Question: 1 Binary tree Inorder Traversal(94)

Code:

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

inorder(root, res);

return res;

}

private void inorder(TreeNode node, List<Integer> res) {

if (node == null) {

return;

}

inorder(node.left, res);

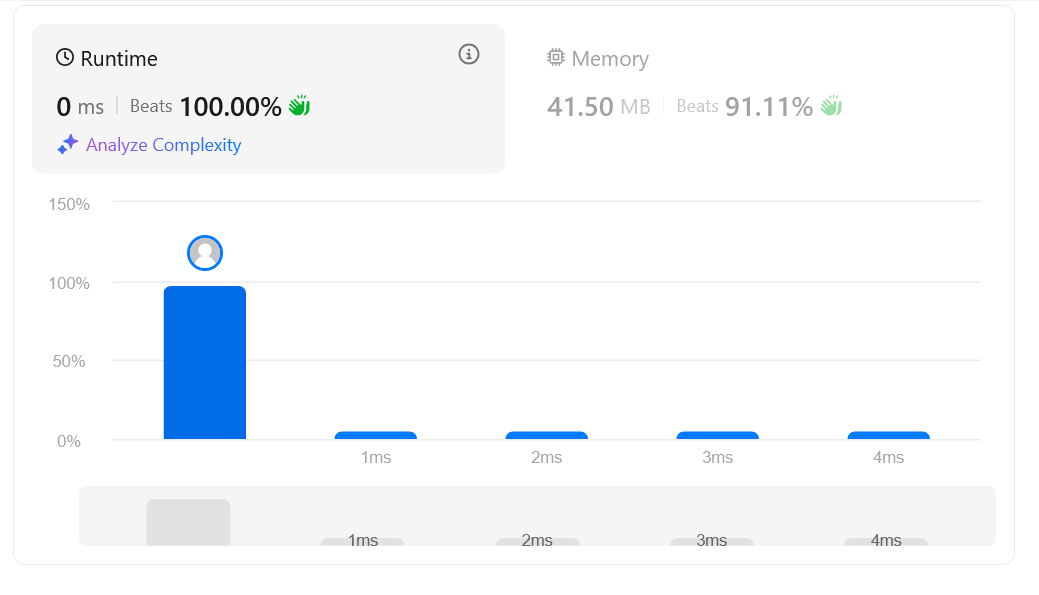
res.add(node.val);

inorder(node.right, res);

}

}

Output:



Question 2: Symmetric tree(101)

Code:

class Solution {

public boolean checkSymmetry(TreeNode left, TreeNode right){

// If both left and right are null.

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

return true;

}

// If either left is null or right is null whereas other one is not.

if((left == null && right != null) || (left != null && right == null)){

return false;

}

// If there values differ

if(left.val != right.val){

return false;

}

boolean check1 = checkSymmetry(left.right, right.left);

boolean check2 = checkSymmetry(left.left, right.right);

return check1 && check2;

}

public boolean isSymmetric(TreeNode root) {

// If there are no node in tree, then it is a symmetric tree.

if(root == null){

return true;

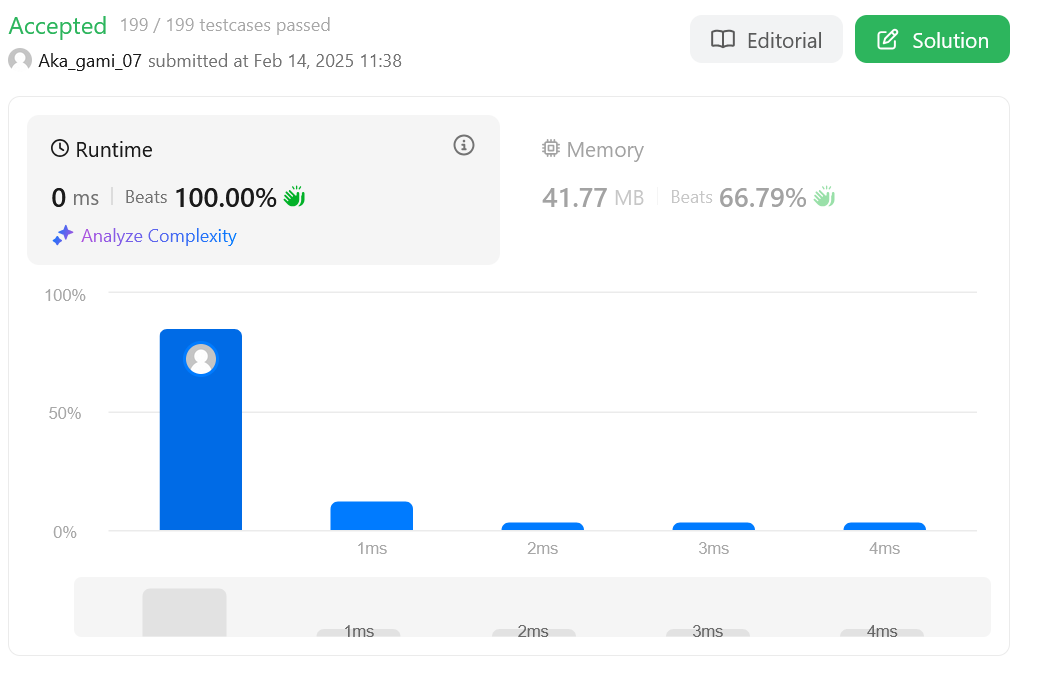
}

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

}

}

Output:



Question 3: Maximum Depth of Binary tree(104)

Code:

class Solution {

public int maxDepth(TreeNode root) {

if(root == null){

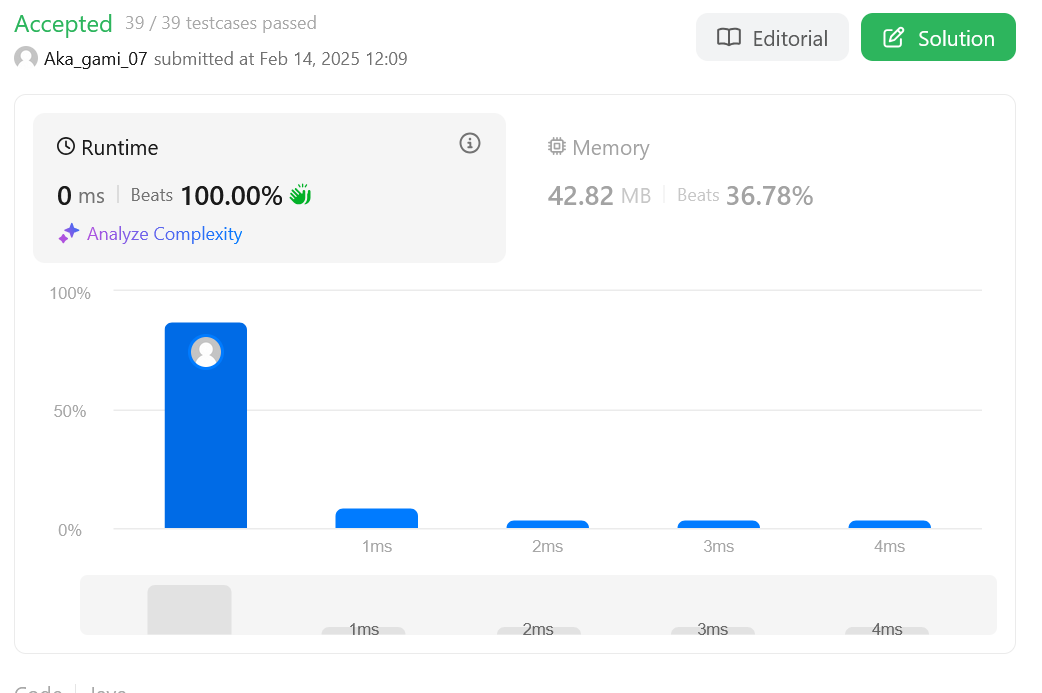
return 0;

}

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

}

}



Question: Validate Binary Search Tree(98)

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;

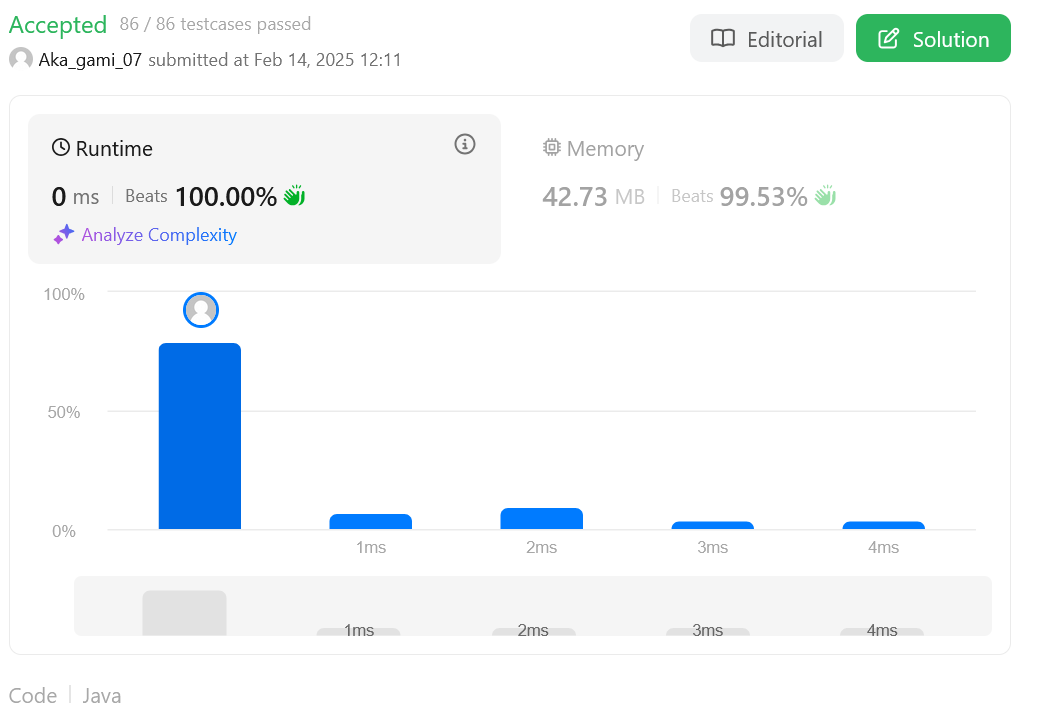
if (!(node.val > minimum && node.val < maximum)) return false;

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

}

}

Output:



Question: Kth Smallest Element in a BST(230)

Code:class Solution {

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

if (root == null) {

return;

}

inorder(root.left, arr);

arr.add(root.val);

inorder(root.right, arr);

}

public int kthSmallest(TreeNode root, int k) {

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

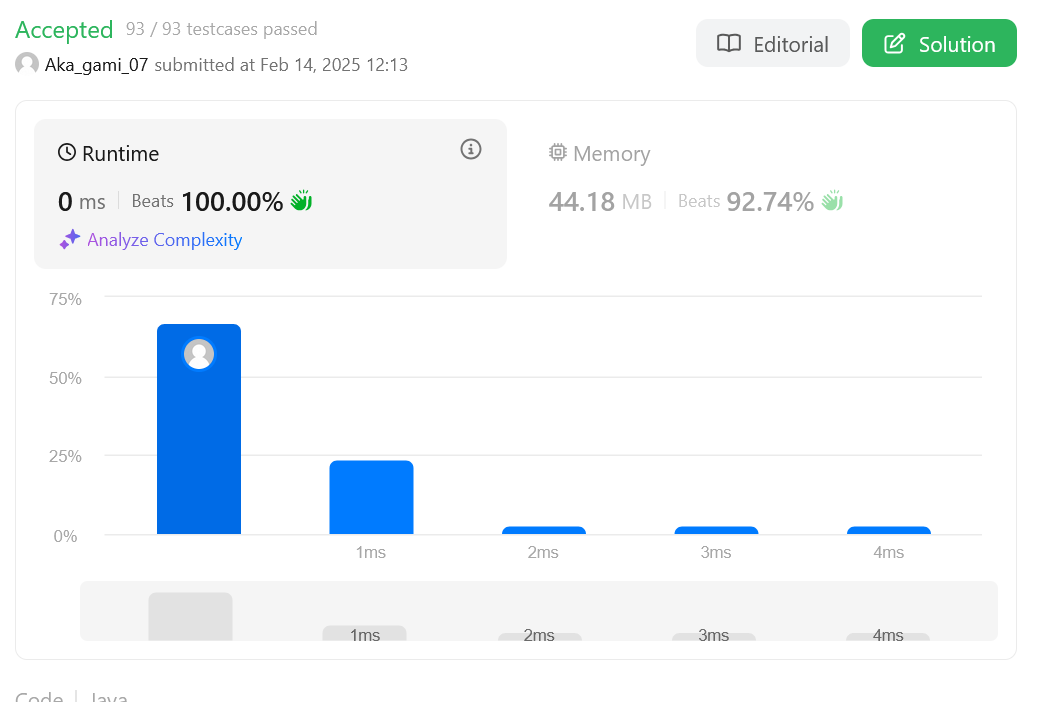
inorder(root, fin);

return fin.get(k - 1);

}

}

Output:



Question: Binary Tree Level Order traversal(102)  
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);

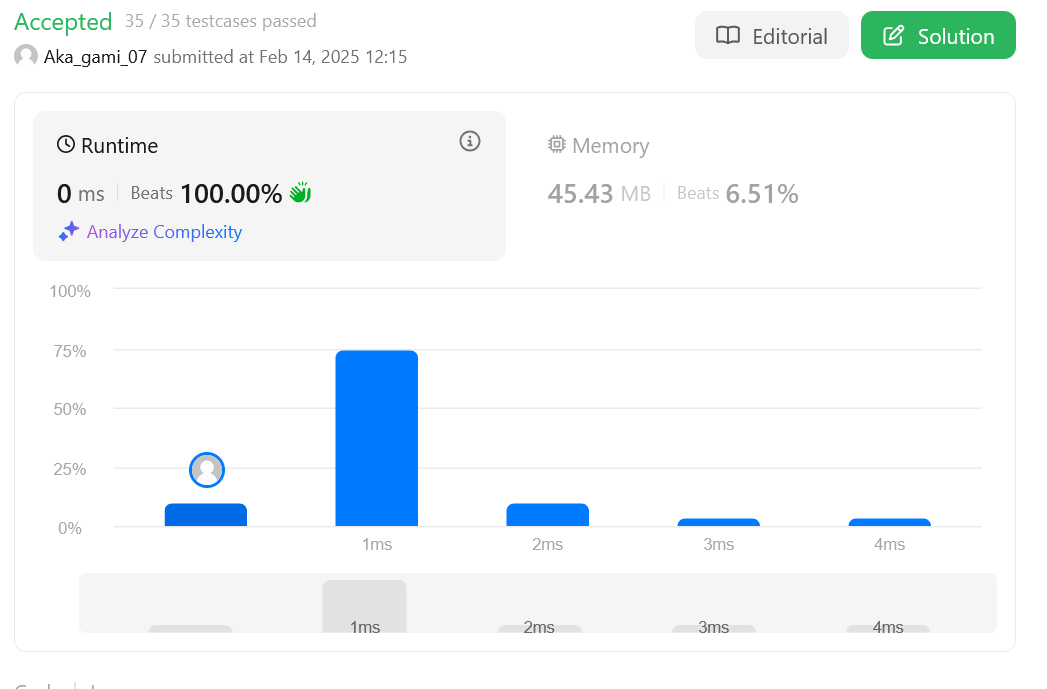
pre(root.left, l + 1, al);

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

}

}

Output:



Question: Binary Tree Level Order Traversal II(107)

Code:

class Solution {

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

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

if (root == null) {

return mainlist;

}

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

q.add(root);

q.add(null);

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

while (!q.isEmpty()) {

TreeNode curr = q.remove();

if (curr == null) {

mainlist.add(sublist);

sublist = new ArrayList<>();

if (q.isEmpty()) {

break;

} else {

q.add(null);

}

} else {

sublist.add(curr.val);

if (curr.left != null) {

q.add(curr.left);

}

if (curr.right != null) {

q.add(curr.right);

}

}

}

for (int i = 0; i < mainlist.size() / 2; i++) {

List<Integer> temp = mainlist.get(i);

mainlist.set(i, mainlist.get(mainlist.size() - i - 1));

mainlist.set(mainlist.size() - i - 1, temp);

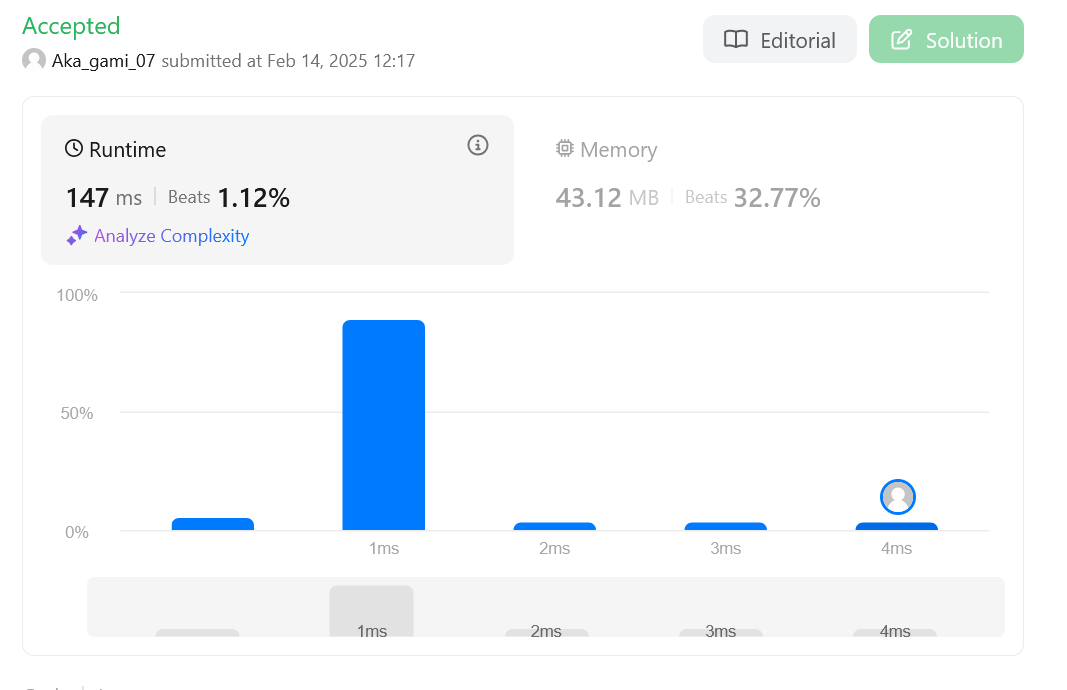
}

return mainlist;

}

}

Output:



Question: Binary Tree Zigzag Level Order traversal(103)

Code:

class Solution {

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

if (root == null)

return new ArrayList<List<Integer>>();

Deque<TreeNode> q = new ArrayDeque<>();

q.add(root);

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

List<Integer> l;

boolean flag = true;

while (!q.isEmpty()) {

int size = q.size();

l = new ArrayList<Integer>();

if (flag) {

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

TreeNode temp = q.pollFirst();

l.add(temp.val);

if (temp.left != null)

q.addLast(temp.left);

if (temp.right != null)

q.addLast(temp.right);

}

} else {

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

TreeNode temp = q.pollLast();

l.add(temp.val);

if (temp.right != null)

q.addFirst(temp.right);

if (temp.left != null)

q.addFirst(temp.left);

}

}

flag = !flag;

res.add(l);

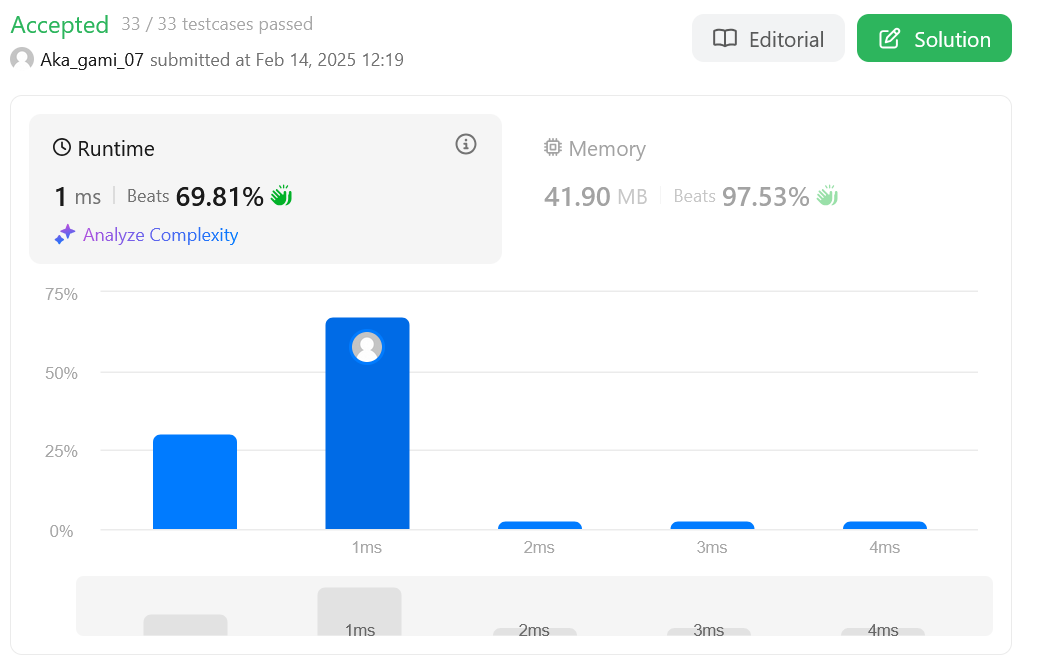
}

return res;

}

}

Output:



Question: Binary Tree Right Side view(199)

Code: class Solution {

public List<Integer> rightSideView(TreeNode root) {

if (root == null)

return new ArrayList<Integer>();

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

dq.add(root);

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

boolean flag = true;

while (!dq.isEmpty()) {

int size = dq.size();

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

TreeNode temp = dq.pollLast();

if (flag) {

res.add(temp.val);

flag = false;

}

if (temp.right != null)

dq.addFirst(temp.right);

if (temp.left != null)

dq.addFirst(temp.left);

}

flag = true;

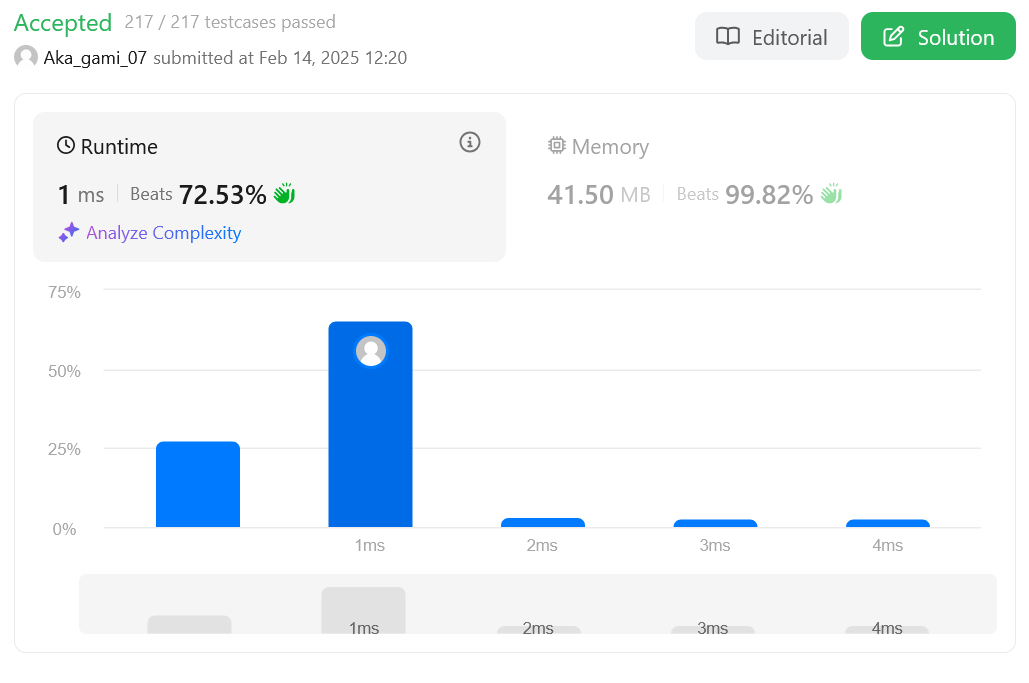
}

return res;

}

}

Output:



Question: Construct Binary Tree From Inorder and PostOrder Traversal(106)

Code:

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,

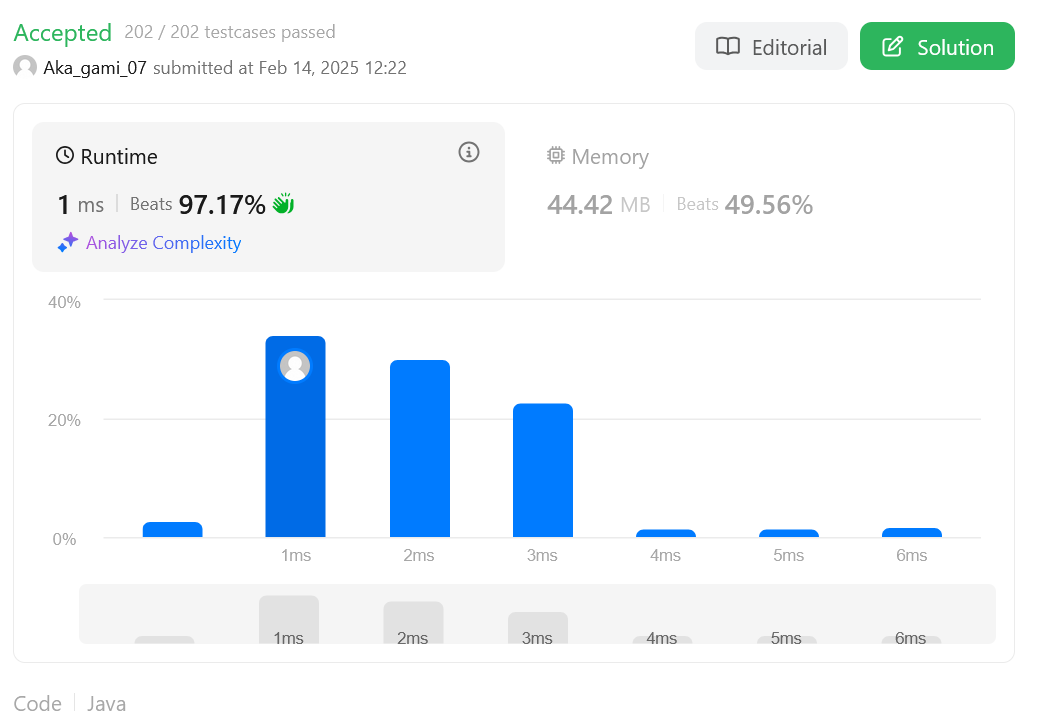
rec);

return root;

}

}

Output:



Question: Find bottom Left Tree Value(513)

Code:

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;

if (curr.left != null)

q.add(curr.left);

if (curr.right != null)

q.add(curr.right);

}

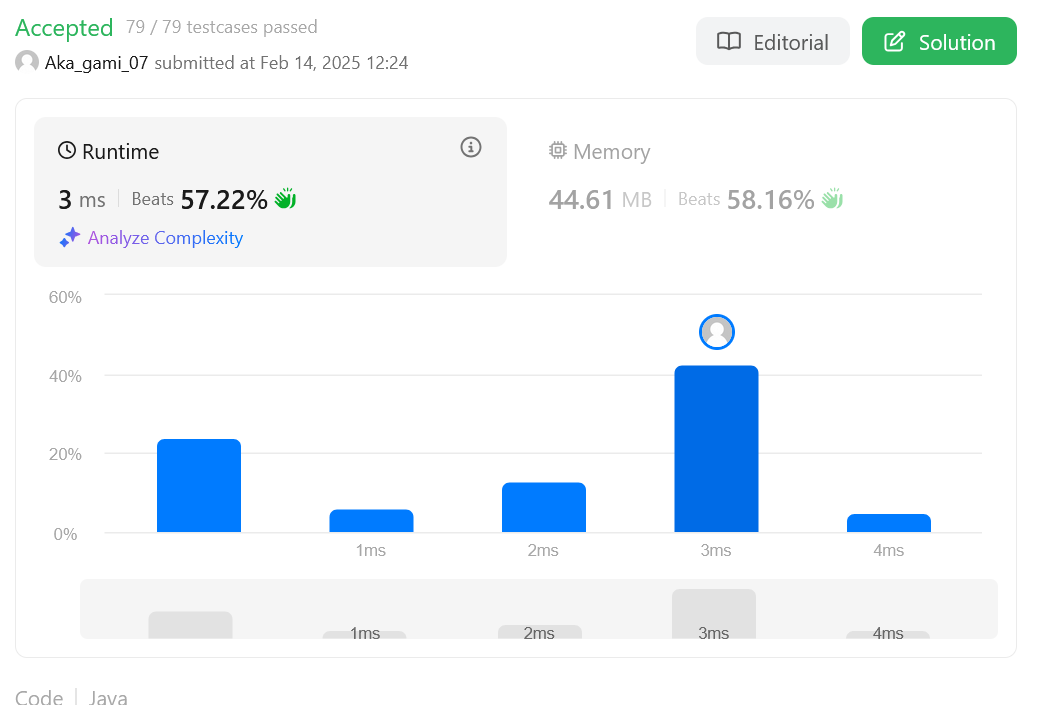
}

return last;

}

}

Output:



Question: Binary Tree Maximum Path Sum(124)

Code:

class Solution {

private int ans = Integer.MIN\_VALUE;

public int maxPathSum(TreeNode root) {

helper(root);

return ans;

}

private int helper(TreeNode root) {

if (root == null)

return 0;

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

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

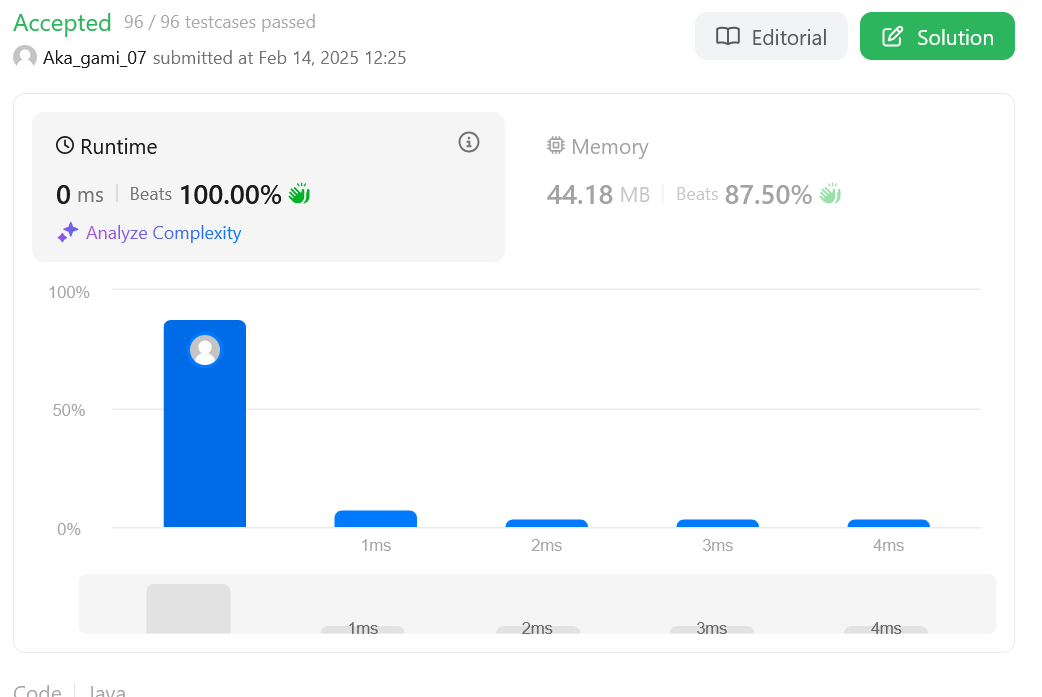
ans = Math.max(ans, root.val + left + right);

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

}

}

Output:



Question: Vertical Order Traversal of a Binary Tree(987)

Code:

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);

}

}

Output:

