**Name:** Aditya Singh

**UID:** 22BCS15063

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

**Assignment – 3 Solutions:-**

1. **Binary Tree Inorder Traversal:-**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

vector<int>ans;

void inOrder(TreeNode\* root){

if(root!=NULL){

inOrder(root->left);

ans.push\_back(root->val);

inOrder(root->right);

}

}

vector<int> inorderTraversal(TreeNode\* root) {

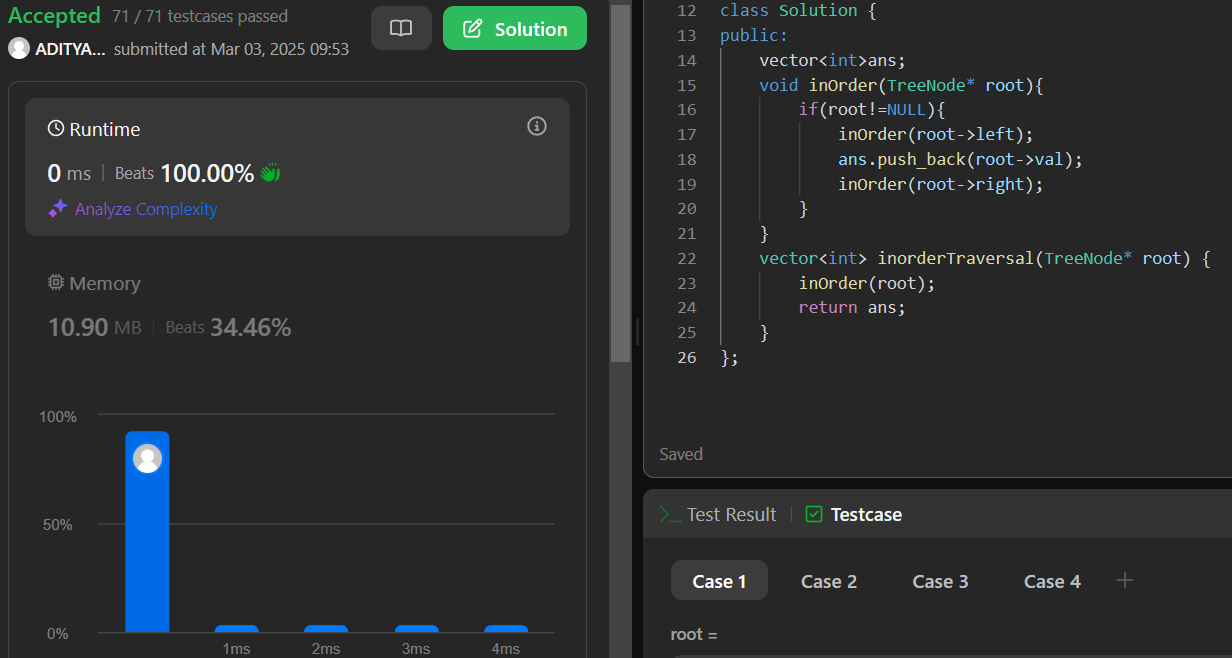
inOrder(root);

return ans;

}

};

Result:-



1. **Symmetric Tree:**

/\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

bool isTreeSymmetric(TreeNode\* leftRoot, TreeNode\* rightRoot){

if(leftRoot == nullptr && rightRoot == nullptr) return true;

if((leftRoot == nullptr && rightRoot != nullptr) || (leftRoot != nullptr && rightRoot == nullptr)){

return false;

}

if(leftRoot -> val != rightRoot -> val){

return false;

}

return isTreeSymmetric(leftRoot -> left, rightRoot -> right) && isTreeSymmetric(leftRoot -> right, rightRoot -> left);

}

bool isSymmetric(TreeNode\* root) {

return isTreeSymmetric(root -> left, root -> right);

}

};

Result:

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Maximum Depth of Binary Tree:**

/\*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 int maxDepth(TreeNode root) {

if(root == null){

return 0;

}

else{

int leftMax = maxDepth(root.left);

int rightMax = maxDepth(root.right);

return Math.max(leftMax, rightMax) + 1;

}

}

}

Result:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Validate Binary Search Tree:**

/\*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 Integer prev;

public boolean isValidBST(TreeNode root) {

prev = null;

return inOrder(root);

}

public boolean inOrder(TreeNode root){

if(root == null) return true;

if(!inOrder(root.left)){

return false;

}

if(prev != null && root.val <= prev){

return false;

}

prev = root.val;

return inOrder(root.right);

}

}

Result:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Kth Smallest Element in a BST:**

/\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

vector<int>ans;

void inOrder(TreeNode\* root,int k){

if(root!=NULL){

if(ans.size()==k) return;

inOrder(root->left,k);

ans.push\_back(root->val);

inOrder(root->right,k);

}

}

int kthSmallest(TreeNode\* root, int k) {

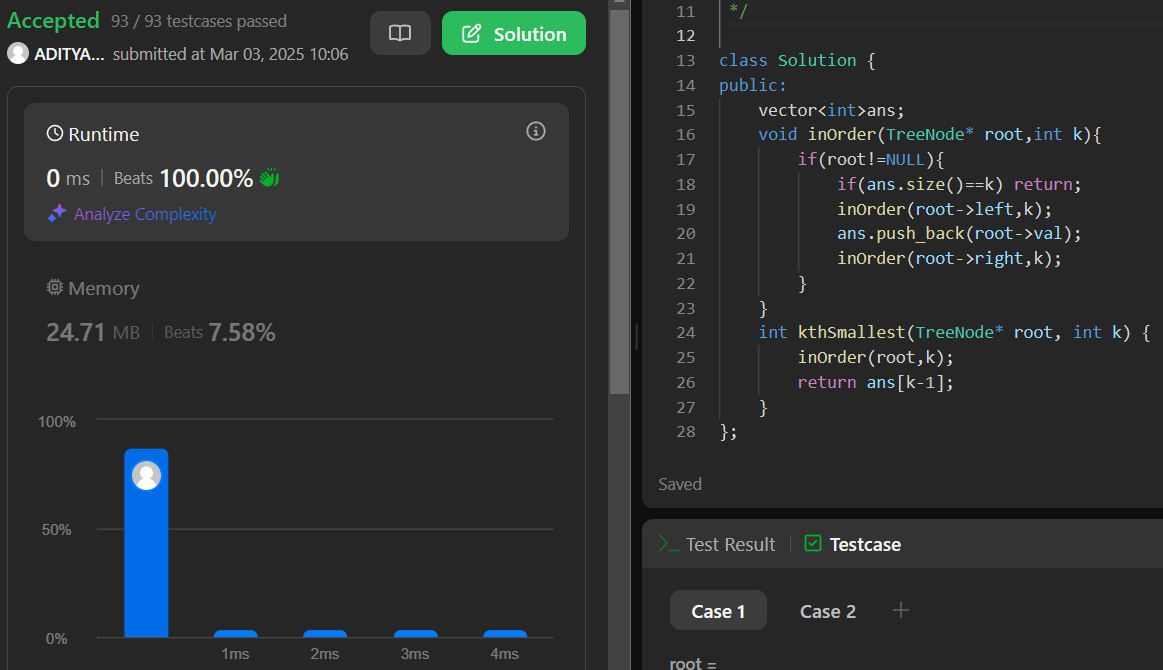
inOrder(root,k);

return ans[k-1];

}

};

Result:

****

1. **Binary Tree Level Order Traversal:**

/\* 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 {

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

public void order(TreeNode node, int level){

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

ans.add(new ArrayList<Integer>());

}

ans.get(level).add(node.val);

if(node.left != null){

order(node.left, level+1);

}

if(node.right != null){

order(node.right, level+1);

}

}

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

if(root == null){

return ans;

}

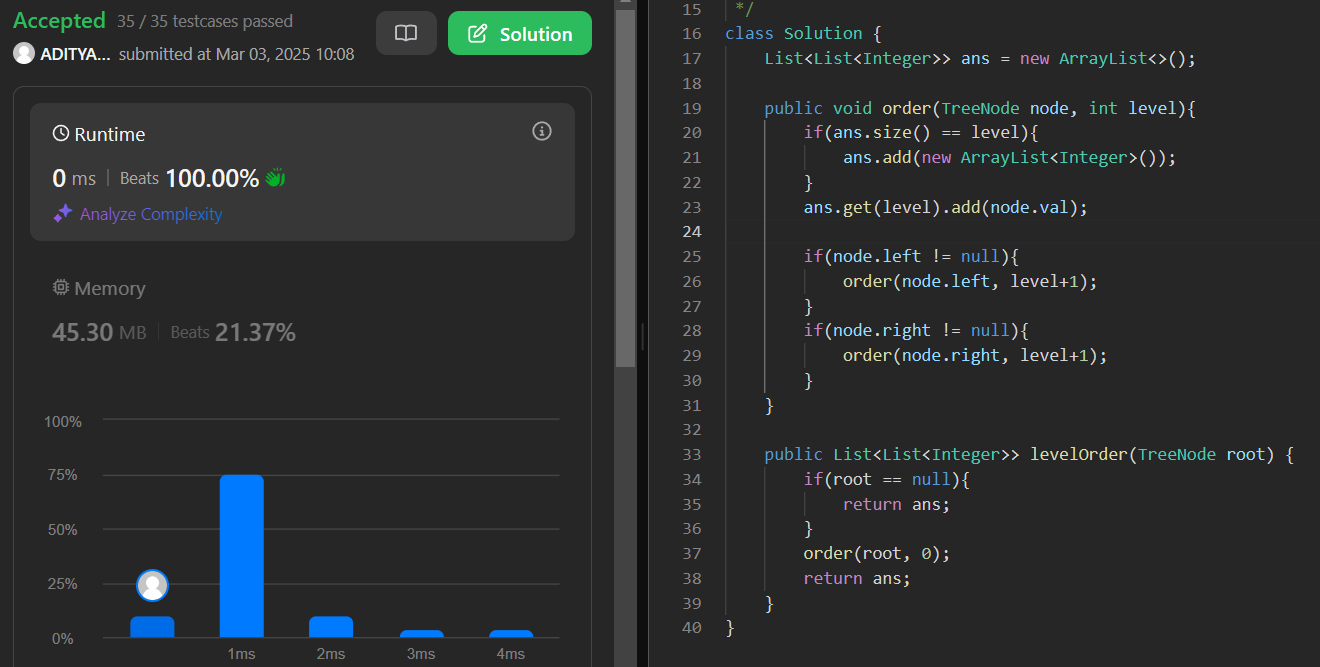
order(root, 0);

return ans;

}

}

Result:



1. **Binary Tree Level Order Traversal II:**

/\*\*

\* 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 {

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

public void order(TreeNode node, int level){

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

ans.add(new ArrayList<Integer>());

}

ans.get(level).add(node.val);

if(node.left != null){

order(node.left, level+1);

}

if(node.right != null){

order(node.right, level+1);

}

}

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

if(root == null){

return ans;

}

order(root, 0);

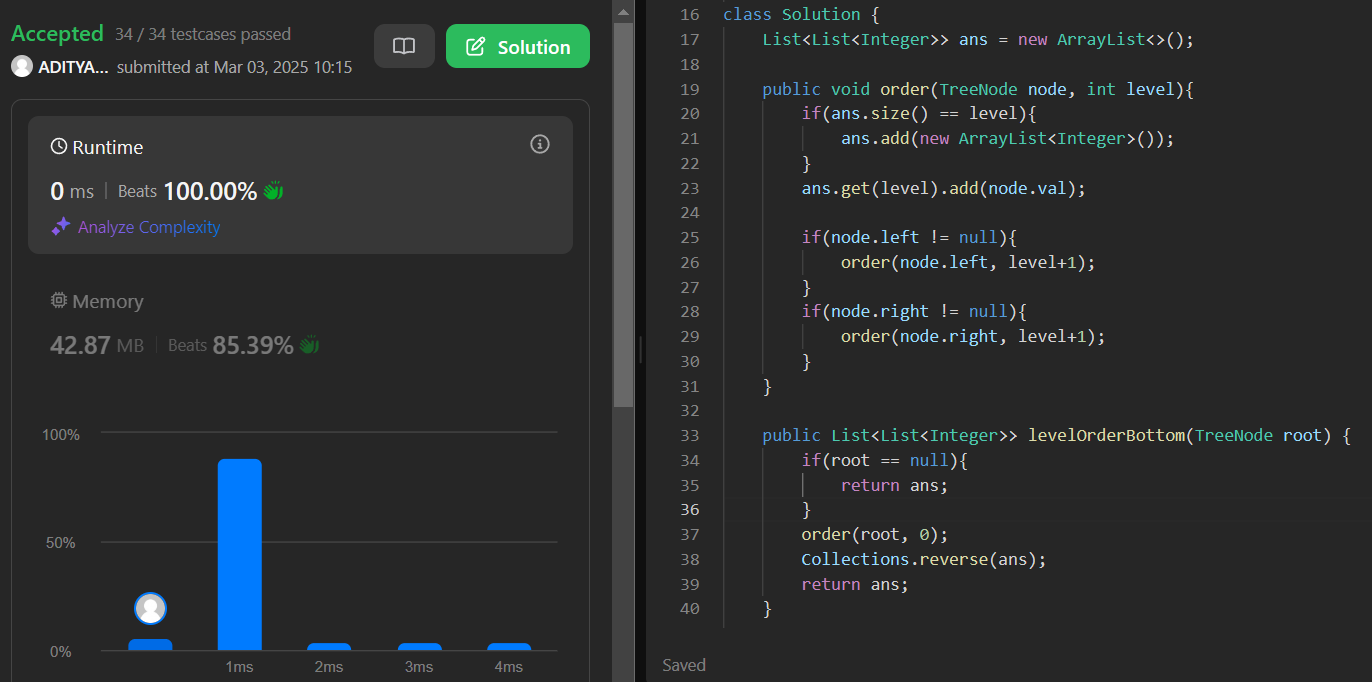
Collections.reverse(ans);

return ans;

}

}

Result:



1. **Binary Tree Zigzag Level Order Traversal:**

/\* 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 {

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

public void order(TreeNode node, int level){

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

ans.add(new ArrayList<Integer>());

}

if(level%2 == 1) ans.get(level).add(0, node.val);

else ans.get(level).add(node.val);

if(node.left != null){

order(node.left, level+1);

}

if(node.right != null){

order(node.right, level+1);

}

}

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

if(root == null){

return ans;

}

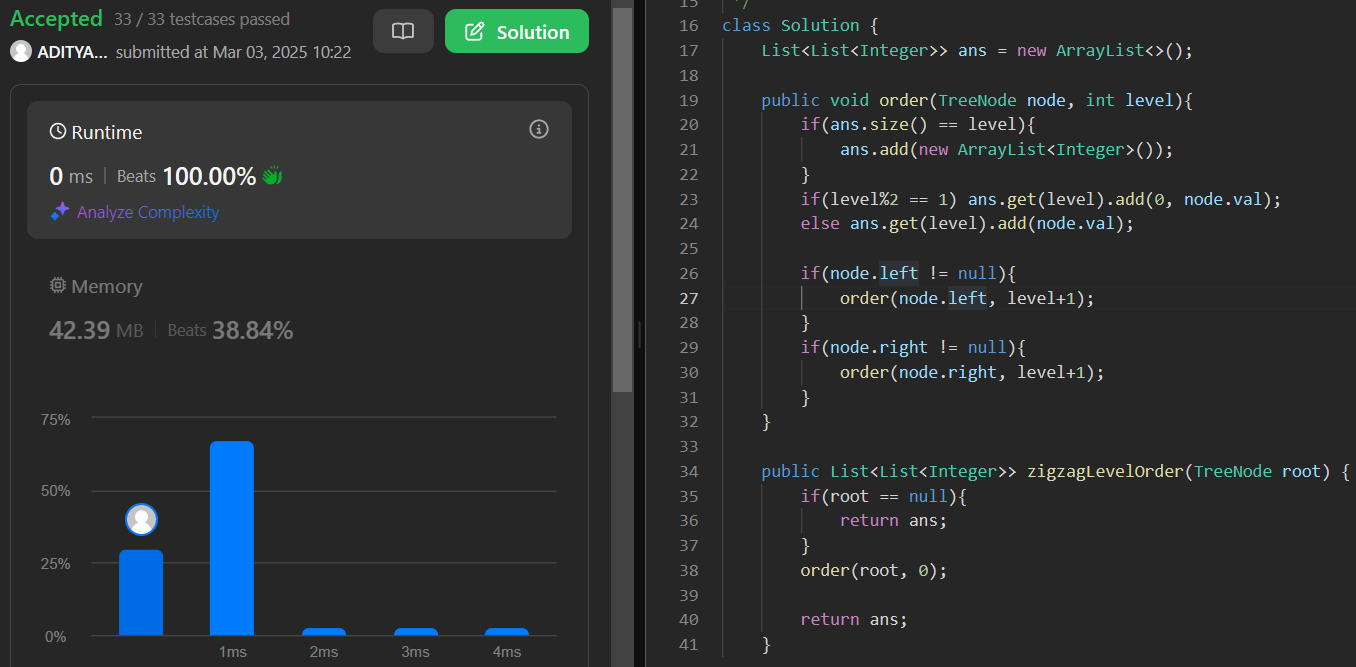
order(root, 0);

return ans;

}

}

Result:-



1. **Binary Tree Right Side View:**

/\*\*

\* 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<Integer> rightSideView(TreeNode root) {

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

if(root == null){

return result;

}

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

queue.add(root);

while(!queue.isEmpty()){

int levelSize = queue.size();

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

TreeNode currentNode = queue.poll();

if(i == levelSize - 1){

result.add(currentNode.val);

}

if(currentNode.left != null){

queue.add(currentNode.left);

}

if(currentNode.right != null){

queue.add(currentNode.right);

}

}

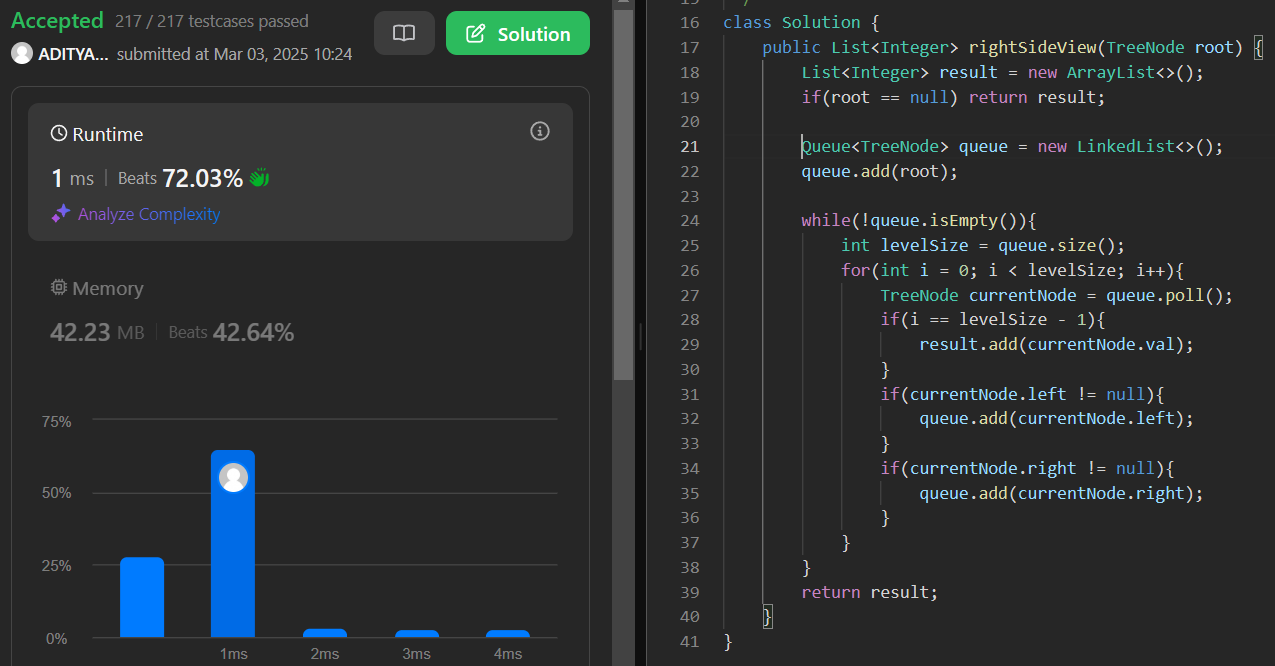
}

return result;

}

}

Result:



1. **Construct Binary Tree from Inorder and Post order Traversal:**

/\* 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 TreeNode buildTree(int[] inorder, int[] postorder) {

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

}

private TreeNode buildTree(int[] inorder, int inStart, int inEnd, int[] postorder, int postStart, int postEnd){

if (inStart > inEnd || postStart > postEnd) {

return null;

}

int rootVal = postorder[postEnd];

TreeNode root = new TreeNode(rootVal);

int rootIndex = 0;

for (int i = inStart; i <= inEnd; i++) {

if (inorder[i] == rootVal) {

rootIndex = i;

break;

}

}

int leftSize = rootIndex - inStart;

int rightSize = inEnd - rootIndex;

root.left = buildTree(inorder, inStart, rootIndex - 1, postorder, postStart, postStart + leftSize - 1);

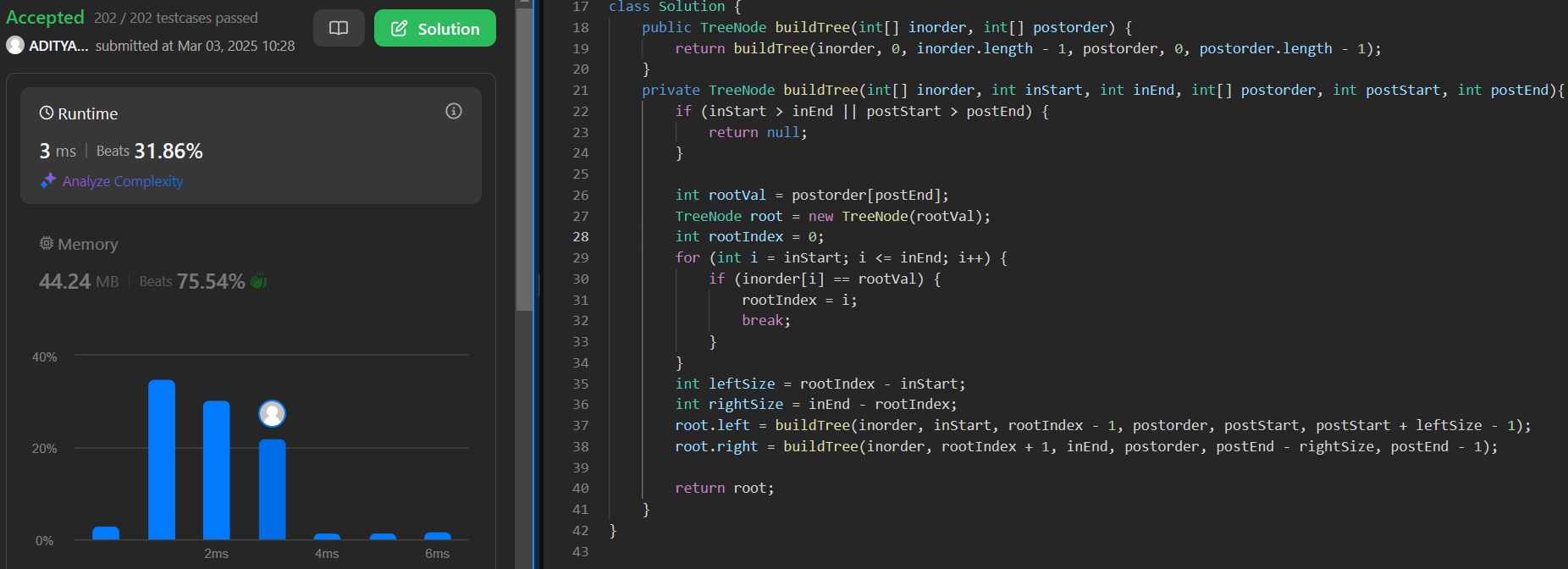
root.right = buildTree(inorder, rootIndex + 1, inEnd, postorder, postEnd - rightSize, postEnd - 1);

return root;

}

}

Result:



1. **Find Bottom Left Tree Value:**

/\* 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 int findBottomLeftValue(TreeNode root) {

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

q.add(root);

int last = 0;

while(!q.isEmpty()){

int n = q.size();

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

TreeNode curr = q.poll();

if(i == 0){

last = curr.val;

}

if(curr.left != null){

q.add(curr.left);

}

if(curr.right != null){

q.add(curr.right);

}

}

}

return last;

}

}

Result:

A screenshot of a computer screen

AI-generated content may be incorrect.

1. **Binary Tree Maximum Path Sum:**

/\*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 {

int maxSum = Integer.MIN\_VALUE;

public int maxGain(TreeNode node){

if(node == null){

return 0;

}

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

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

int priceNewPath = node.val + leftGain + rightGain;

maxSum = Math.max(maxSum, priceNewPath);

return node.val + Math.max(leftGain, rightGain);

}

public int maxPathSum(TreeNode root) {

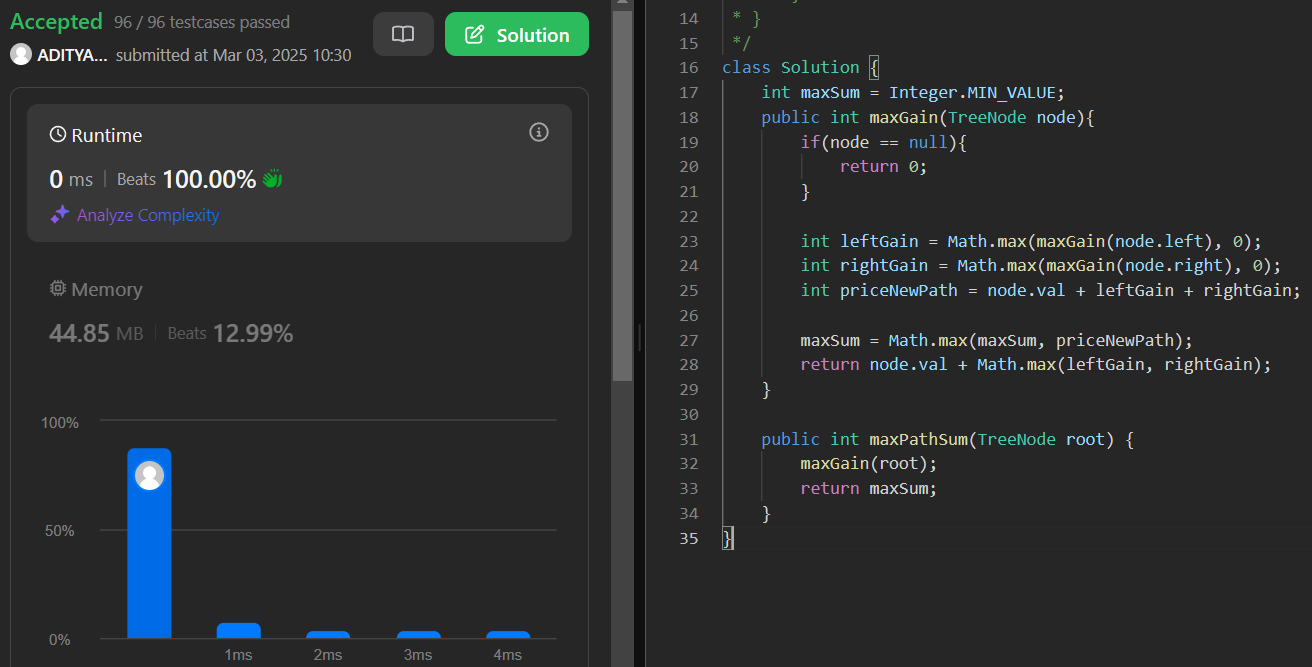
maxGain(root);

return maxSum;

}

}

Result:



1. **Vertical Order Traversal of a Binary Tree:**

/\* 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 {

Map<Integer, ArrayList<int[]>> map = new TreeMap<>();

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

dfs(root, 0, 0);

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

for(ArrayList<int[]> list: map.values()) {

Collections.sort(list, (a, b) -> a[0] == b[0] ?

Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

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

for(int[] num : list) {

current.add(num[1]);

}

result.add(current);

}

return result;

}

void dfs(TreeNode root, int index, int dept) {

if(root == null) {

return;

}

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

map.get(index).add(new int[]{dept, root.val});

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

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

}

}

Result:

**A screenshot of a computer screen

AI-generated content may be incorrect.**