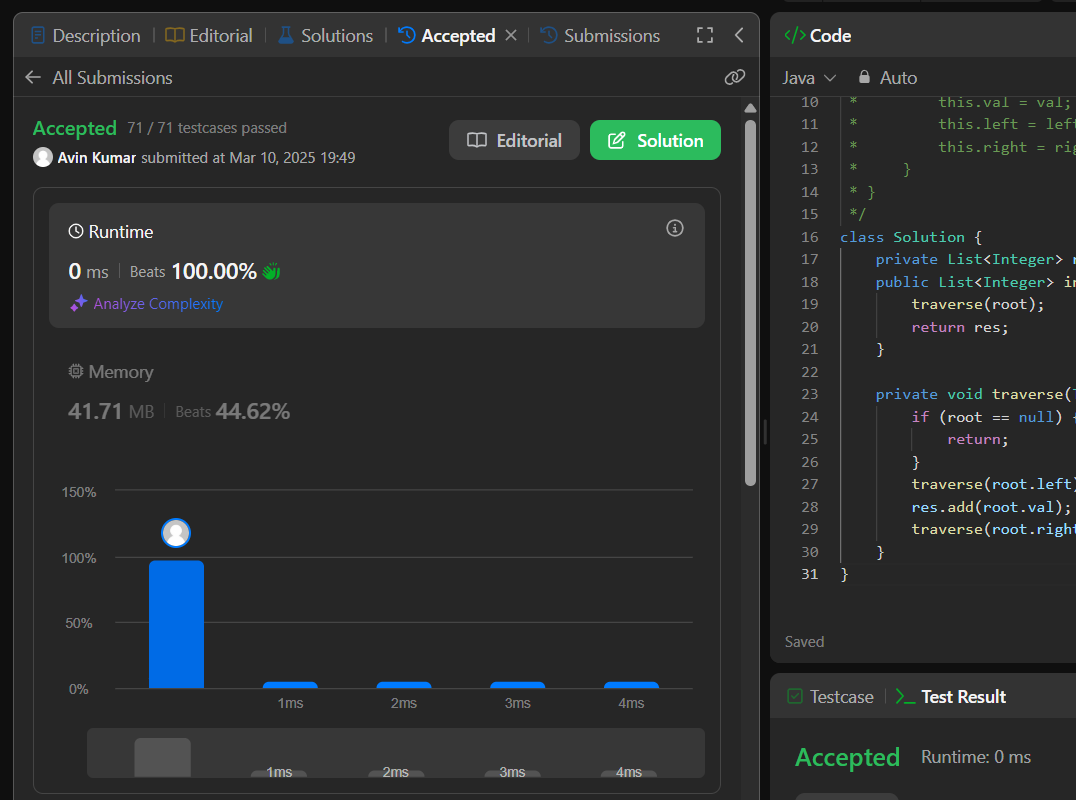
if(left != null && right != null) return root;

return left != null ? left : right;

}

}

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



**Code:**

class Solution {

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

public List<Integer> inorderTraversal(TreeNode root) {

traverse(root);

return res;

}

private void traverse(TreeNode root) {

if (root == null) {

return;

}

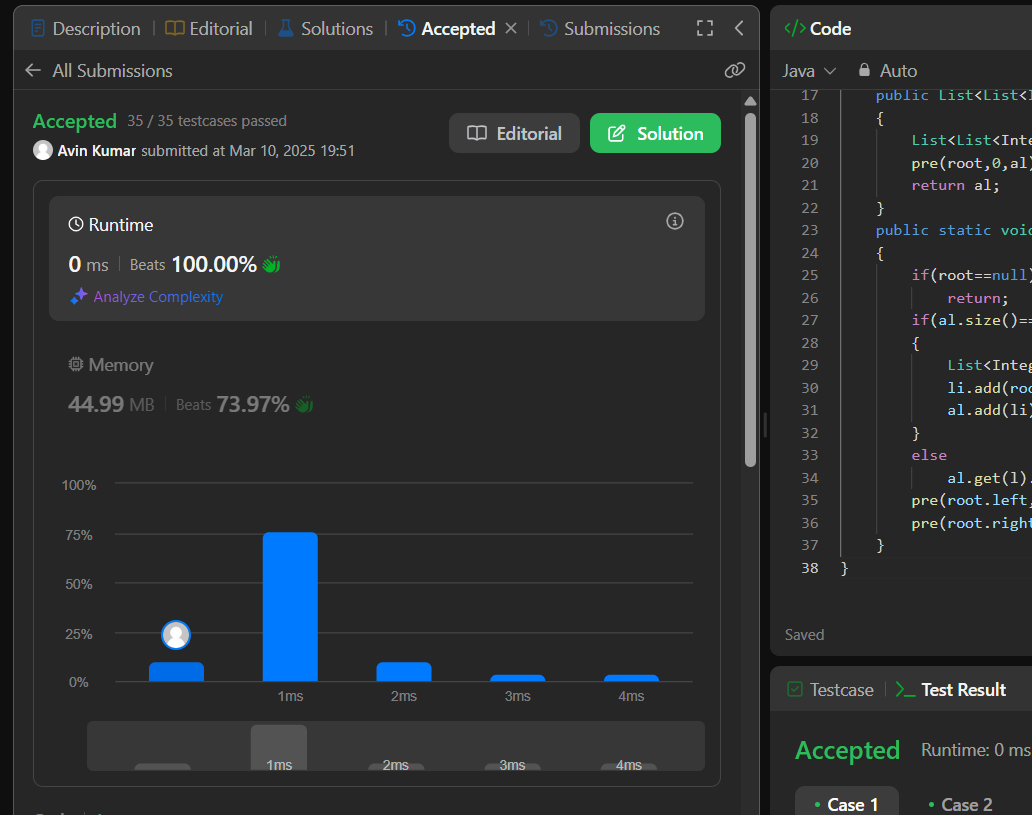
traverse(root.left);

res.add(root.val);

traverse(root.right);

}}

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

****

**Code:**

class Solution {

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

{

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

pre(root,0,al);

return al;

}

public static void pre(TreeNode root,int l,List<List<Integer>>al)

{

if(root==null)

return;

if(al.size()==l)

{

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

li.add(root.val);

al.add(li);

}

else

al.get(l).add(root.val);

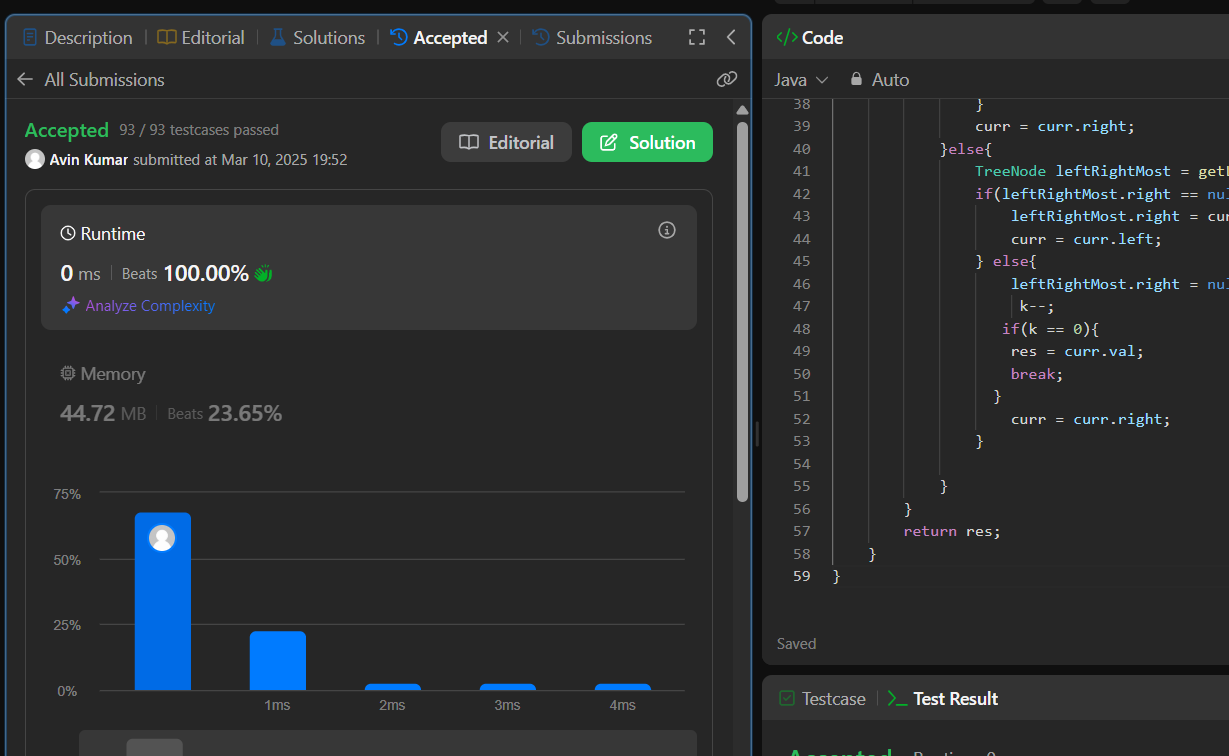
pre(root.left,l+1,al);

pre(root.right,l+1,al);

}

}

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



**Code:**

class Solution {

public int kthSmallest(TreeNode root, int k) {

return morris(root, k);

}

public TreeNode getLeftRightMost(TreeNode node, TreeNode parent){

while(node.right != null && node.right != parent){

node = node.right;

}

return node;

}

public int morris(TreeNode root, int k){

TreeNode curr = root;

int res = -1;

while(curr != null){

TreeNode left = curr.left;

if(left == null){

k--;

if(k == 0){

res = curr.val;

break;

}

curr = curr.right;

}else{

TreeNode leftRightMost = getLeftRightMost(curr.left, curr);

if(leftRightMost.right == null){

leftRightMost.right = curr;

curr = curr.left;

} else{

leftRightMost.right = null;

k--;

if(k == 0){

res = curr.val;

break;

}

curr = curr.right;

}

}

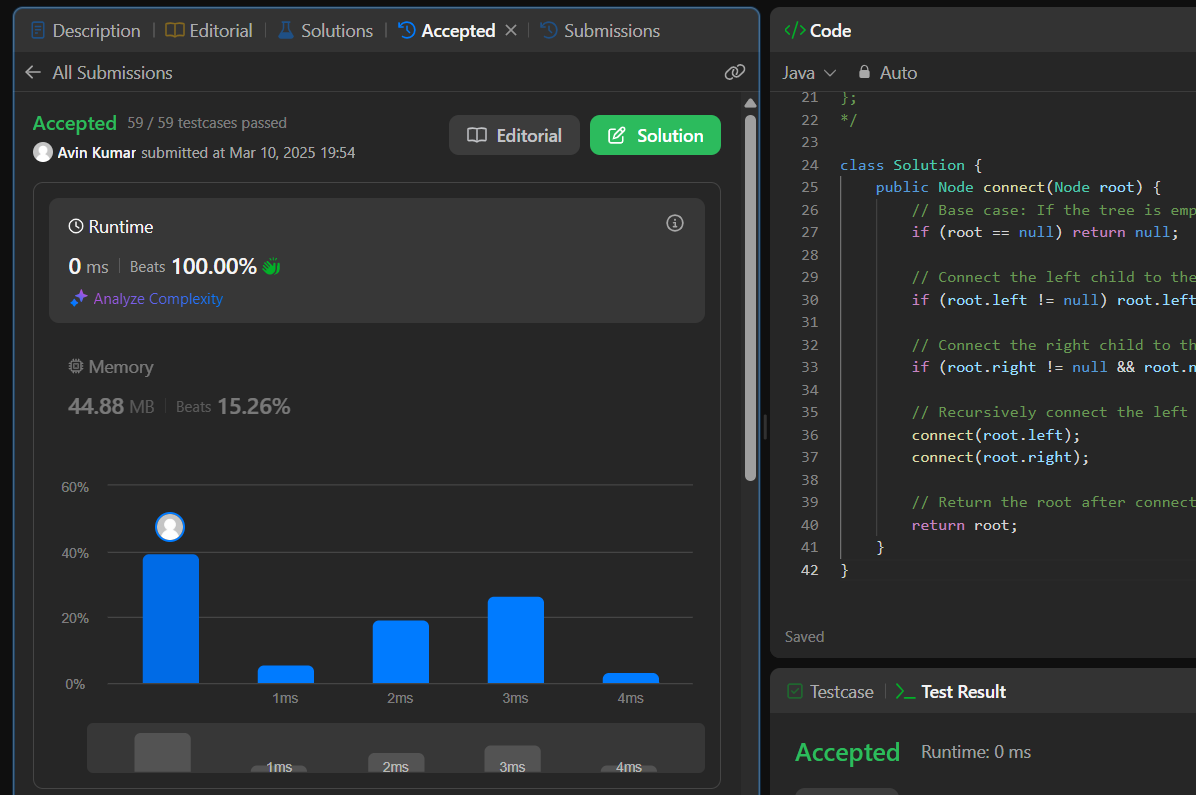
}

return res;

}

}

[**116. Populating Next Right Pointers in Each Node**](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/)



**Code:**

class Solution {

public Node connect(Node root) {

if (root == null) return null;

if (root.left != null) root.left.next = root.right;

if (root.right != null && root.next != null) root.right.next = root.next.left;

connect(root.left);

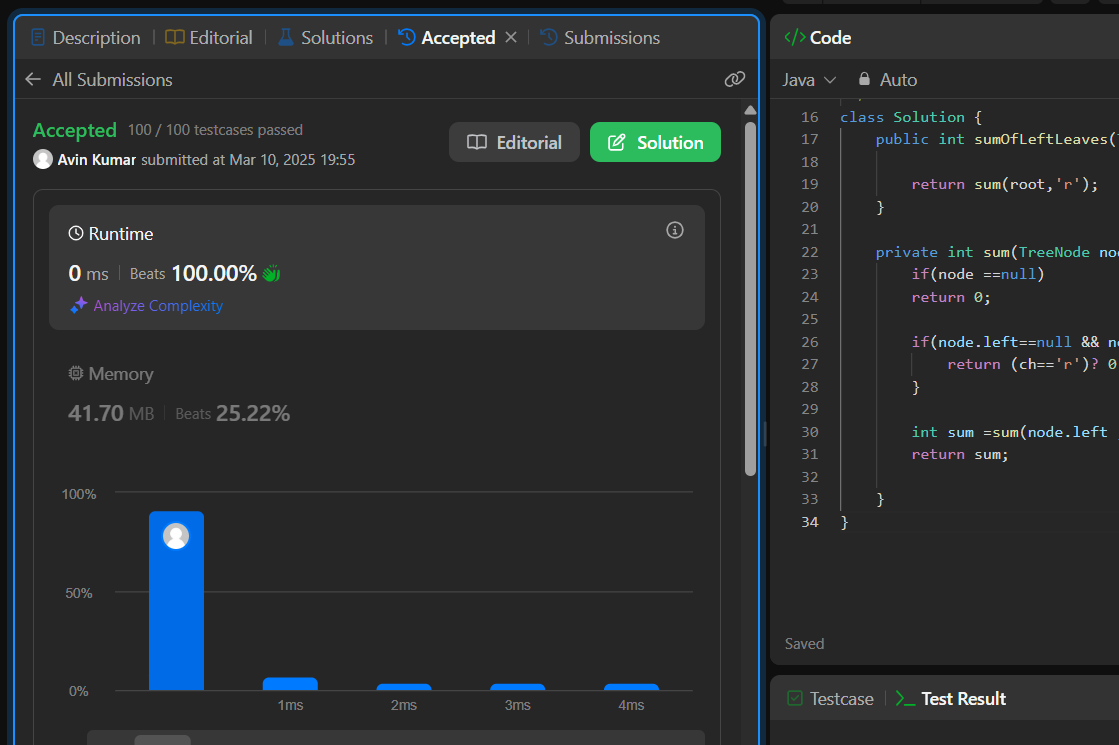
connect(root.right);

return root;

}

}

[**404. Sum of Left Leaves**](https://leetcode.com/problems/sum-of-left-leaves/)



**Code:**

class Solution {

public int sumOfLeftLeaves(TreeNode root) {

return sum(root,'r');

}

private int sum(TreeNode node , char ch){

if(node ==null)

return 0;

if(node.left==null && node.right==null){

return (ch=='r')? 0: node.val;

}

int sum =sum(node.left ,'l') + sum(node.right ,'r');

return sum;

}}