# Name: Dhruv Sharma

# UID: 22BCS13922

# Section: 612-B

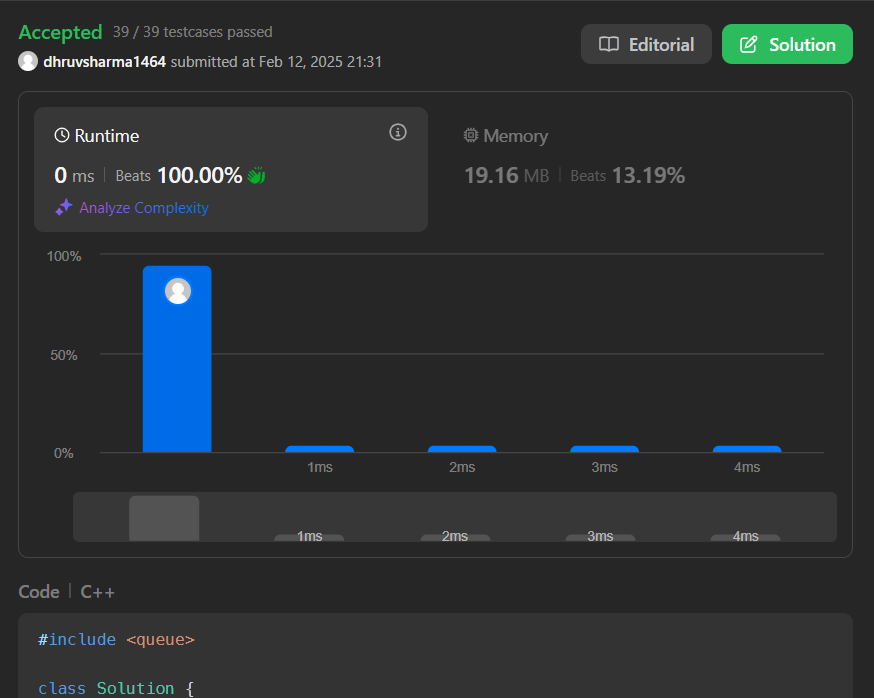
# Subject: Advanced Programming 2

# Assignment - 5

## 104. Maximum Depth of Binary Tree

class Solution {  
public:  
 int maxDepth(TreeNode\* root) {  
 if (!root) return 0;  
 return 1 + max(maxDepth(root->left), maxDepth(root->right));  
 }  
};

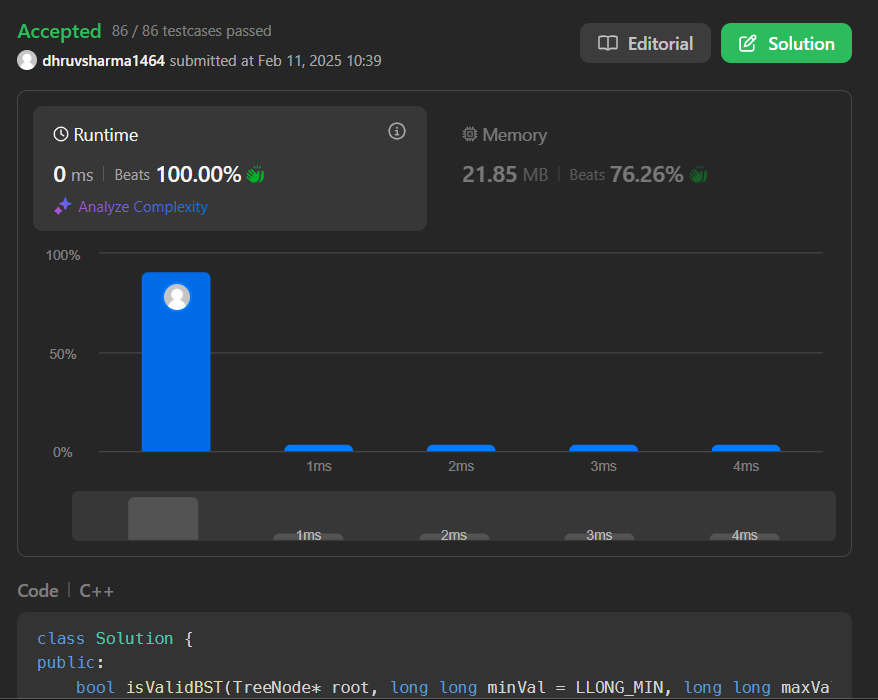
OUTUPT:



## 98. Validate Binary Search Tree

class Solution {  
public:  
 bool isValidBST(TreeNode\* root, TreeNode\* minNode = nullptr, TreeNode\* maxNode = nullptr) {  
 if (!root) return true;  
 if ((minNode && root->val <= minNode->val) || (maxNode && root->val >= maxNode->val))  
 return false;  
 return isValidBST(root->left, minNode, root) && isValidBST(root->right, root, maxNode);  
 }  
};

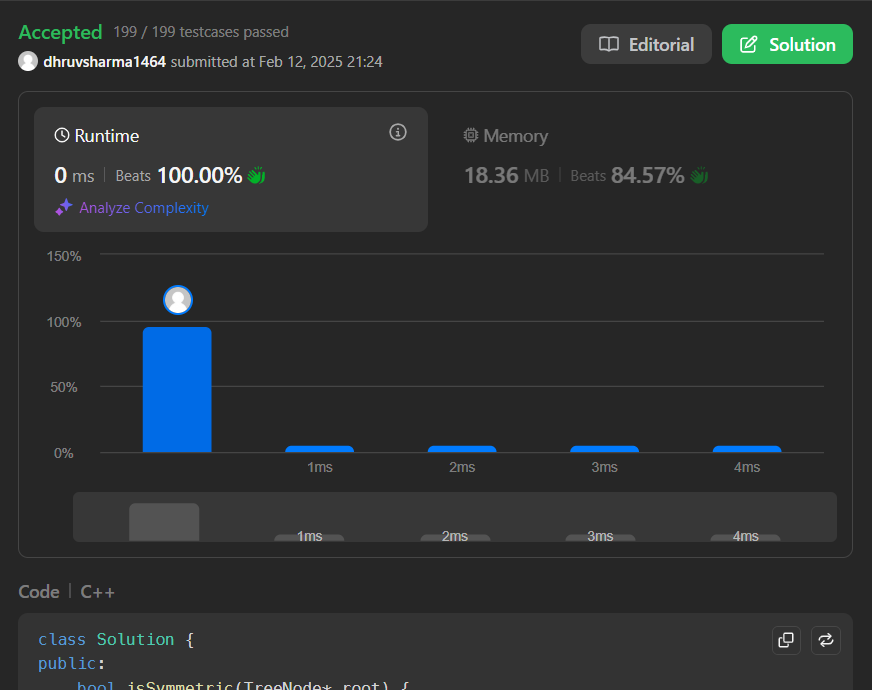
OUTPUT:



## 101. Symmetric Tree

class Solution {  
public:  
 bool isMirror(TreeNode\* t1, TreeNode\* t2) {  
 if (!t1 || !t2) return t1 == t2;  
 return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);  
 }  
   
 bool isSymmetric(TreeNode\* root) {  
 return isMirror(root, root);  
 }  
};

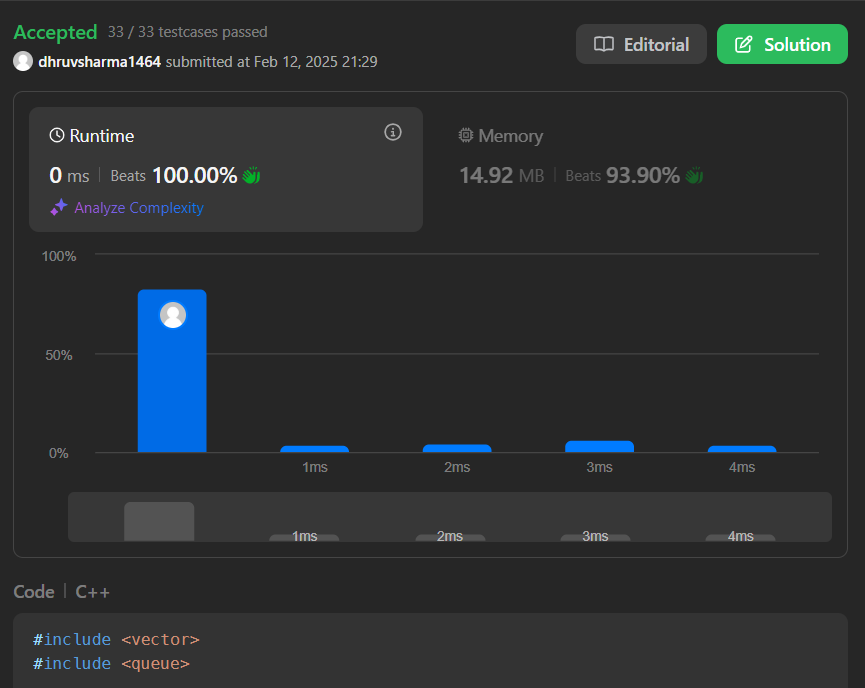
OUTPUT:



## 103. Binary Tree Zigzag Level Order Traversal

class Solution {  
public:  
 vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {  
 vector<vector<int>> result;  
 if (!root) return result;  
   
 queue<TreeNode\*> q;  
 q.push(root);  
 bool leftToRight = true;  
   
 while (!q.empty()) {  
 int size = q.size();  
 vector<int> row(size);  
 for (int i = 0; i < size; ++i) {  
 TreeNode\* node = q.front();  
 q.pop();  
   
 int index = leftToRight ? i : (size - 1 - i);  
 row[index] = node->val;  
   
 if (node->left) q.push(node->left);  
 if (node->right) q.push(node->right);  
 }  
 result.push\_back(row);  
 leftToRight = !leftToRight;  
 }  
 return result;  
 }  
};

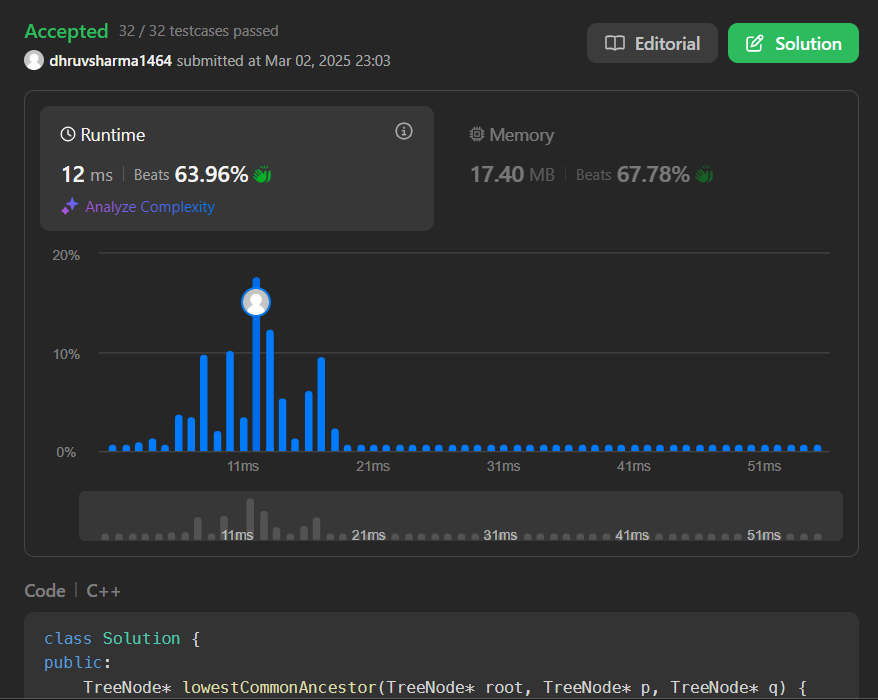
OUTPUT:



## 236. Lowest Common Ancestor of a Binary Tree

class Solution {  
public:  
 TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {  
 if (!root || root == p || root == q) return root;  
 TreeNode\* left = lowestCommonAncestor(root->left, p, q);  
 TreeNode\* right = lowestCommonAncestor(root->right, p, q);  
 return left && right ? root : left ? left : right;  
 }  
};

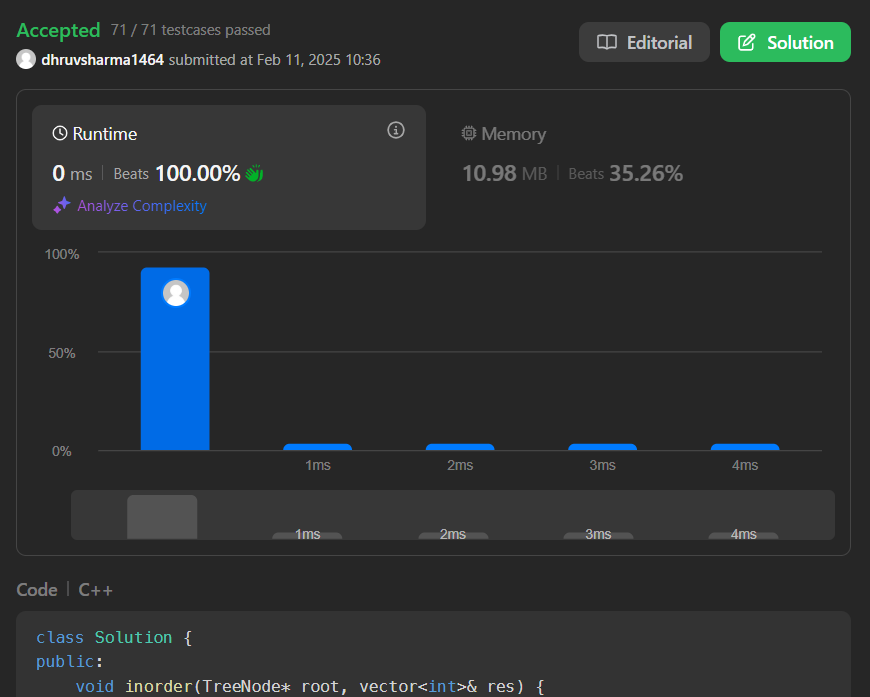
OUTPUT:



## 94. Binary Tree Inorder Traversal

class Solution {  
public:  
 vector<int> inorderTraversal(TreeNode\* root) {  
 vector<int> result;  
 stack<TreeNode\*> st;  
 TreeNode\* curr = root;  
 while (curr || !st.empty()) {  
 while (curr) {  
 st.push(curr);  
 curr = curr->left;  
 }  
 curr = st.top();  
 st.pop();  
 result.push\_back(curr->val);  
 curr = curr->right;  
 }  
 return result;  
 }  
};

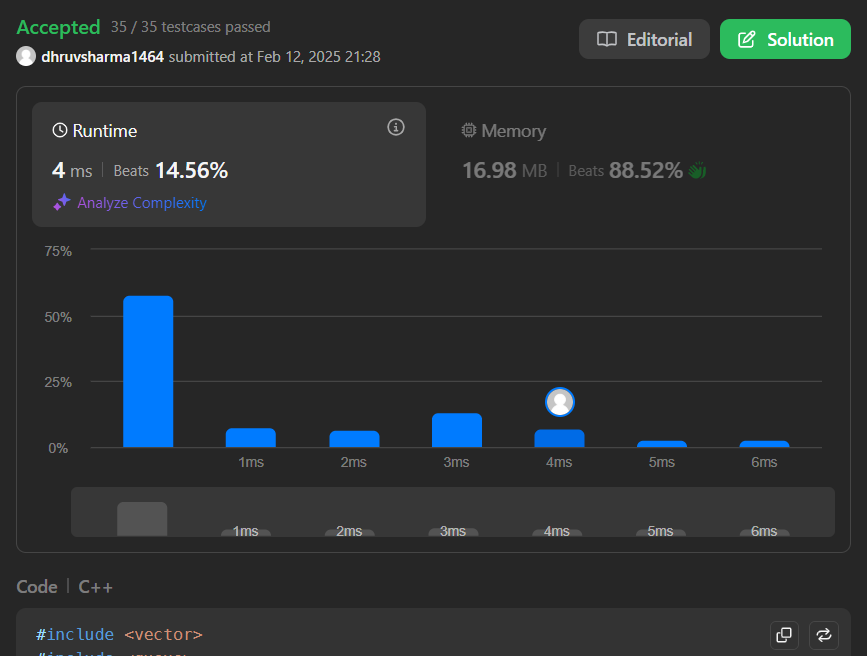
OUTPUT:



## 102. Binary Tree Level Order Traversal

class Solution {  
public:  
 vector<vector<int>> levelOrder(TreeNode\* root) {  
 vector<vector<int>> result;  
 if (!root) return result;  
   
 queue<TreeNode\*> q;  
 q.push(root);  
   
 while (!q.empty()) {  
 int size = q.size();  
 vector<int> level;  
 for (int i = 0; i < size; ++i) {  
 TreeNode\* node = q.front();  
 q.pop();  
 level.push\_back(node->val);  
   
 if (node->left) q.push(node->left);  
 if (node->right) q.push(node->right);  
 }  
 result.push\_back(level);  
 }  
 return result;  
 }  
};

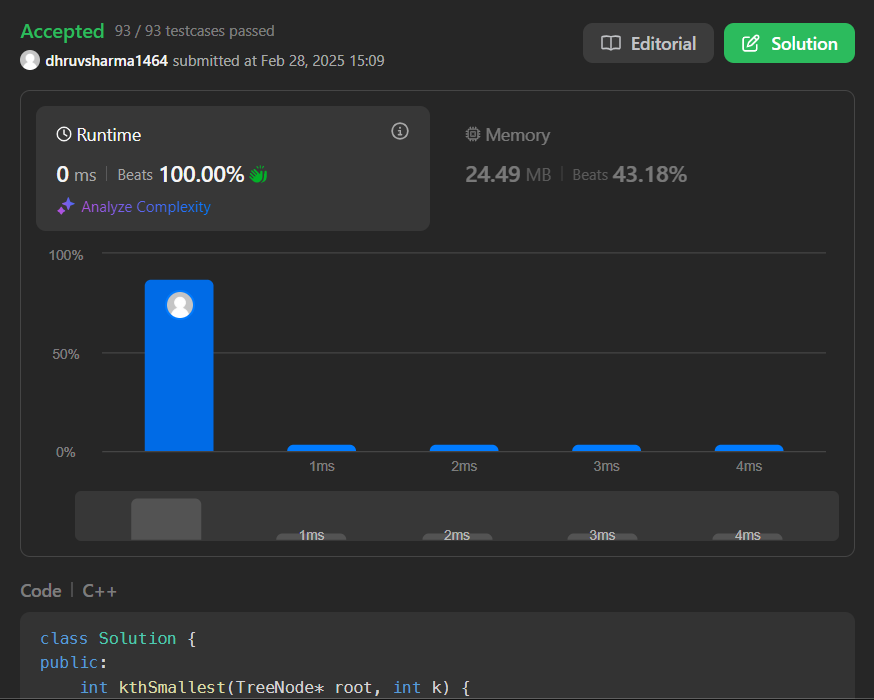
OUTPUT:



## 230. Kth Smallest Element in a BST

class Solution {  
public:  
 int kthSmallest(TreeNode\* root, int k) {  
 stack<TreeNode\*> st;  
 TreeNode\* curr = root;  
 while (curr || !st.empty()) {  
 while (curr) {  
 st.push(curr);  
 curr = curr->left;  
 }  
 curr = st.top();  
 st.pop();  
 if (--k == 0) return curr->val;  
 curr = curr->right;  
 }  
 return -1;  
 }  
};

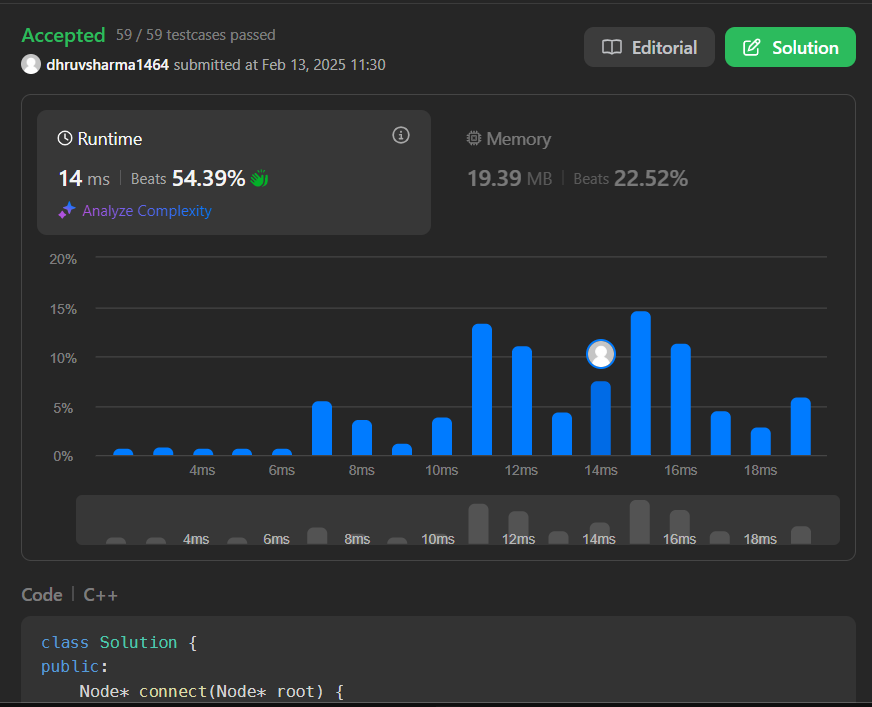
OUTPUT:



## 116. Populating Next Right Pointers in Each Node

class Solution {  
public:  
 Node\* connect(Node\* root) {  
 if (!root) return nullptr;  
   
 queue<Node\*> q;  
 q.push(root);  
   
 while (!q.empty()) {  
 int size = q.size();  
 Node\* prev = nullptr;  
   
 for (int i = 0; i < size; ++i) {  
 Node\* node = q.front();  
 q.pop();  
   
 if (prev) prev->next = node;  
 prev = node;  
   
 if (node->left) q.push(node->left);  
 if (node->right) q.push(node->right);  
 }  
 }  
 return root;  
 }  
};

OUTPUT:



## 404. Sum of Left Leaves

class Solution {  
public:  
 int sumOfLeftLeaves(TreeNode\* root) {  
 if (!root) return 0;  
   
 int sum = 0;  
 if (root->left && !root->left->left && !root->left->right)  
 sum += root->left->val;  
   
 return sum + sumOfLeftLeaves(root->left) + sumOfLeftLeaves(root->right);  
 }  
};

OUTPUT:

