**Name: Vanshika UID: 22BCS15478**

**Assignment-3**

1. **Binary Tree Inorder Traversal**

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;

}

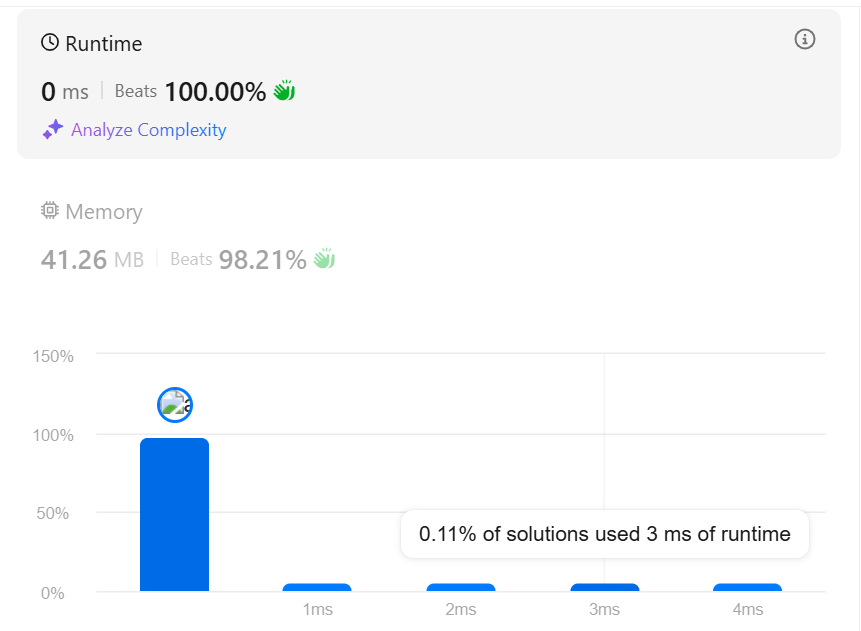
inorder(node.left, res);

res.add(node.val);

inorder(node.right, res);

}

}



1. **Symmetric Tree**

class Solution {

public boolean isSymmetric(TreeNode root) {

if (root == null) {

return false;

}

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

}

public boolean isSymmetricTree(TreeNode a, TreeNode b) {

if (a == null && b == null)

return true;

if (a == null || b == null)

return false;

if (a.val != b.val) {

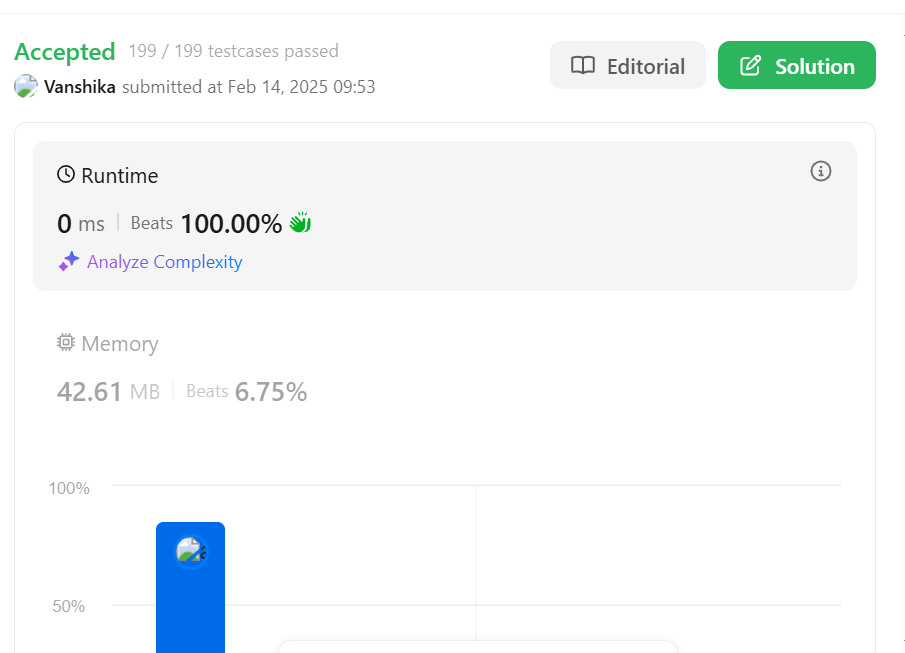
return false;

}

return isSymmetricTree(a.left, b.right) && isSymmetricTree(a.right, b.left);

}

}



1. **Maximum Depth of Binary Tree**

class Solution {

public int maxDepth(TreeNode root) {

if(root==null) return 0;

int ans=0;

int left=maxDepth(root.left);

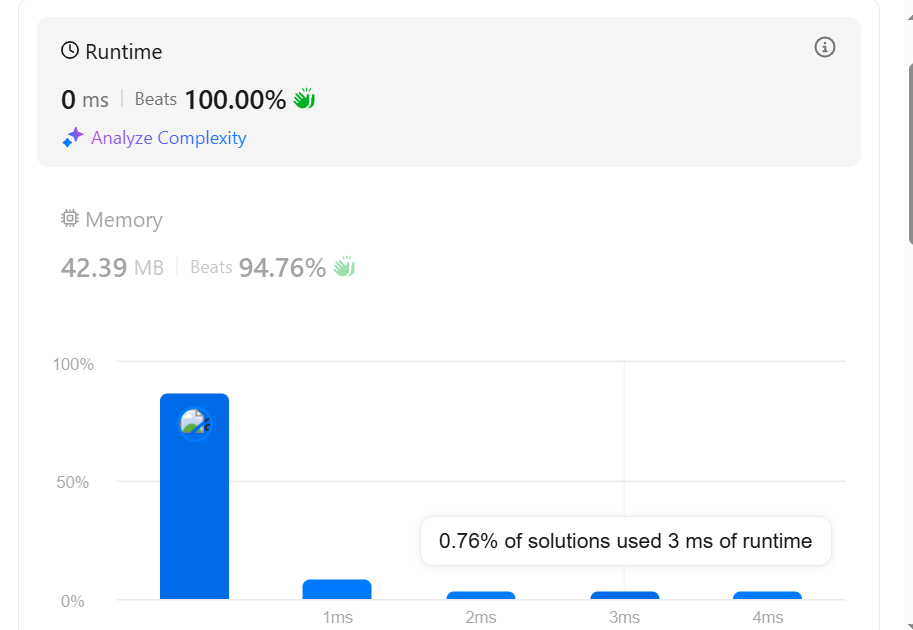
int right=maxDepth(root.right);

ans=1+Math.max(left,right);

return ans;

}

}



1. **Validate Binary Search Tree**

class Solution{

public boolean isValidBST(TreeNode root) {

if(root.left==null&&root.right==null)

return true;

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

}

boolean valid(long min, long max, TreeNode root)

{

if(root==null)

return true;

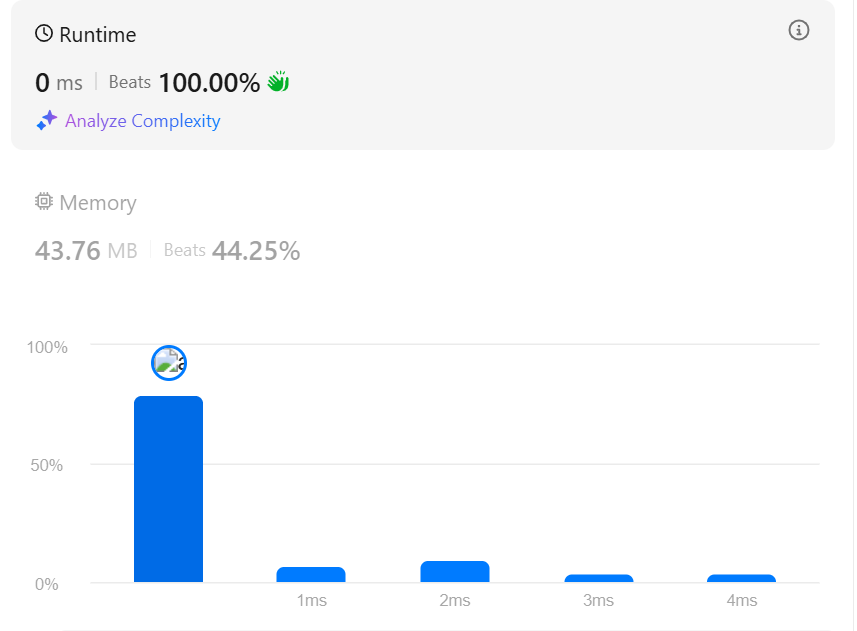
if(root.val>=max||root.val<=min)

return false;

return valid(min,root.val,root.left)&&valid(root.val,max,root.right);

}

}



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

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) {

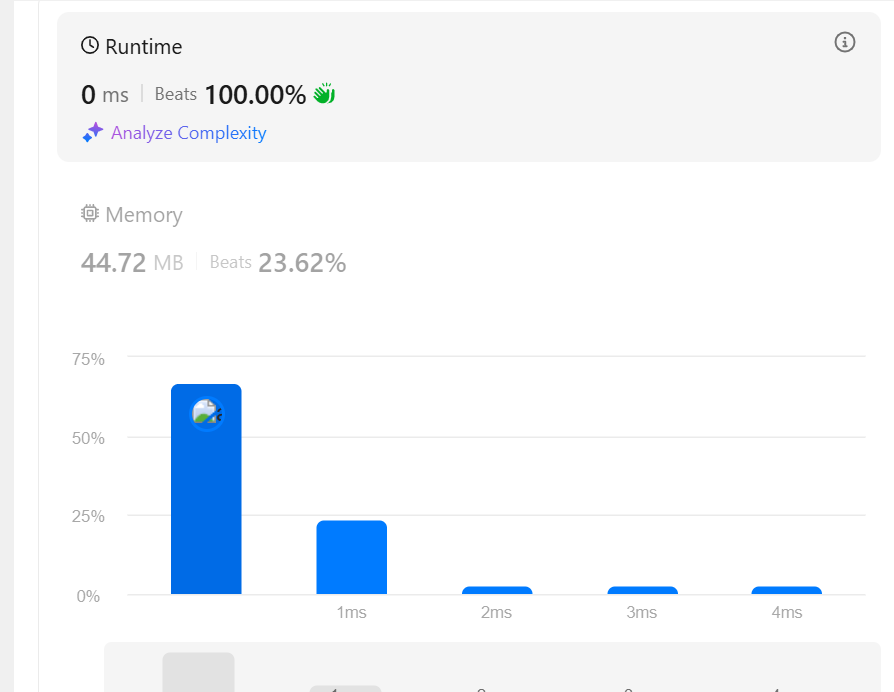
ArrayList<Integer> fin = new ArrayList<>();

inorder(root, fin);

return fin.get(k - 1);

}

}



1. **Binary Tree Level Order Traversal**

class Solution {

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

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

if(root == null)

return res;

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

q.add(root);

while(!q.isEmpty()){

int n = q.size();

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

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

TreeNode node = q.peek();

q.poll();

if(node.left != null)

q.add(node.left);

if(node.right != null)

q.add(node.right);

nodes.add(node.val);

}

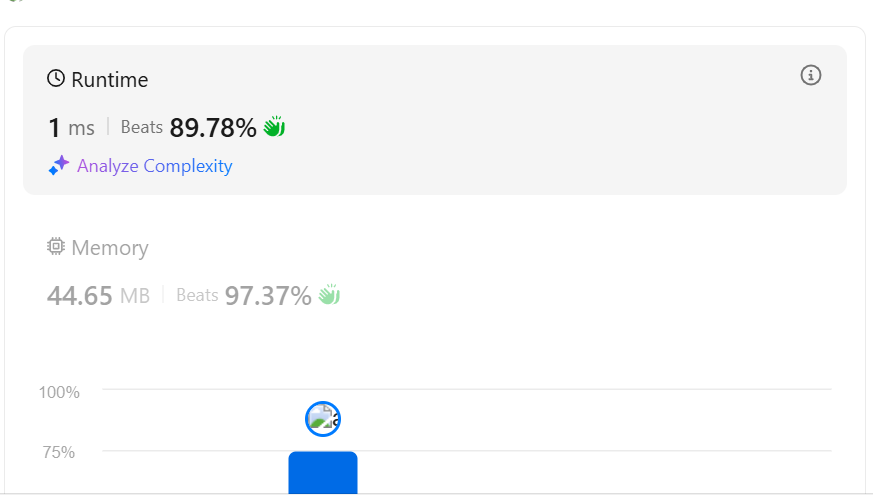
res.add(nodes);

}

return res;

}

}



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

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);//saved

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

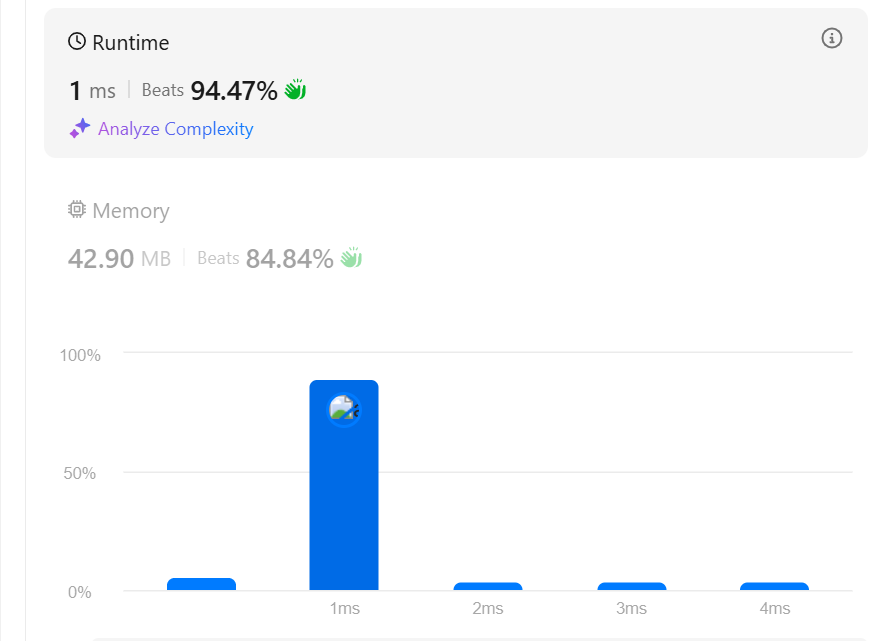
mainlist.set(mainlist.size()-i-1,temp);

}

return mainlist;

}

}



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

class Solution {

public int level(TreeNode root){

if(root==null) return 0;

int l=1+Math.max(level(root.left),level(root.right));

return l;

}

public void leftTr(TreeNode root,int i,List<Integer> smallAns){

if(root==null) return;

if(i==1){

smallAns.add(root.val);

}

leftTr(root.right,i-1,smallAns);

leftTr(root.left,i-1,smallAns);

}

public void rightTr(TreeNode root,int i,List<Integer> smallAns){

if(root==null) return;

if(i==1){

smallAns.add(root.val);

}

rightTr(root.left,i-1,smallAns);

rightTr(root.right,i-1,smallAns);

}

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

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

int n=level(root);

for(int i=1;i<=n;i++){

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

if(i%2==0){//left traversal

leftTr(root,i,smallAns);

}

else{//right traversal

rightTr(root,i,smallAns);

}

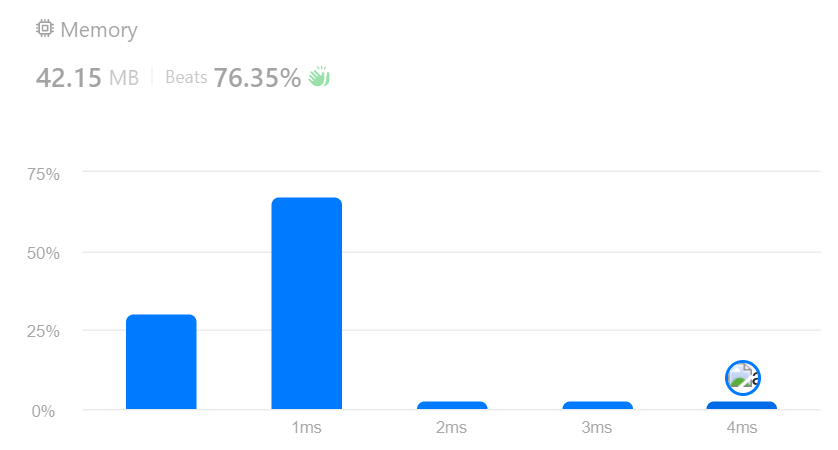
ans.add(smallAns);

}

return ans;

}

}



1. **Binary Tree Right Side View**

import java.util.\*;

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 node = queue.poll();

if (i == levelSize - 1) result.add(node.val);

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

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

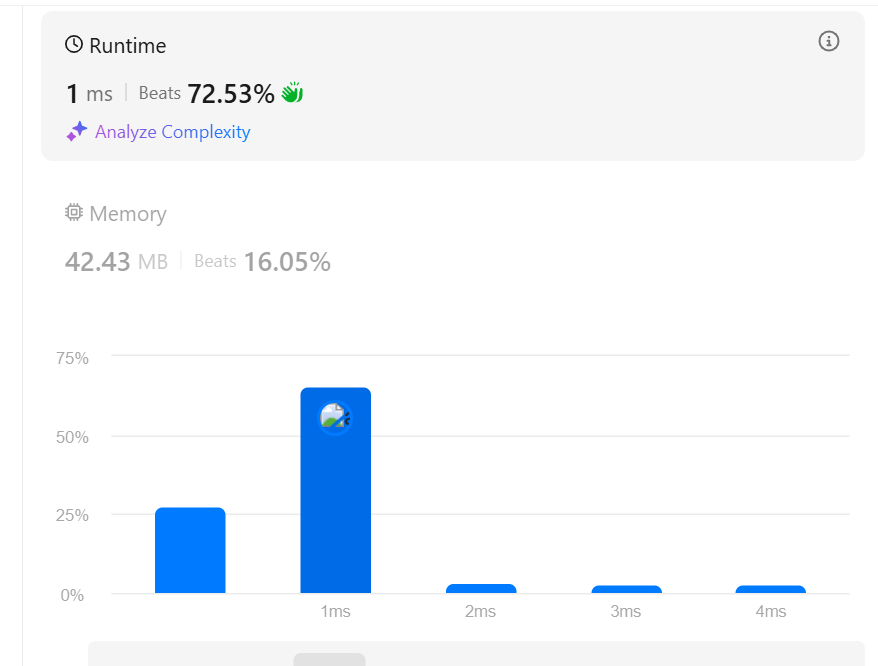
}

}

return result;

}

}



1. **Construct Binary Tree from Inorder and Postorder Traversal**

import java.util.HashMap;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) { val = x; }

}

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,

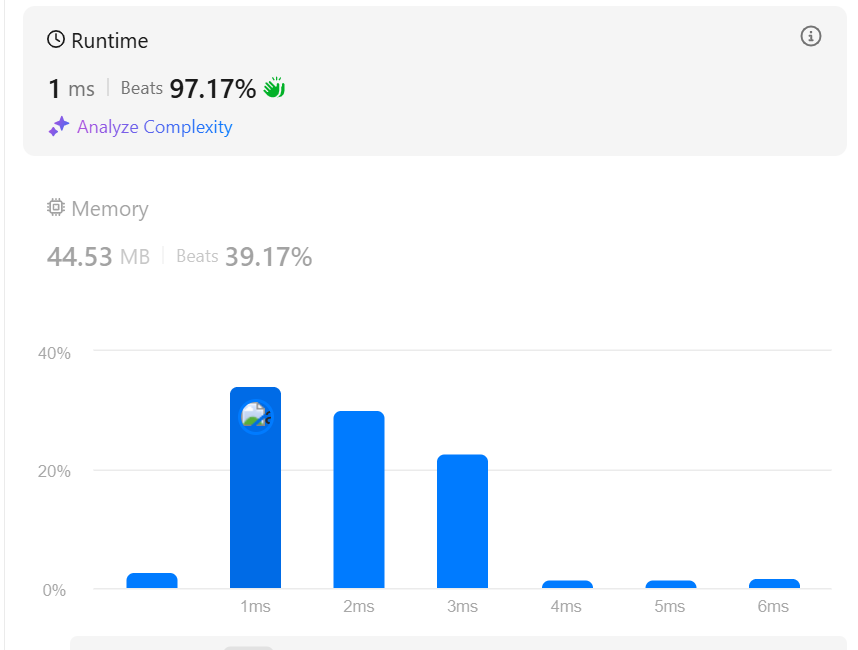
postStart + leftSubtreeSize, postEnd - 1,

rec);

return root;

}

}



1. **Find Bottom Left Tree Value**

class Solution {

public int findBottomLeftValue(TreeNode root) {

int[] result = new int[]{0, 0};

help(root, 1, result);

return result[1];

}

private void help(TreeNode node, int currD, int[] result) {

if (node == null) {

return;

}

if (currD > result[0]) {

result[0] = currD;

result[1] = node.val;

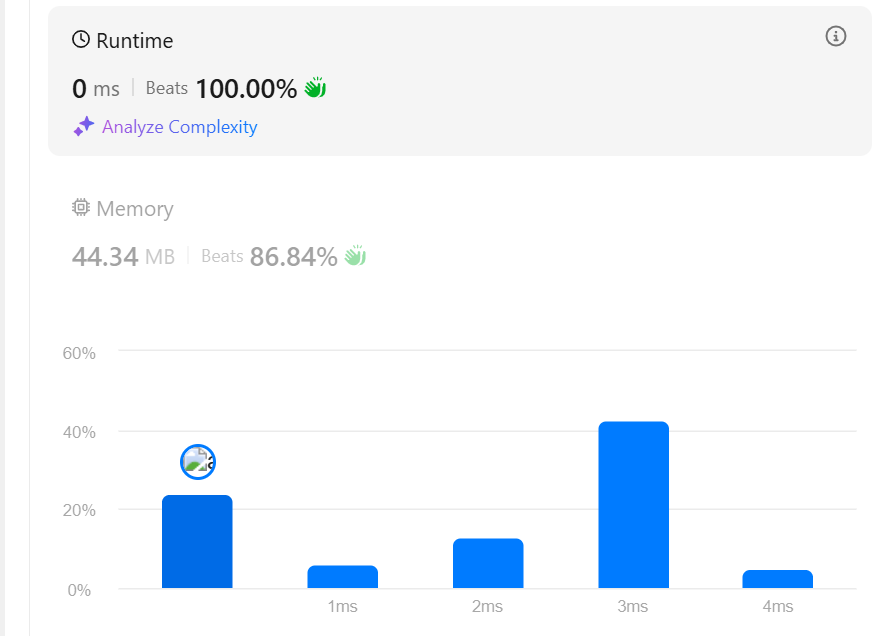
}

help(node.left, currD + 1, result);

help(node.right, currD + 1, result);

}

}



1. **Binary Tree Maximum Path Sum**

class TreeNode {

int val;

TreeNode left, right;

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

}

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));

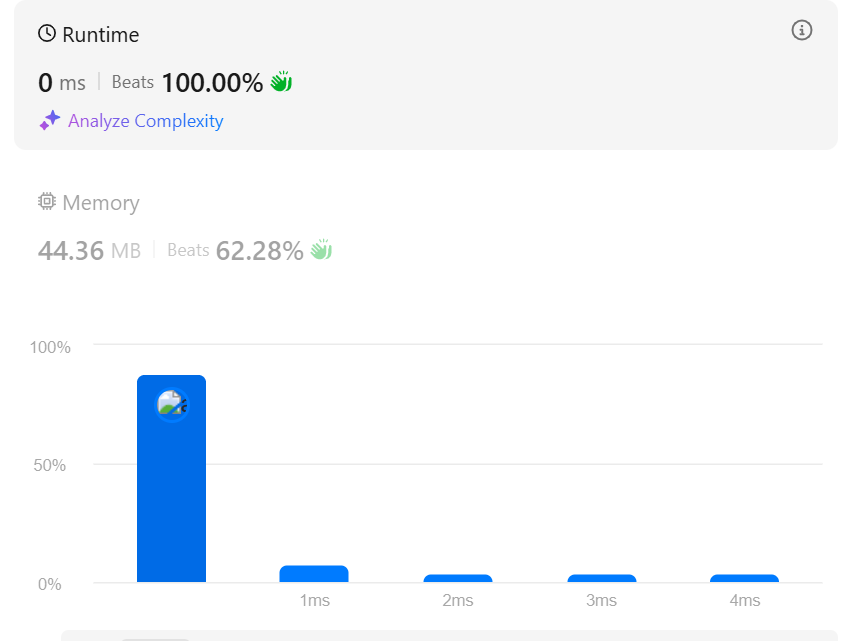
int right = Math.max(0, helper(root.right));

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

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

}

}



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

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);

}

}

