**ASSIGNMENT -3 (ADVANCED PROGRAMMING)**

1. **Problem 1:** Binary Tree In Order Traversal
2. **Implementation/Code:**

class Solution {

public List<Integer> inorder(TreeNode root, List<Integer> list){

if(root == null)return list;

inorder(root.left,list);

list.add(root.val);

inorder(root.right,list);

return list;

}

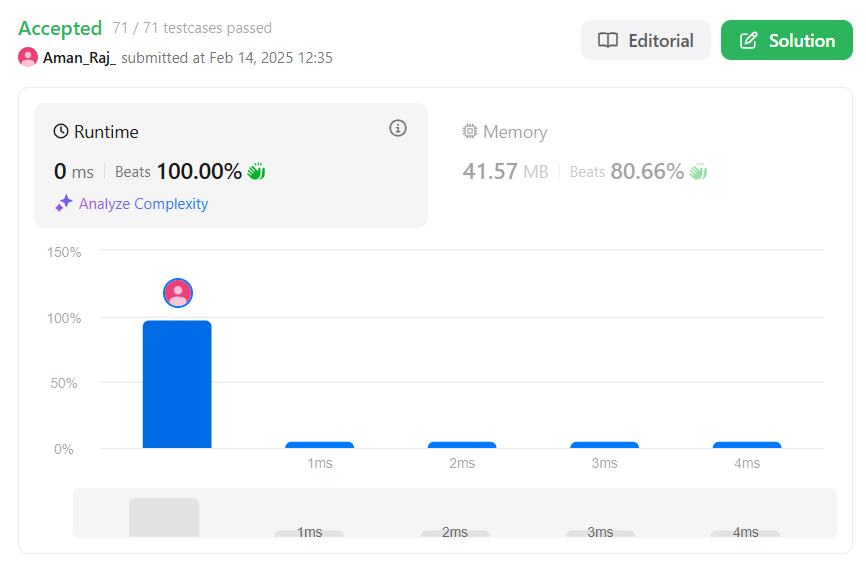
public List<Integer> inorderTraversal(TreeNode root) {

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

return inorder(root,list); }

}

1. **Output:**

****

1. **Problem 2:** Symmetric Tree
2. **Implementation/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) {

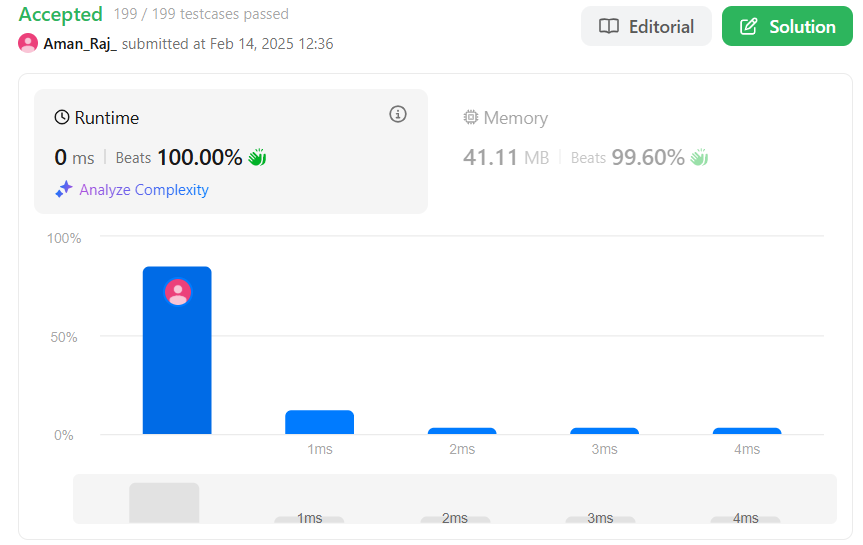
return false; }

if (node1.val != node2.val) {

return false; }

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

1. **Output:**

****

1. **Problem 3:** Maximum Depth of Binary Tree
2. **Implementation/code:**

public class Solution {

public int maxDepth(TreeNode root) {

return height(root); }

private int height(TreeNode node) {

if (node == null) return 0;

int leftHeight = height(node.left);

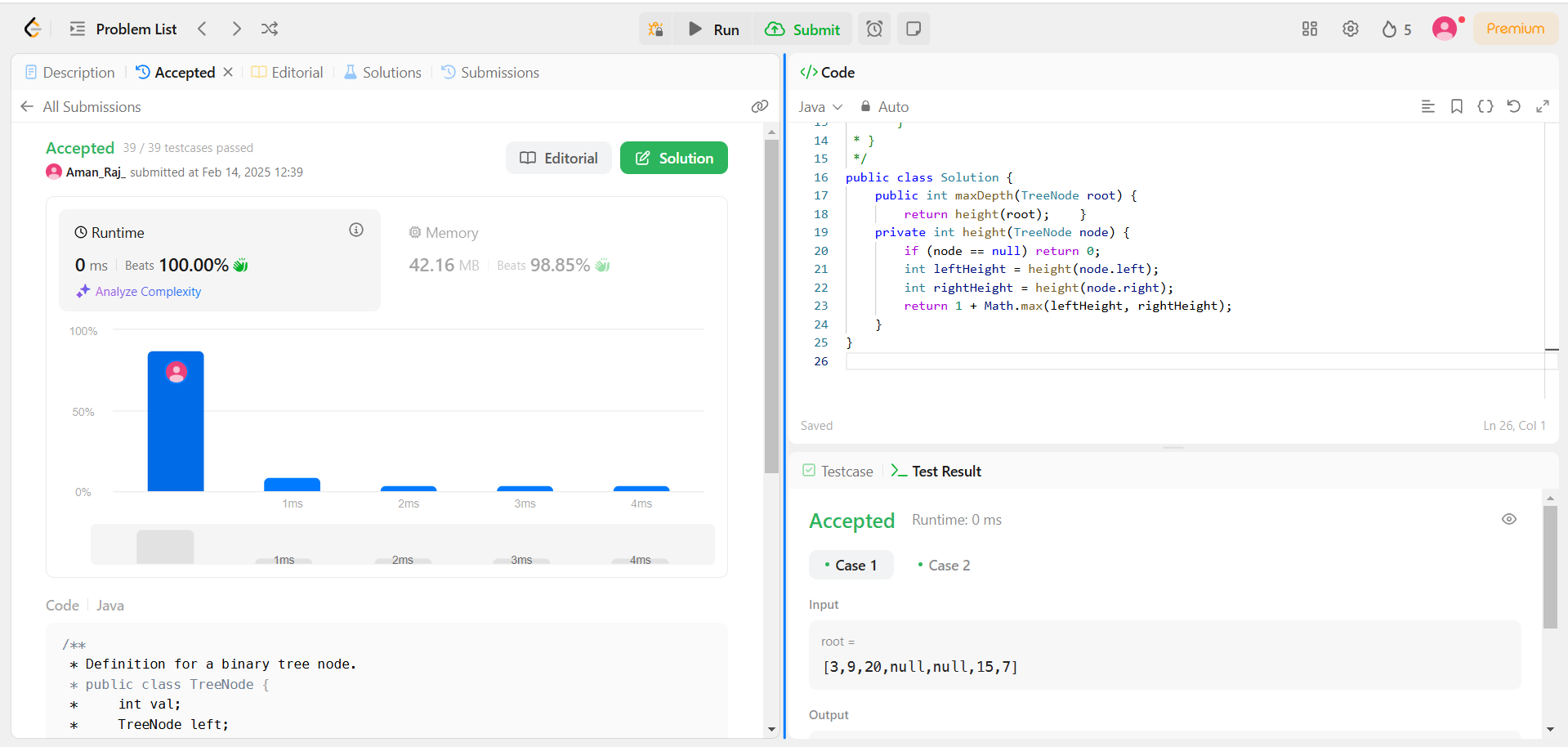
int rightHeight = height(node.right);

return 1 + Math.max(leftHeight, rightHeight);

}

}

1. **Output:**

****

1. **Problem 4:** Validate Binary search Tree
2. **Implementation/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;

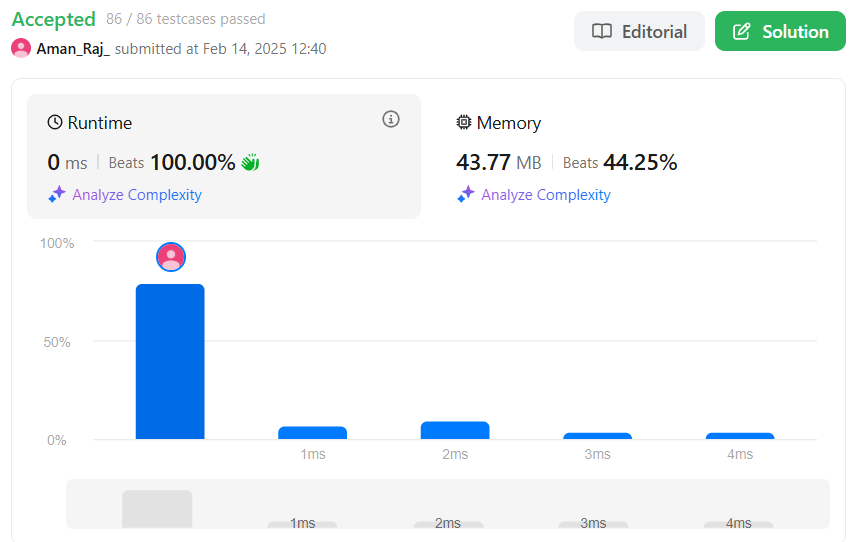
if (minVal >= root.val) return false;

minVal = root.val;

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

return true; } }

1. **Output:**

****

1. **Problem 5:** Kth Smallest Element in a BST
2. **Implementation/Code:**

class Solution {

public int kthSmallest(TreeNode root, int k) {

int count = countNodes(root.left);

if (k <= count) {

return kthSmallest(root.left, k);

} else if (k > count + 1) {

return kthSmallest(root.right, k-1-count);

}

return root.val; }

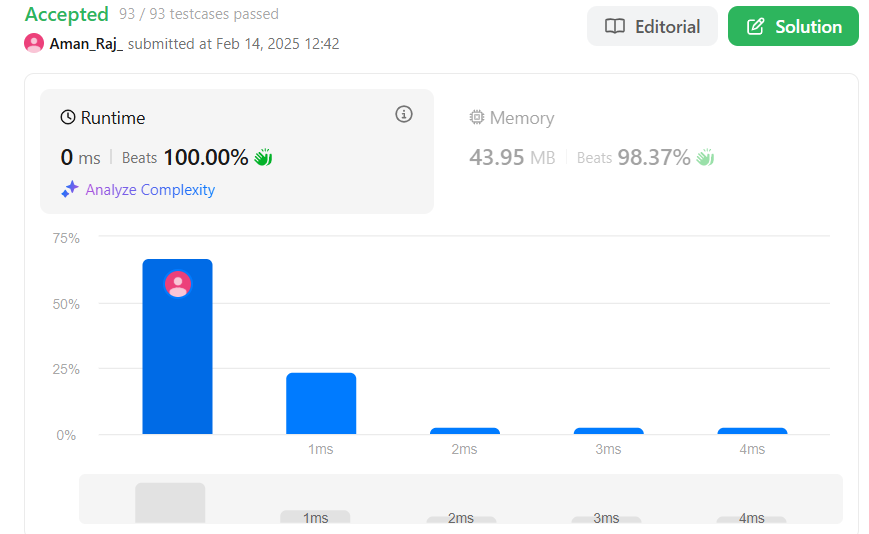
public int countNodes(TreeNode n) {

if (n == null) return 0;

return 1 + countNodes(n.left) + countNodes(n.right);

}}

1. **Output:**

****

1. **Problem 6:** Binary Tree Level Order Traversal
2. **Implementation/Code:**

class Solution {

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

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

if (root == null) return result;

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

queue.offer(root);

while (!queue.isEmpty()) {

int size = queue.size();

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

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

TreeNode node = queue.poll();

level.add(node.val);

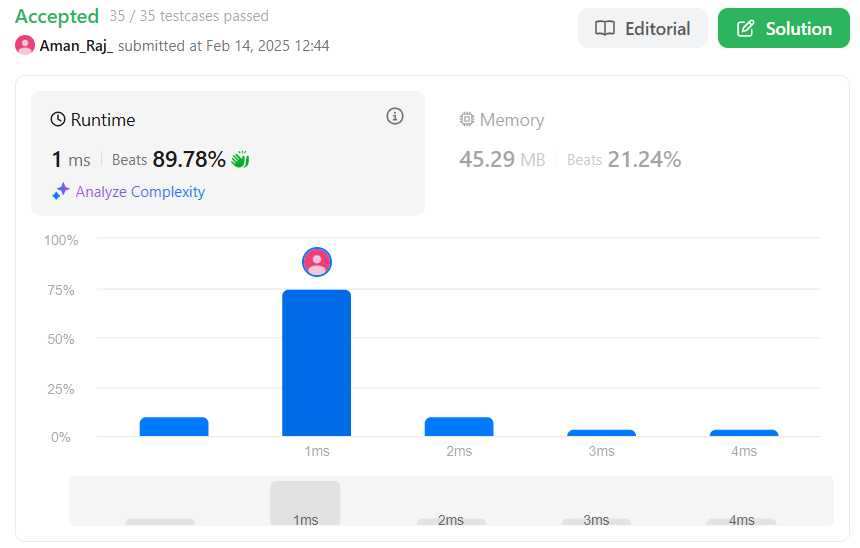
if (node.left != null) queue.offer(node.left);

if (node.right != null) queue.offer(node.right); }

result.add(level); }

return result; }}

1. **Output:**

****

1. **Problem 7:** Binary Tree Level Order Traversal II
2. **Implementation/code:**

class Solution {

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

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

if (root == null) return result;

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

queue.add(root);

while (!queue.isEmpty()) {

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

int size = queue.size();

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

TreeNode node = queue.poll();

level.add(node.val);

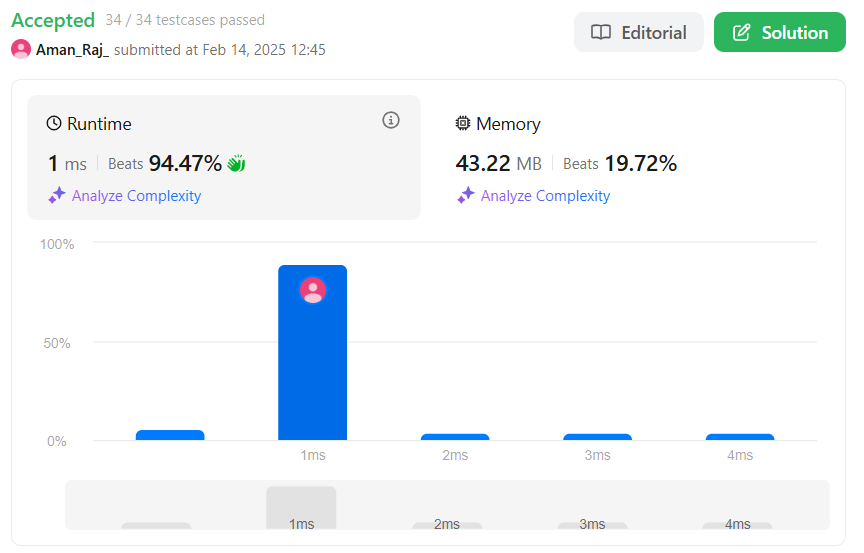
if (node.left != null) queue.add(node.left);

if (node.right != null) queue.add(node.right); }

result.add(0, level); }

return result; }}

1. **Output:**

****

1. **Problem 8:** Binary Tree Zig Zag Level Order Traversal
2. **Implementation/code:**

class Solution {

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

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

if (root == null) return res;

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

queue.add(root);

boolean leftToRight = true;

while (!queue.isEmpty()) {

int size = queue.size();

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

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

TreeNode node = queue.poll();

if (leftToRight) level.addLast(node.val);

else level.addFirst(node.val);

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

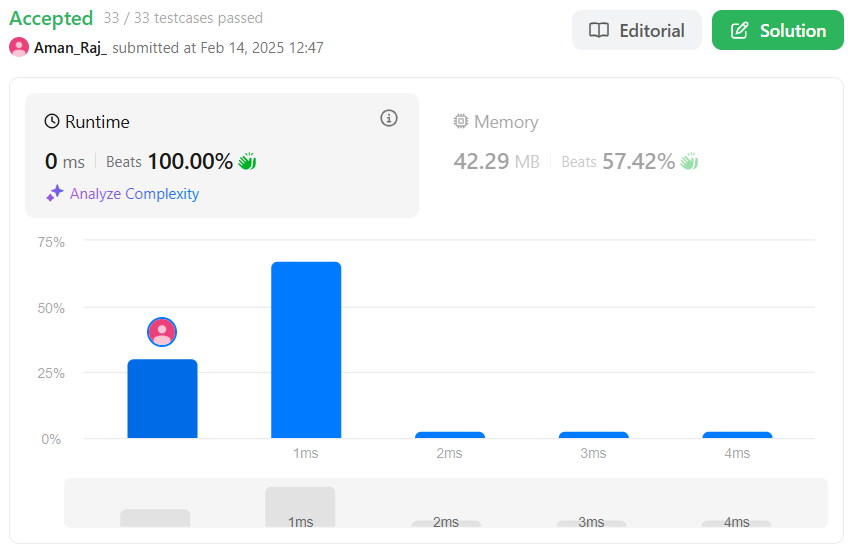
if (node.right != null) queue.add(node.right); }

res.add(level);

leftToRight = !leftToRight; }

return res; }}

1. **Output:**

****

1. **Problem 9:** Binary Tree Right Side View
2. **Implementation/code:**

class Solution {

public List<Integer> rightSideView(TreeNode root) {

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

if (root == null) return res;

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

queue.add(root);

while (!queue.isEmpty()) { int size = queue.size();

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

TreeNode node = queue.poll();

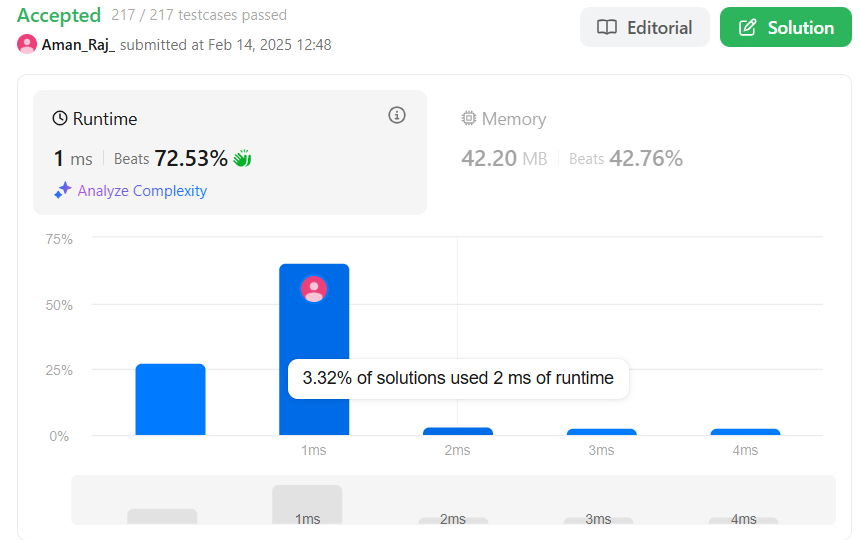
if (i == size - 1) res.add(node.val);

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

if (node.right != null) queue.add(node.right); } }

return res; }}

1. **Output:**

****

1. **Problem 10:** Construct Binary Tree From Inorder and Postorder Traversal
2. **Code:**

import java.util.\*;

class Solution {

int postIndex;

Map<Integer, Integer> inMap;

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

inMap = new HashMap<>();

postIndex = postorder.length - 1;

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

inMap.put(inorder[i], i); }

return build(postorder, 0, inorder.length - 1); }

private TreeNode build(int[] postorder, int inStart, int inEnd) {

if (inStart > inEnd) return null;

int rootVal = postorder[postIndex--];

TreeNode root = new TreeNode(rootVal);

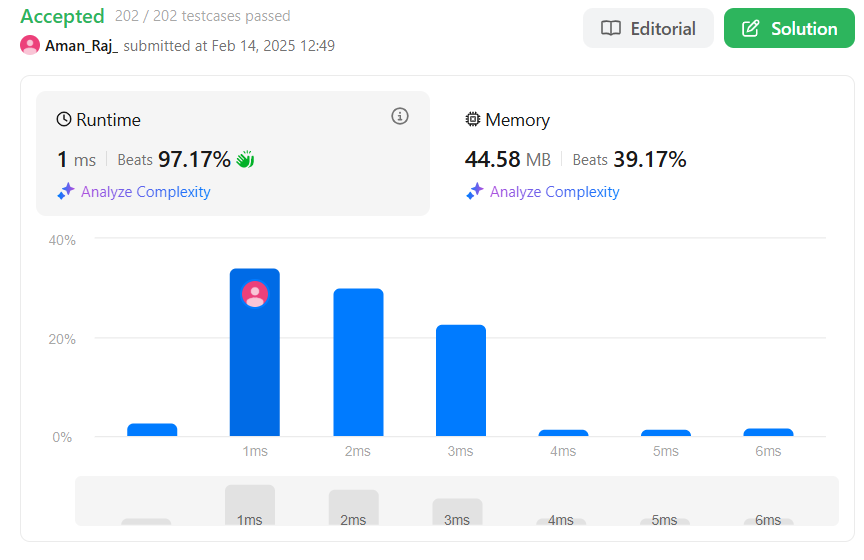
int inIndex = inMap.get(rootVal);

root.right = build(postorder, inIndex + 1, inEnd);

root.left = build(postorder, inStart, inIndex - 1);

return root; }}

1. **Output:**

****

1. **Problem 11:** Find Bottom Left Tree Value
2. **Code:**

class Solution {

public int findBottomLeftValue(TreeNode root) {

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

queue.add(root);

int bottomLeft = root.val;

while (!queue.isEmpty()) {

TreeNode node = queue.poll();

bottomLeft = node.val;

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

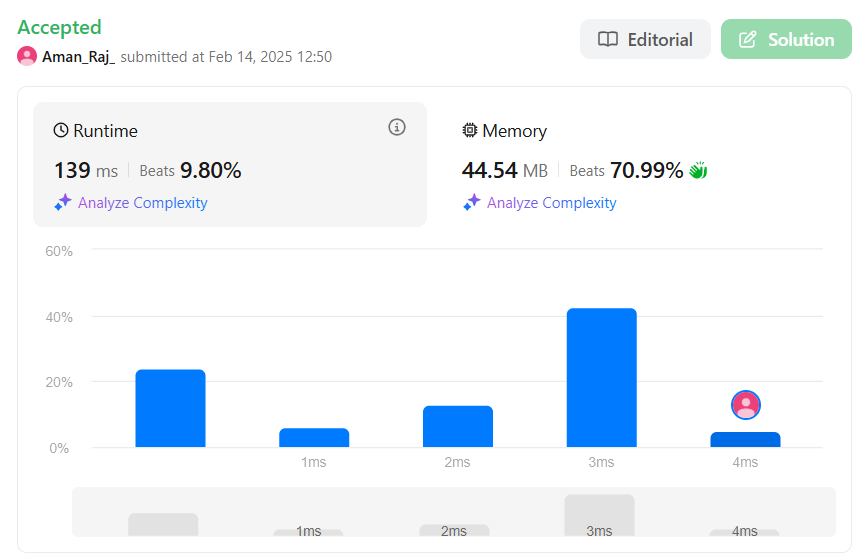
if (node.left != null) queue.add(node.left);

}

return bottomLeft;

}}

1. **Output:**

****

1. **Problem 12:** Binary Tree Maximum Path Sum
2. **Code:**

class Solution {

int maxSum = Integer.MIN\_VALUE;

public int maxPathSum(TreeNode root) {

dfs(root);

return maxSum; }

private int dfs(TreeNode node) {

if (node == null) return 0;

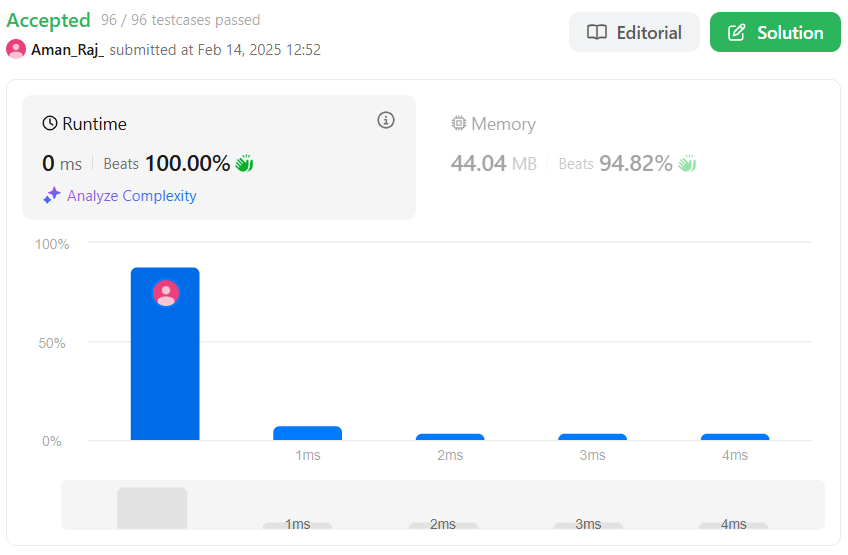
int left = Math.max(0, dfs(node.left));

int right = Math.max(0, dfs(node.right));

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

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

1. **Output:**

****

1. **Problem 13:** Vertical Order Traversal of Binary Tree
2. **Code:**

class Solution {

Map<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map;

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

if (root == null)

return null;

map = new TreeMap<>();

dfs(root, 0, 0);

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

for (int key : map.keySet()){

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

TreeMap<Integer, PriorityQueue<Integer>> tm = map.get(key);

for (int k : tm.keySet()){

PriorityQueue<Integer> pq = tm.get(k);

while (!pq.isEmpty()){

list.add(pq.poll());

}

}

res.add(list);

}

return res;

}

private void dfs(TreeNode root, int index, int level){

if (root == null)

return;

map.putIfAbsent(index, new TreeMap<>());

map.get(index).putIfAbsent(level, new PriorityQueue<>());

map.get(index).get(level).add(root.val);

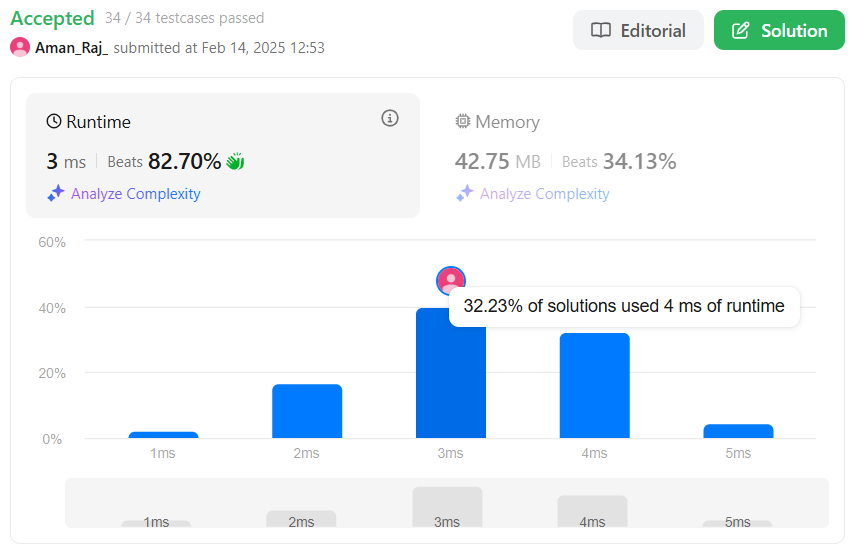
dfs(root.left, index - 1, level + 1);

dfs(root.right, index + 1, level + 1);

}

}

1. **Output:**

****