**AP Assignment-3**

**Q Binary Tree Inorder Traversal**

Code:

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

Stack<TreeNode> stack = new Stack<TreeNode>();

TreeNode cur = root;

while(cur!= null || !stack.empty()){

while(cur != null){

stack.push(cur);

cur = cur.left;

}

cur = stack.pop();

list.add(cur.val);

cur = cur.right;

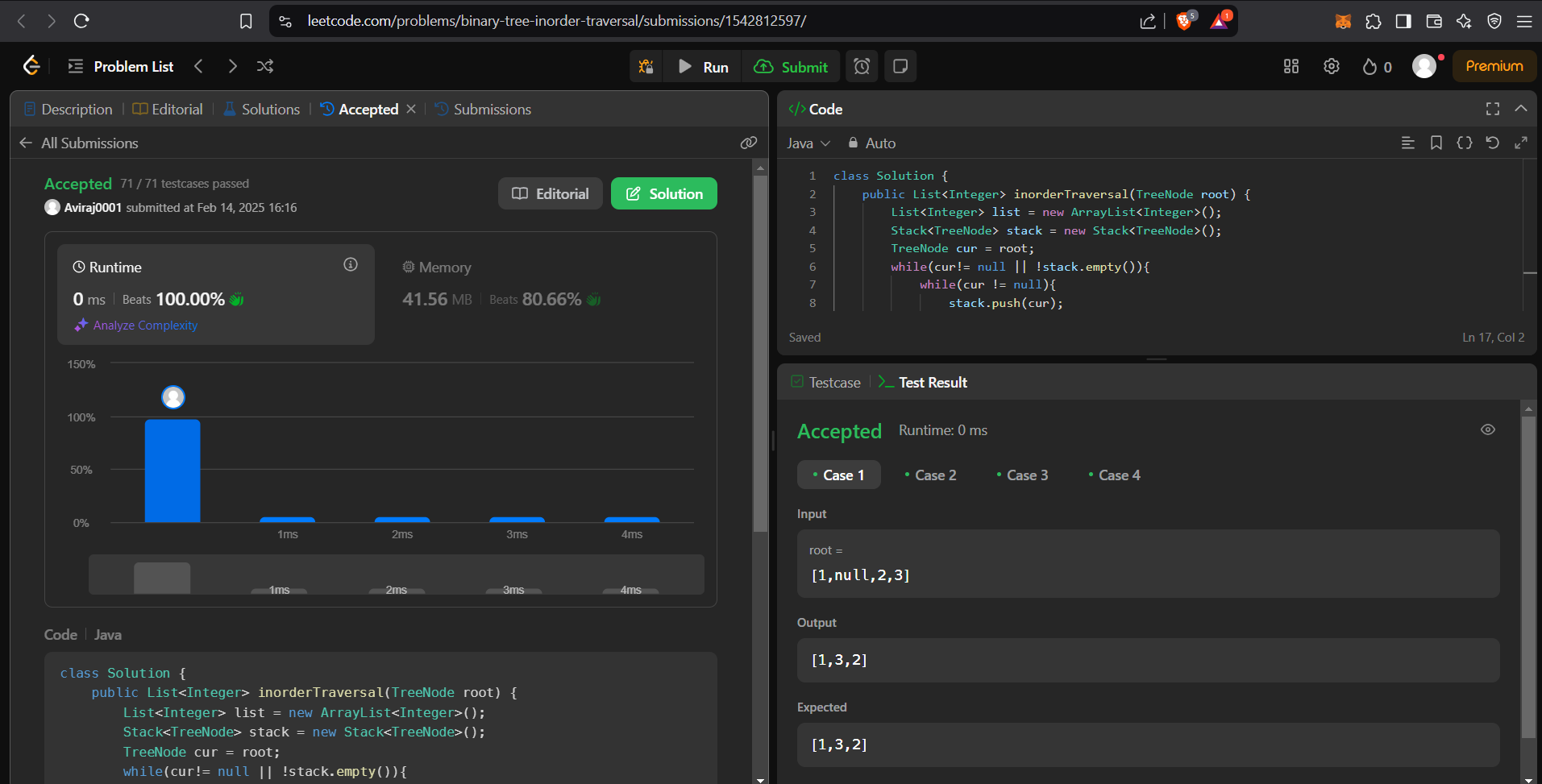
}

return list;

}

}

Output:



**Q Symmetric Tree**

Code:

class Solution {

    public boolean isSymmetric(TreeNode root) {

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

    }

    public boolean check(TreeNode p,TreeNode q){

        if(p == null && q == null)return true;

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

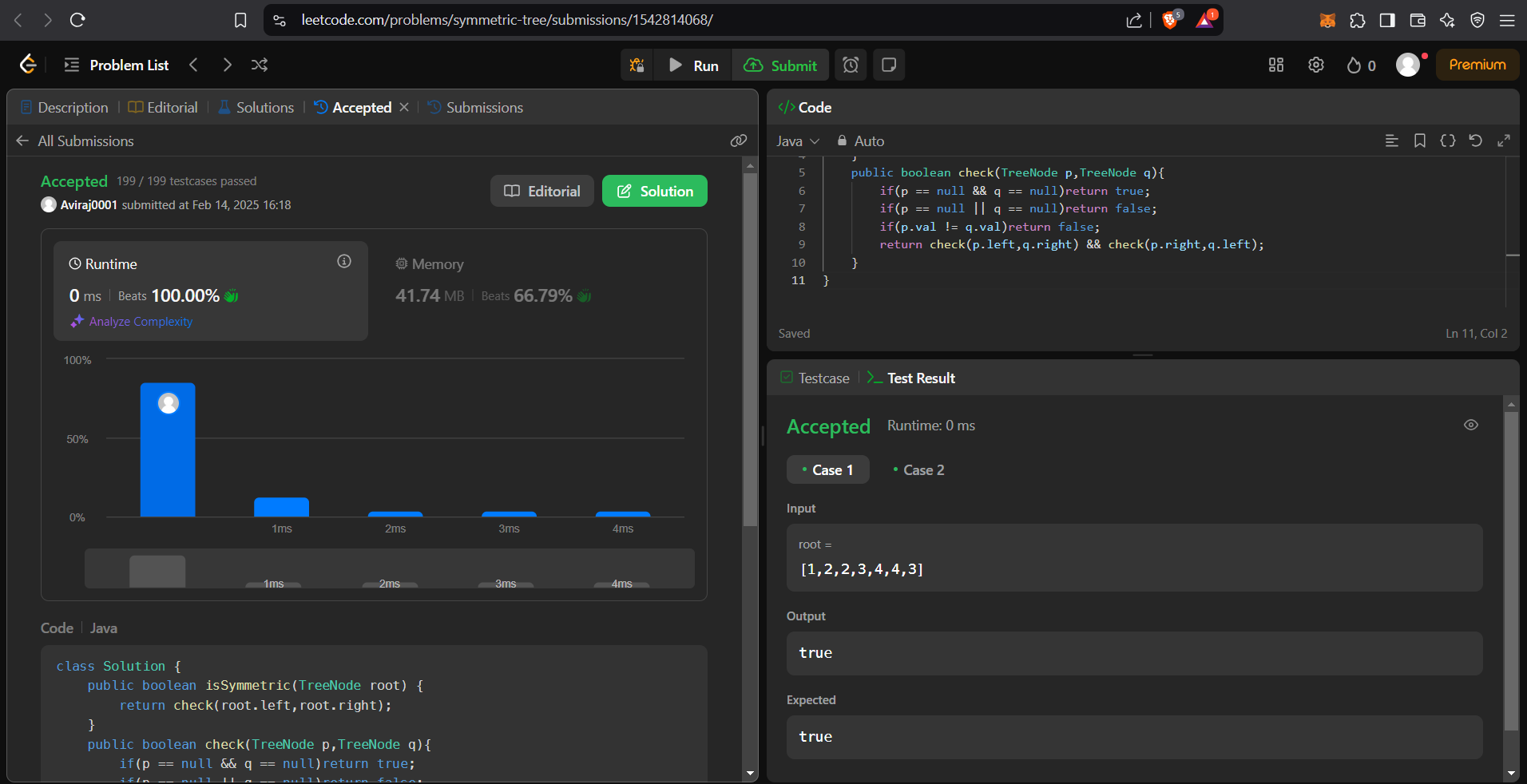
        if(p.val != q.val)return false;

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

    }

}

Output:



**Q Maximum Depth of Binary Tree**

Code:

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int ld = maxDepth(root.left);

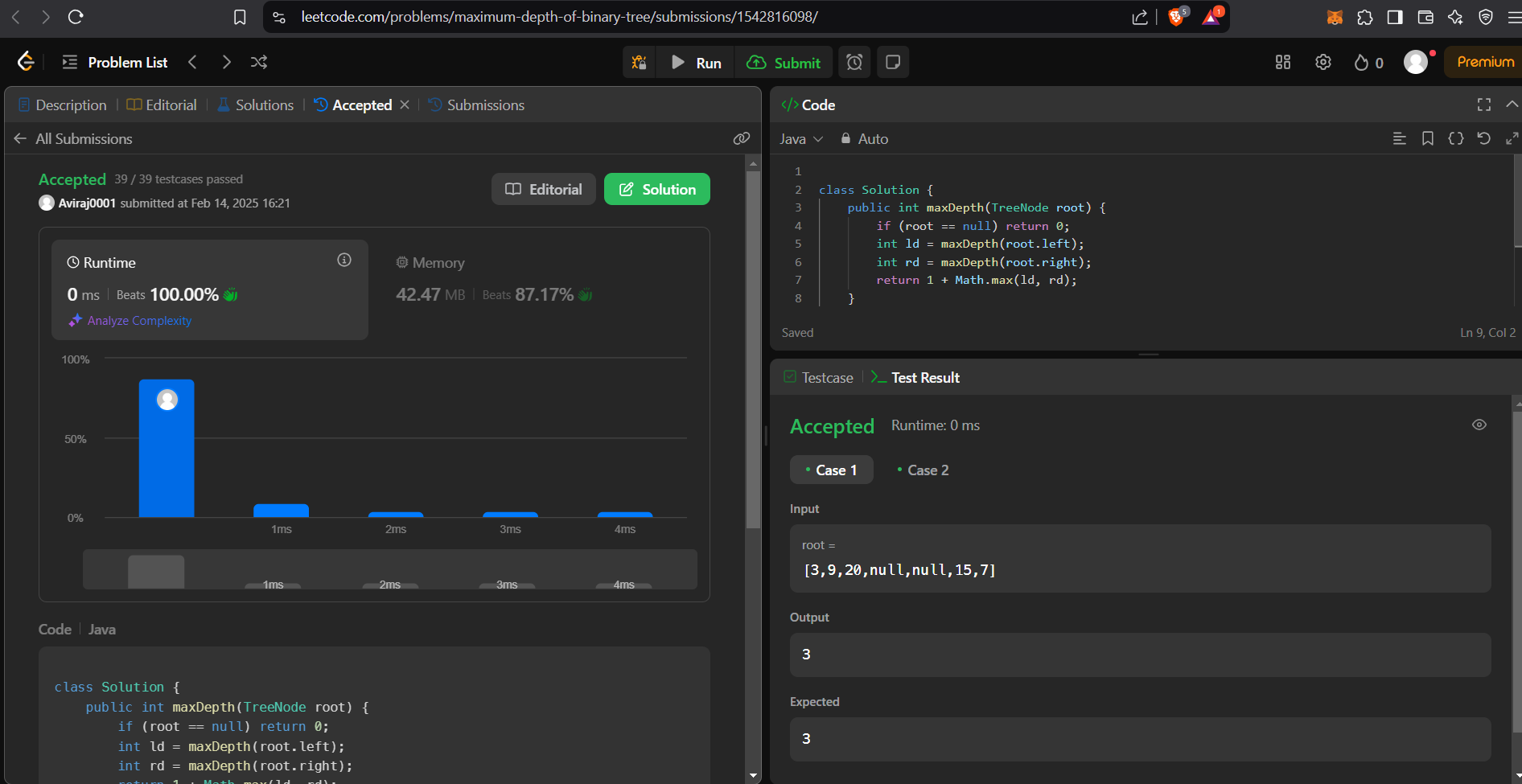
int rd = maxDepth(root.right);

return 1 + Math.max(ld, rd);

}

}

Output:



**Q Validate Binary Search Tree**

Code:

class Solution {

public boolean isValidBST(TreeNode root) {

return valid(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

}

private boolean valid(TreeNode node, long minimum, long maximum) {

if (node == null) return true;

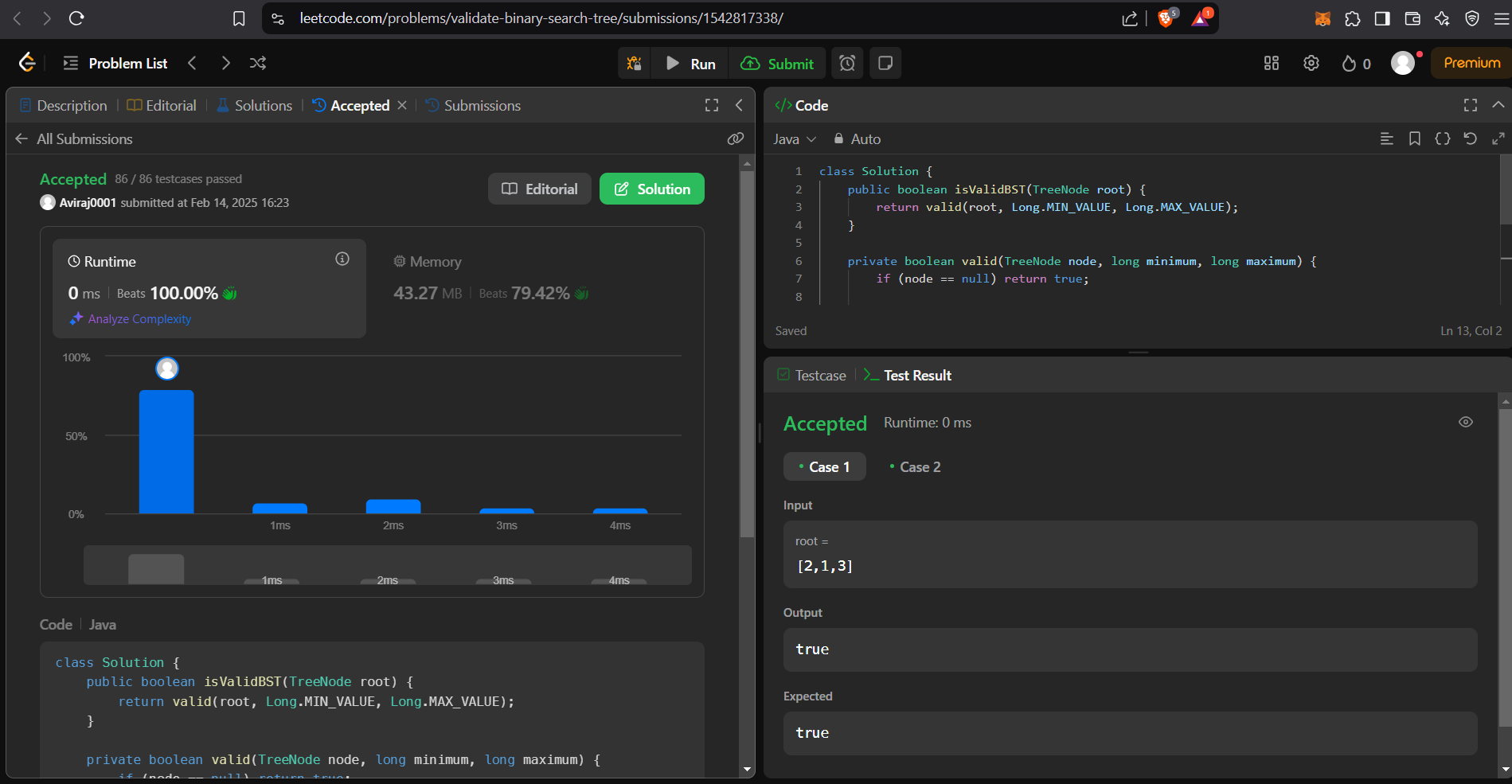
if (!(node.val > minimum && node.val < maximum)) return false;

return valid(node.left, minimum, node.val) && valid(node.right, node.val, maximum);

}

}

Output:



**Q Kth Smallest Element in a BST**

Code:

class Solution {

int count=0;

public int kthSmallest(TreeNode root, int k) {

if(root == null){

return -1;

}

int res = kthSmallest(root.left,k);

if(res !=-1){

return res;

}

count++;

if(count==k){

return root.val;

}

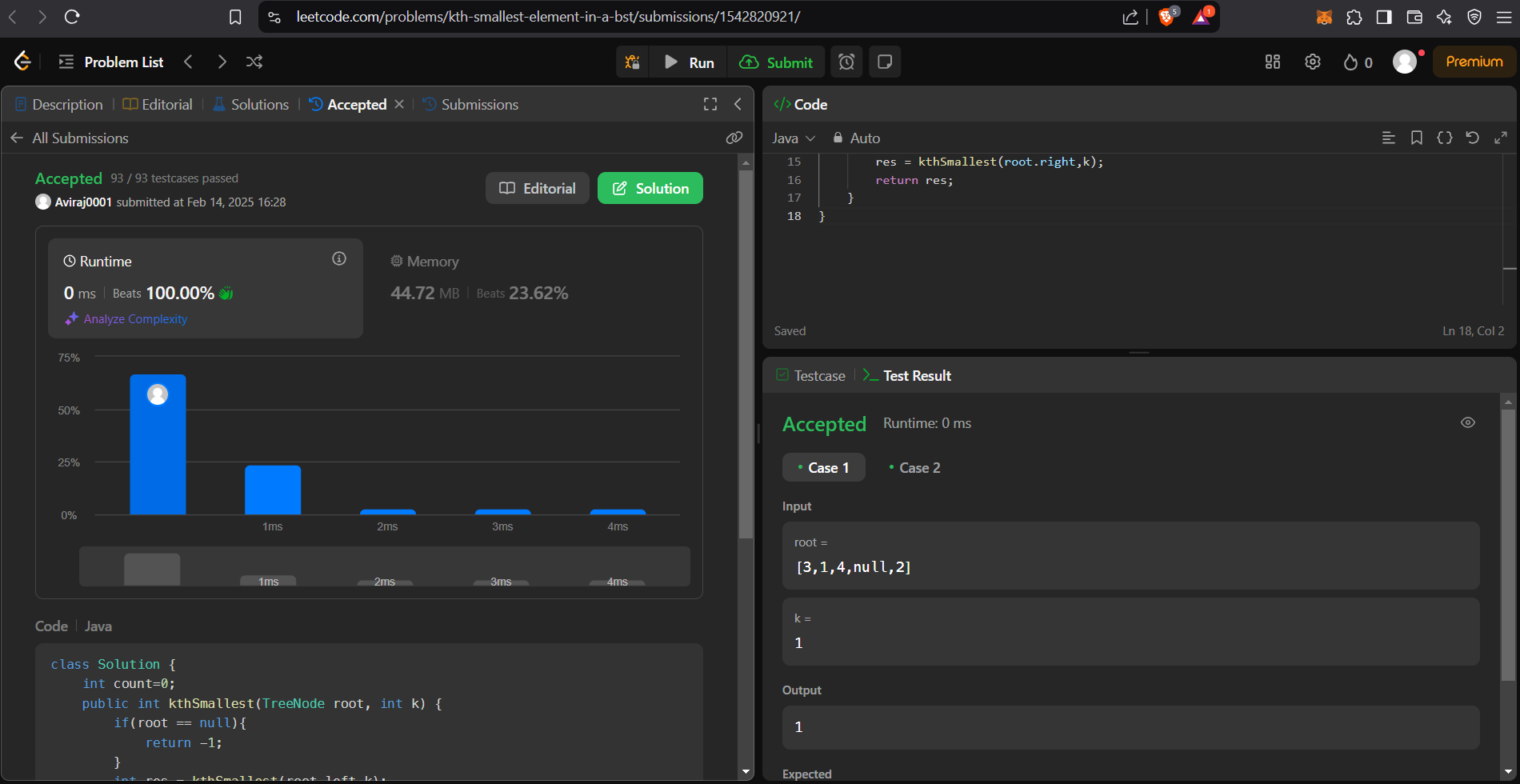
res = kthSmallest(root.right,k);

return res;

}

}

Output:



**Q Binary Tree Level Order Traversal**

Code:

class Solution {

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

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

if(root == null) return res;

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

queue.add(root);

while(!queue.isEmpty()){

int cnt = queue.size();

List<Integer> level = new ArrayList<>(cnt);

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

TreeNode node = queue.poll();

level.add(node.val);

if(node.left != null)

queue.add(node.left);

if(node.right != null)

queue.add(node.right);

}

res.add(level);

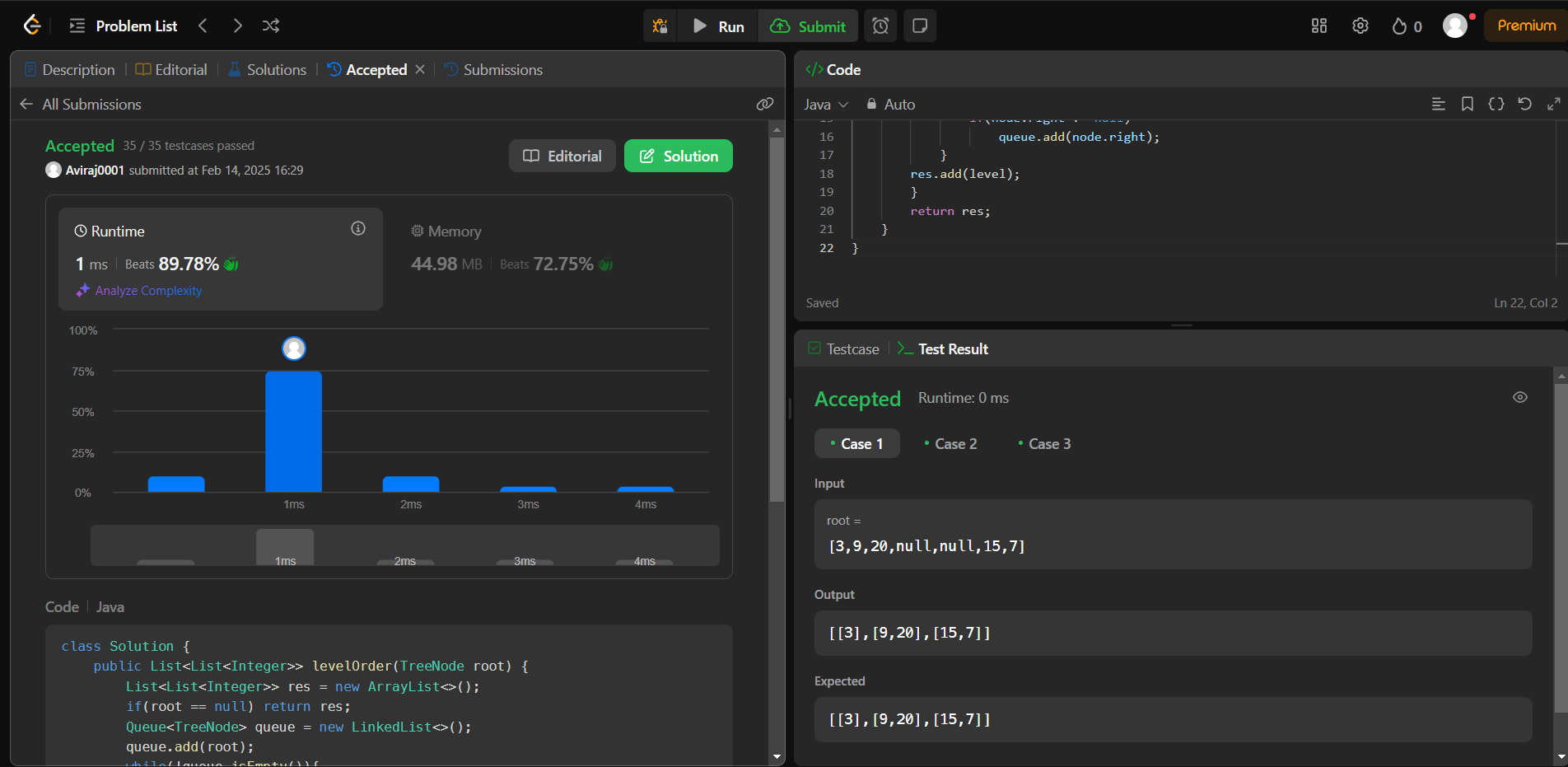
}

return res;

}

}

Output:



**Q Binary Tree Level Order Traversal II**

Code:

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

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

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

if(root==null) return ans;

Deque<TreeNode>q =new ArrayDeque();

q.offer(root);

while(!q.isEmpty())

{

List <Integer> l1=new ArrayList();

int size=q.size();

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

TreeNode curr=q.poll();

l1.add(curr.val);

if(curr.left!=null)q.offer(curr.left);

if(curr.right!=null)q.offer(curr.right);

}

ans.add(l1);

}

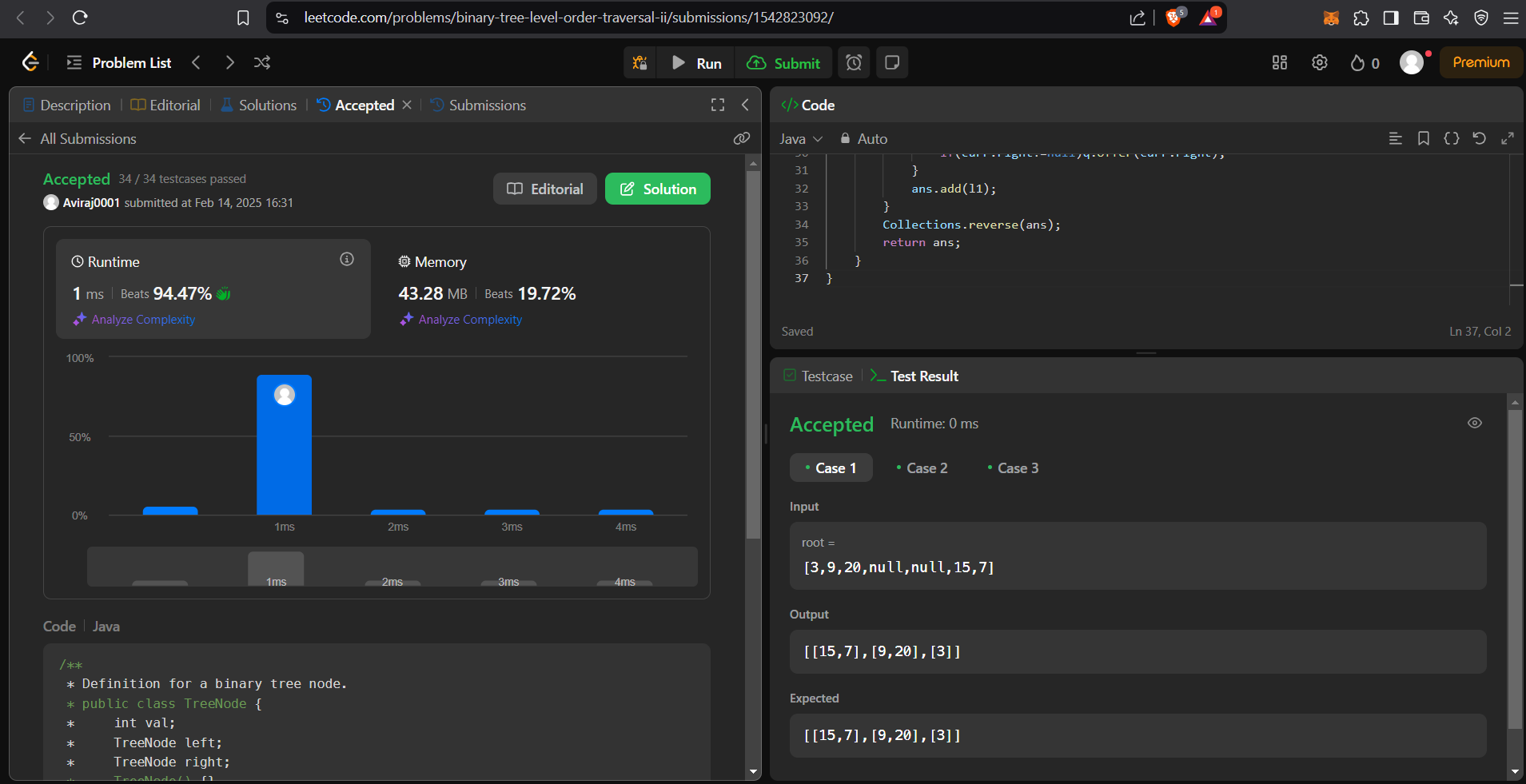
Collections.reverse(ans);

return ans;

}

}

Output:



**Q Binary Tree Zigzag Level Order Traversal**

Code:

class Solution {

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

{

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

travel(root, sol, 0);

return sol;

}

private void travel(TreeNode curr, List<List<Integer>> sol, int level)

{

if(curr == null) return;

if(sol.size() <= level)

{

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

sol.add(newLevel);

}

List<Integer> collection = sol.get(level);

if(level % 2 == 0) collection.add(curr.val);

else collection.add(0, curr.val);

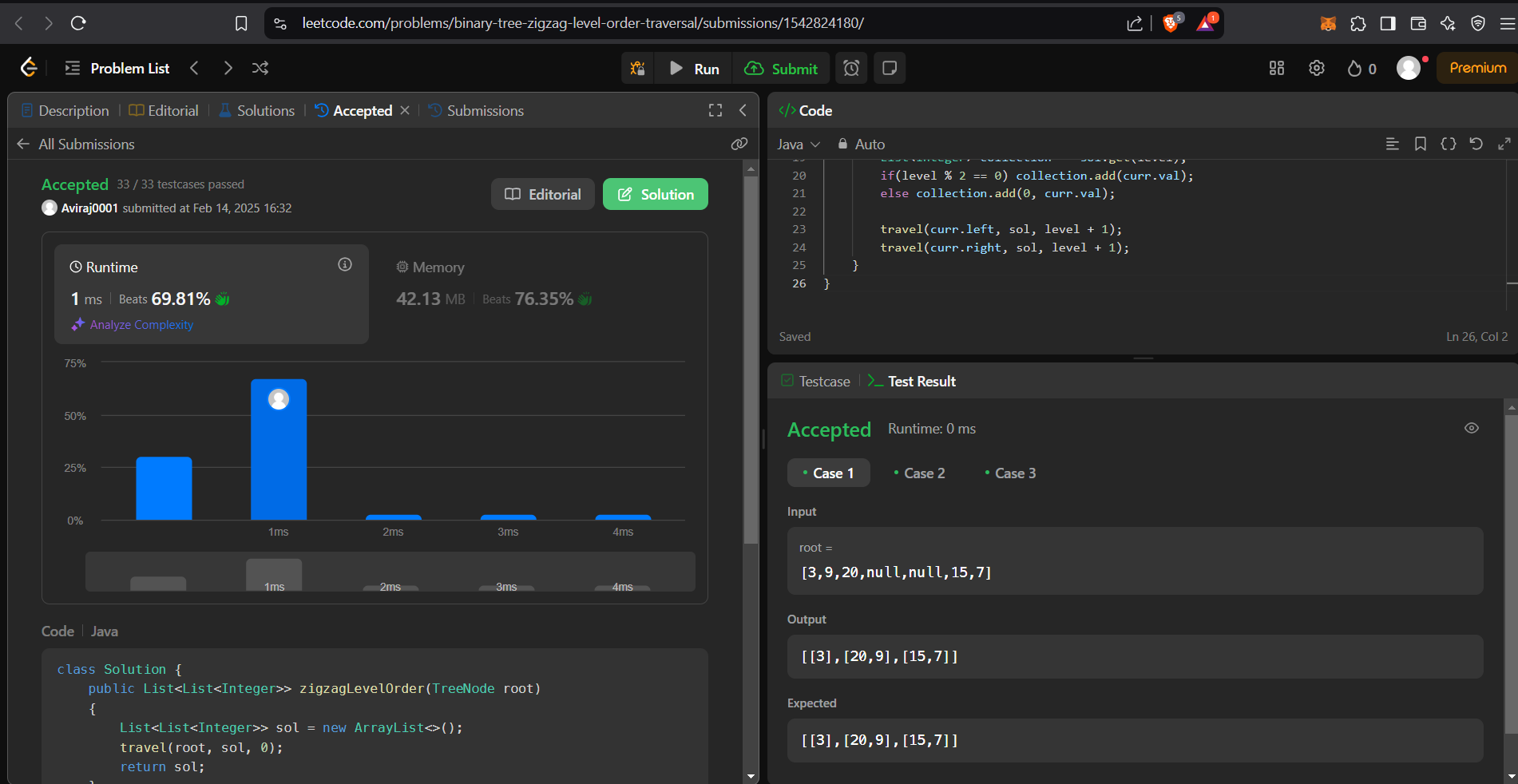
travel(curr.left, sol, level + 1);

travel(curr.right, sol, level + 1);

}

}

Output:



**Q Binary Tree Right Side View**

Code:

// If BFS TC O(N) && SC O(N/2) in worst case

// Recursive TC O(N) && SC O(H) height of tree

class Solution {

public List<Integer> rightSideView(TreeNode root) {

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

helper(root,0,list);

return list;

}

private void helper(TreeNode root,int level,List<Integer> list){

if(root==null){

return ;

}

// we will only add elemnt if DS(list) size== level.

// so we need to maintain level here

if(level==list.size()){

list.add(root.val);

}

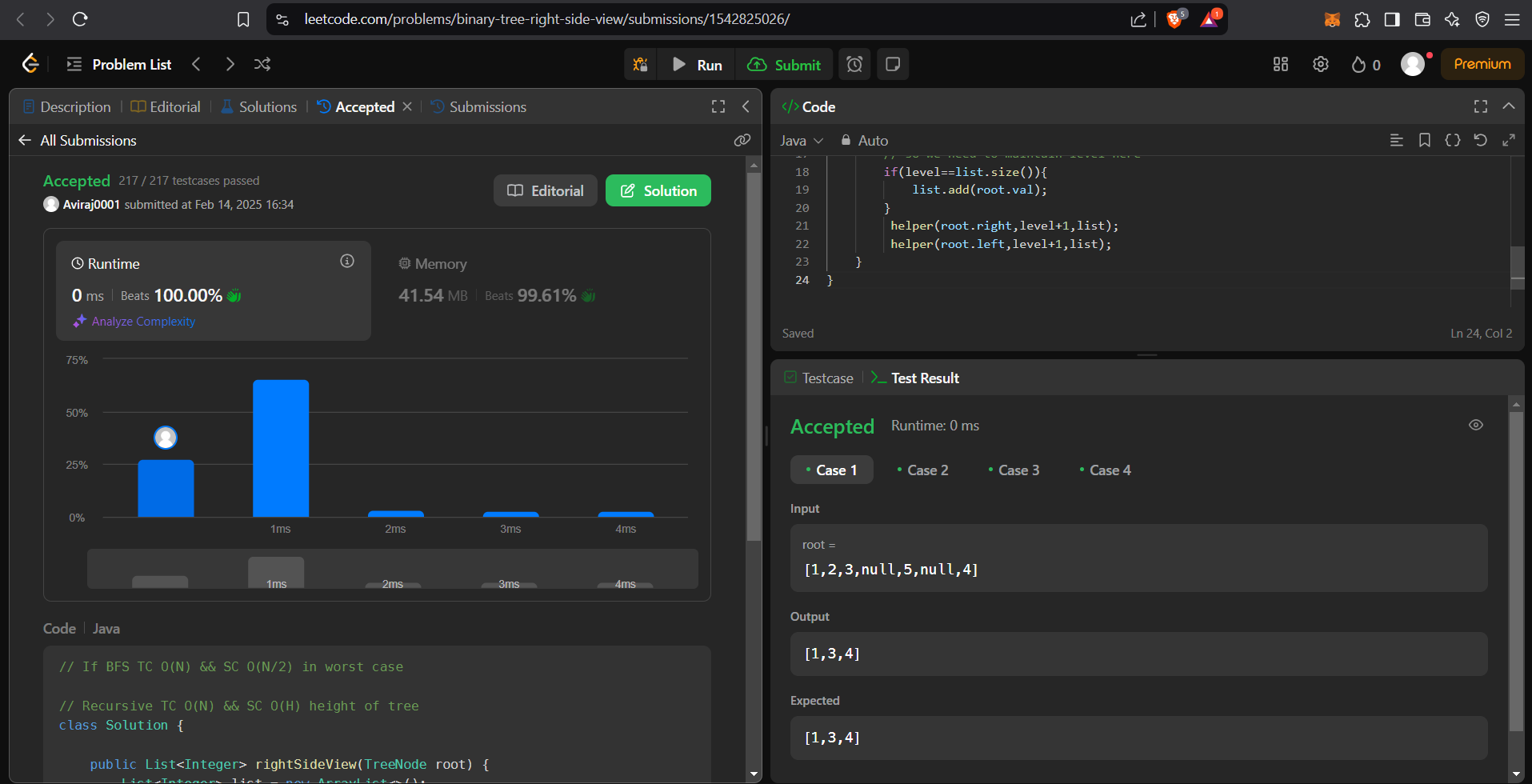
helper(root.right,level+1,list);

helper(root.left,level+1,list);

}

}

Output:



**Q Construct Binary Tree from Inorder and Postorder Traversal**

Code:

class Solution {

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

int r=inorder.length-1;

return build(inorder,postorder,0,r,0,r);

}

public TreeNode build(int a[],int b[],int l,int r,int p,int q)

{

if(l>r)

return null;

int val=b[q];

TreeNode node=new TreeNode(val);

int k=-1;

for(int i=l;i<=r;i++)

{if(a[i]==val)

{k=i;break;}}

node.left=build(a,b,l,k-1,p,p+k-l-1);

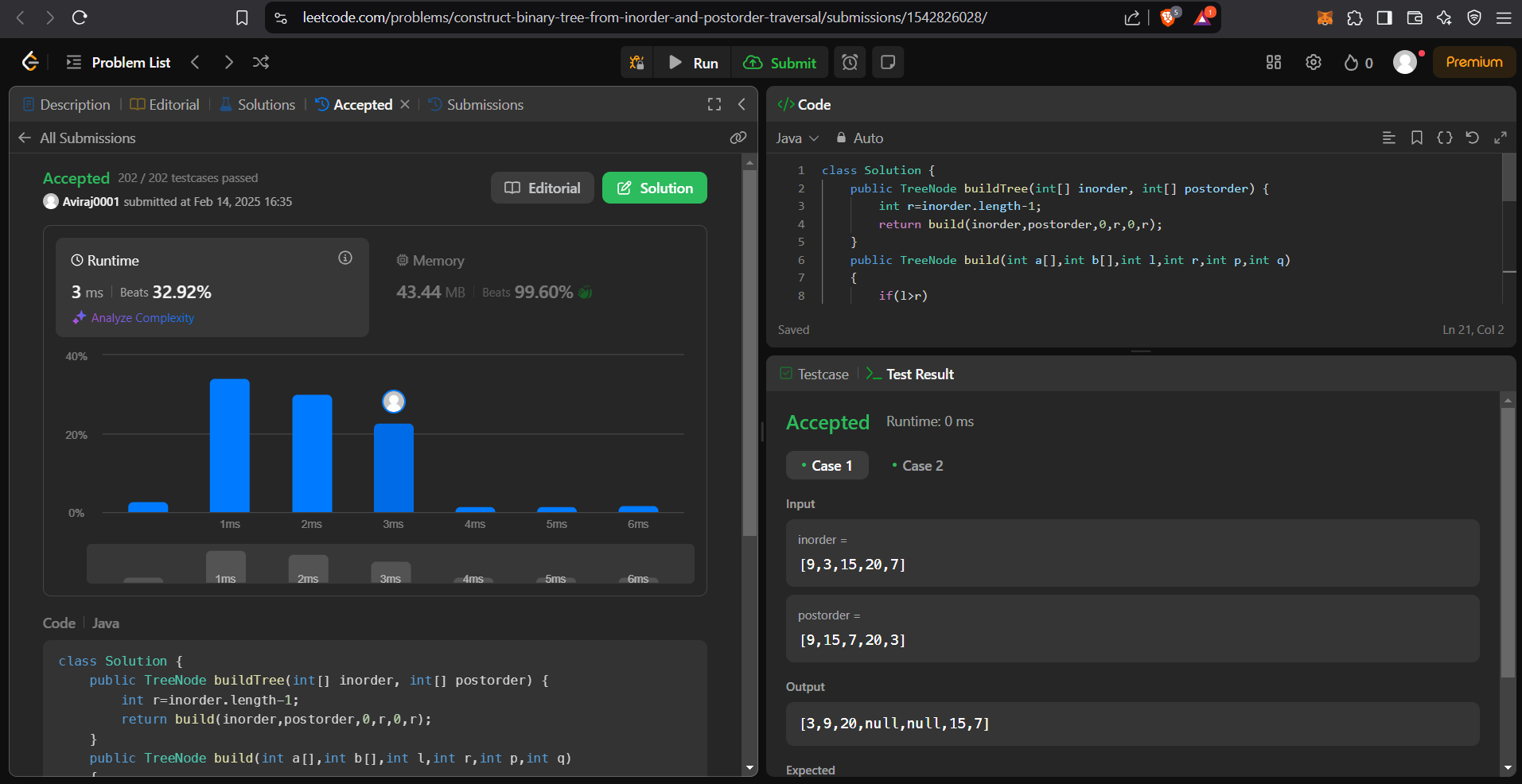
node.right=build(a,b,k+1,r,p+k-l,q-1);

return node;

}

}

Output:



**Q Find Bottom Left Tree Value**

Code:

class Solution {

private int maxDepth = -1;

private int value;

public int findBottomLeftValue(TreeNode root) {

dfs(root, 0);

return value;

}

private void dfs(TreeNode node, int depth) {

if (node == null) return;

if (depth > maxDepth) {

value = node.val;

maxDepth = depth;

}

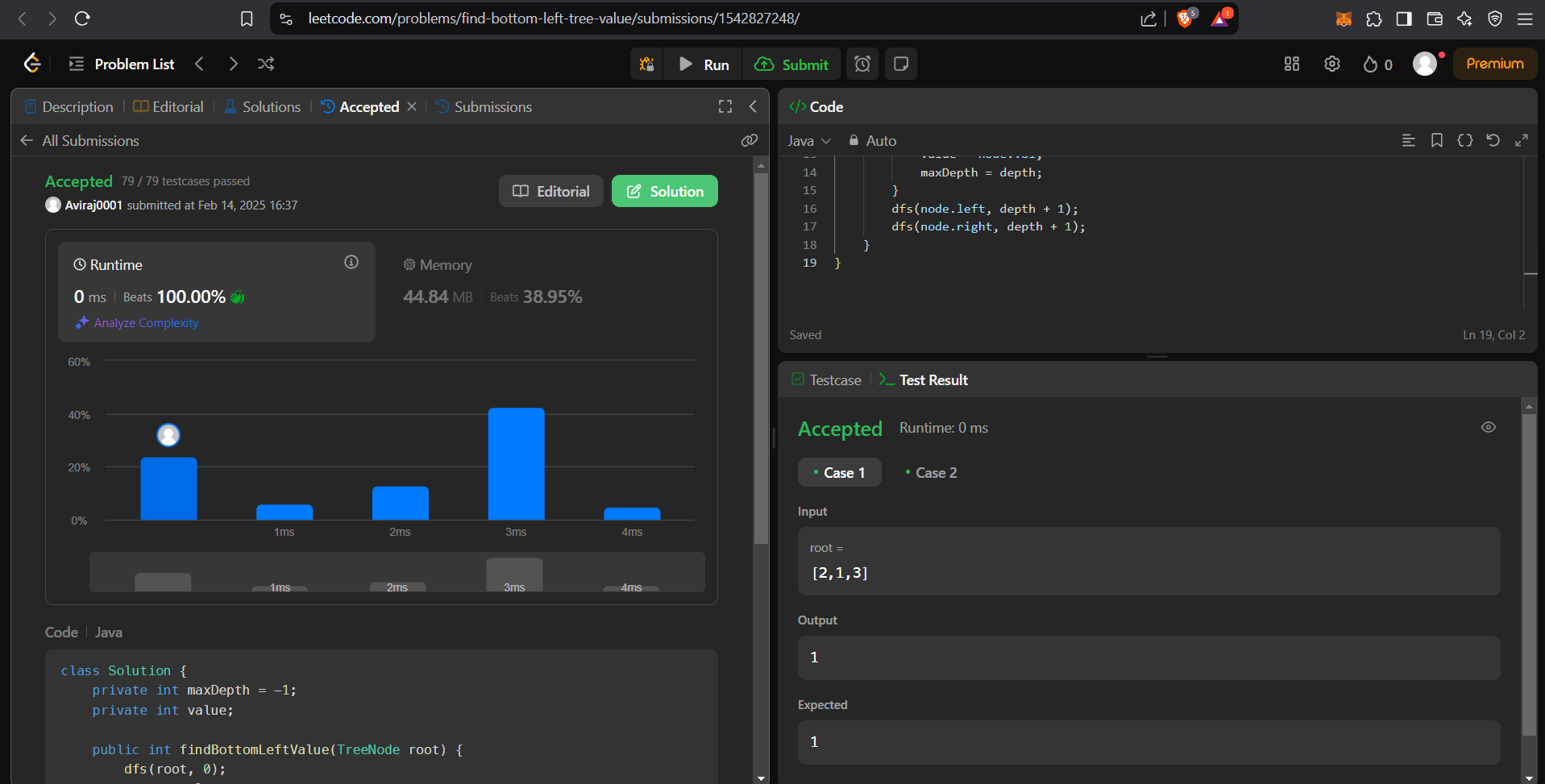
dfs(node.left, depth + 1);

dfs(node.right, depth + 1);

}

}

Output:



**Q Binary Tree Maximum Path Sum**

Code:

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

private int maxi=Integer.MIN\_VALUE;

public int path(TreeNode root){

if(root==null) return 0;

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

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

maxi=Math.max(l+r+root.val,maxi);

return root.val+Math.max(l,r);

}

public int maxPathSum(TreeNode root) {

maxi=Integer.MIN\_VALUE;

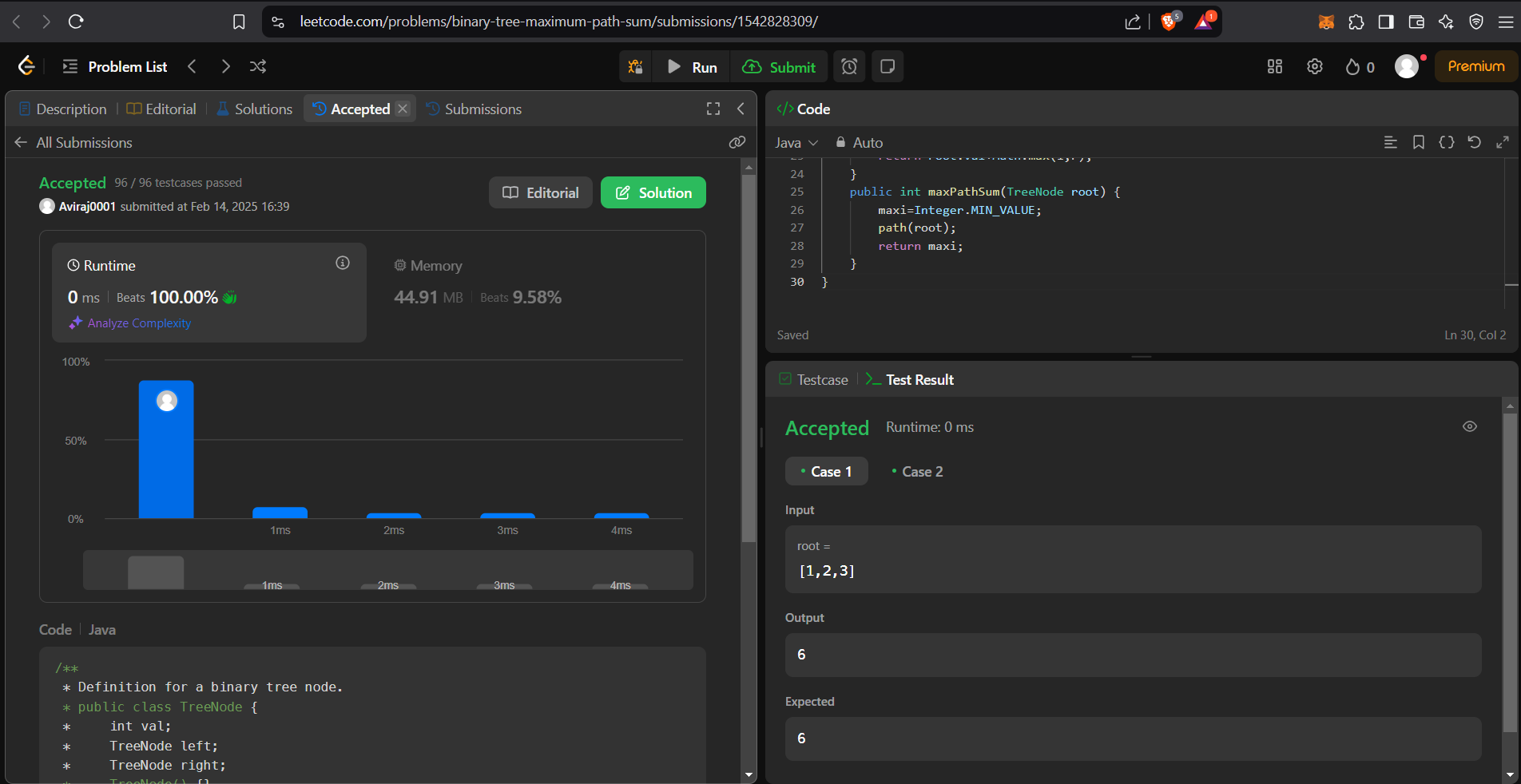
path(root);

return maxi;

}

}

Output:



**Q Vertical Order Traversal of a Binary Tree**

Code:

class Solution {

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

TreeMap<Integer, LinkedList<Integer>> table = new TreeMap<>();

L nkedList<TreeNode> list = new LinkedList<>();

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

list.add(root);

coor.add(0);

for(int level = 1; !list.isEmpty(); level++)

for(int n = list.size(), mask = level << 10; n-- != 0;){

root = list.pollFirst();

int x = coor.pollFirst();

if(!table.containsKey(x)) table.put(x, new LinkedList<Integer>());

table.get(x).add(mask|root.val);

if(root.left != null){list.addLast(root.left); coor.addLast(x-1);}

if(root.right != null){list.addLast(root.right); coor.addLast(x+1);}

}

int mask = 1023;

List<List<Integer>> ans = new ArrayList<>(table.size());

for(Map.Entry<Integer, LinkedList<Integer>> set : table.entrySet()){

LinkedList<Integer> l = set.getValue();

Collections.sort(l);

for(int i = 0; i != l.size(); i++) l.set(i, l.get(i)&mask);

ans.add(l);

}

return ans;

}

}

Output:

