** **

**UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**(BE-CSE/IT-5th Sem)**

****

**Subject Name: AP LAB-II**

**Subject Code: 22CSP-351**

**Submitted to: Submitted by:**

Faculty name Name: Ankit Kharb

Bhumika UID: 22BCS16964

Section: 22BCS-IOT-614

Group: B

**Question 1.** Maximum Depth Of binary tree

**CODE:**

class Solution {

public:

    int maxDepth(TreeNode\* root) {

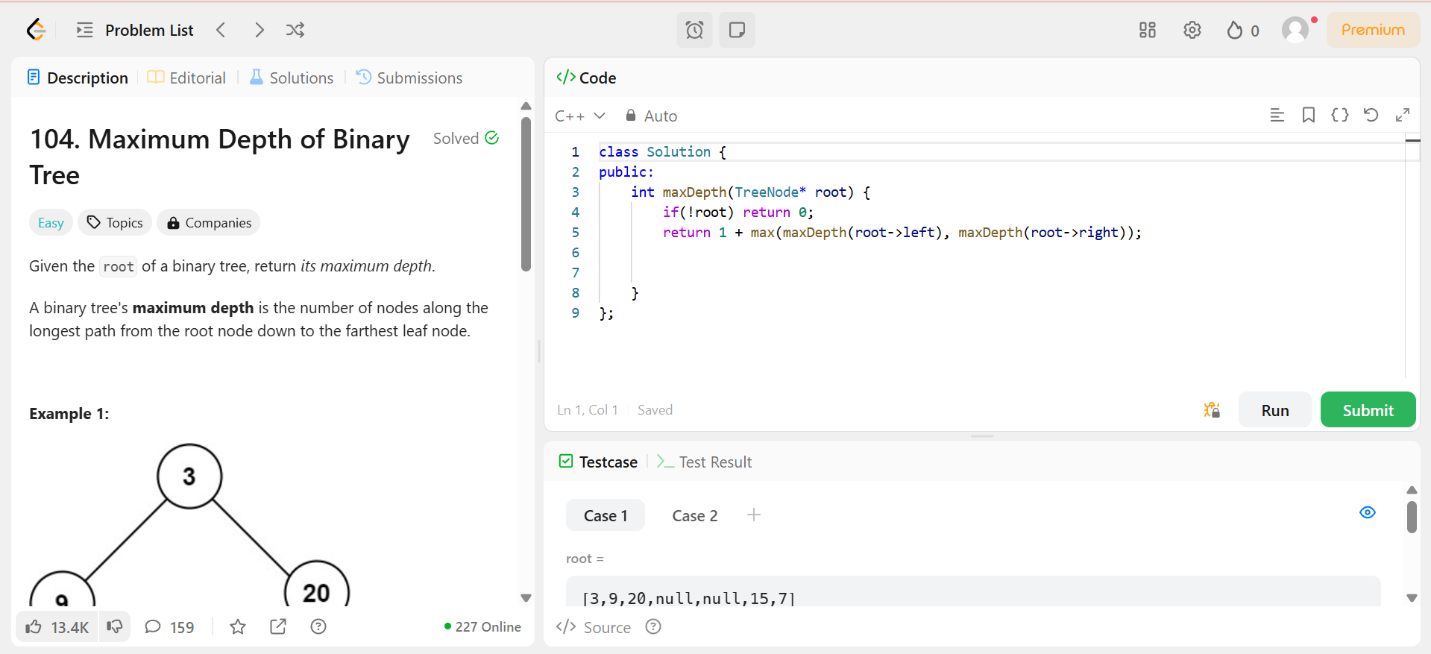
        if(!root) return 0;

        return 1 + max(maxDepth(root->left), maxDepth(root->right));

    }

};

**OUTPUT:**

****

**Question 2:** Validate Binary Search Tree

**CODE:**

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

        return validate(root, LONG\_MIN, LONG\_MAX);

    }

private:

    bool validate(TreeNode\* node, long minVal, long maxVal) {

        if (!node) return true;

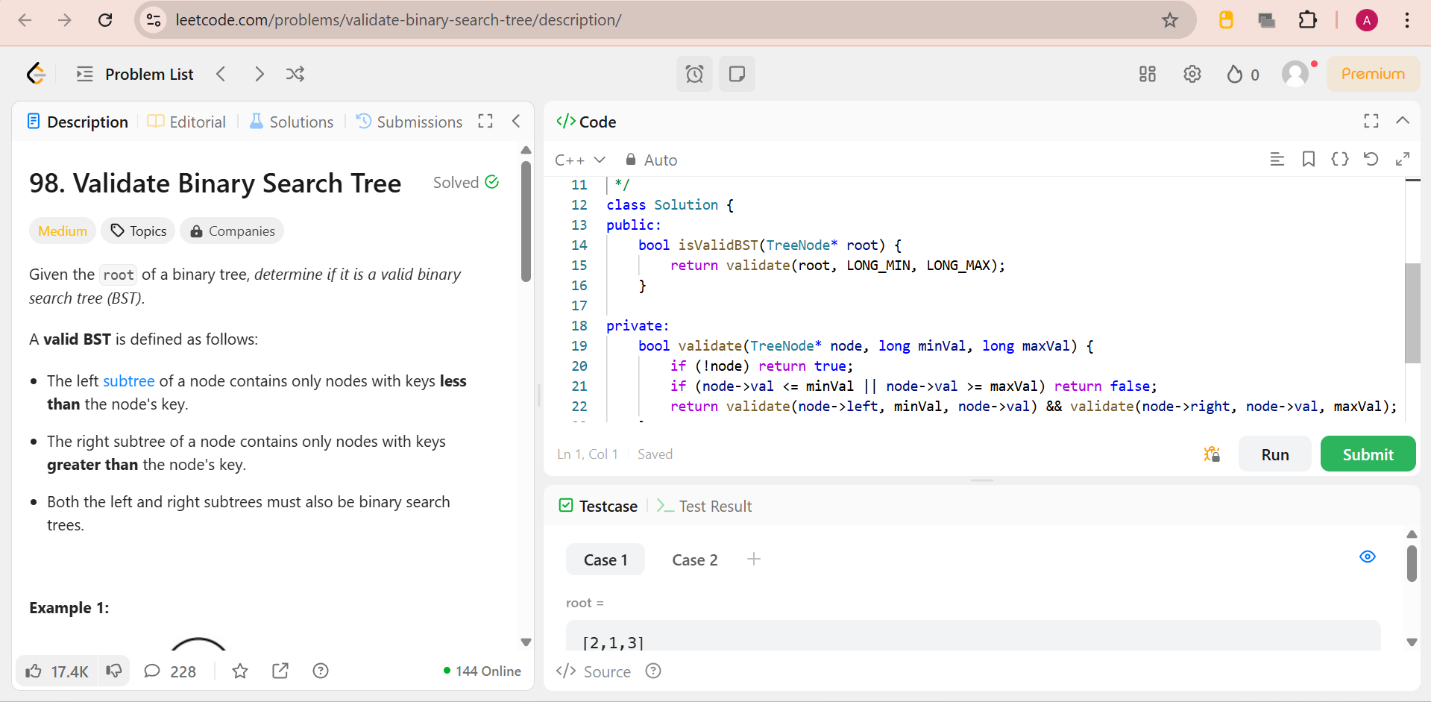
        if (node->val <= minVal || node->val >= maxVal) return false;

        return validate(node->left, minVal, node->val) && validate(node->right, node->val, maxVal);

    }

};

**OUTPUT:**



**Question 3:** Symmetric Tree

**CODE:**

class Solution {

public:

    bool isMirror(TreeNode\* left, TreeNode\* right) {

        if (!left && !right) return true;  // Both nodes are NULL

        if (!left || !right) return false; // One node is NULL, the other isn't

        return (left->val == right->val)   // Values should be the same

            && isMirror(left->left, right->right)   // Check outer children

            && isMirror(left->right, right->left);  // Check inner children

    }

    bool isSymmetric(TreeNode\* root) {

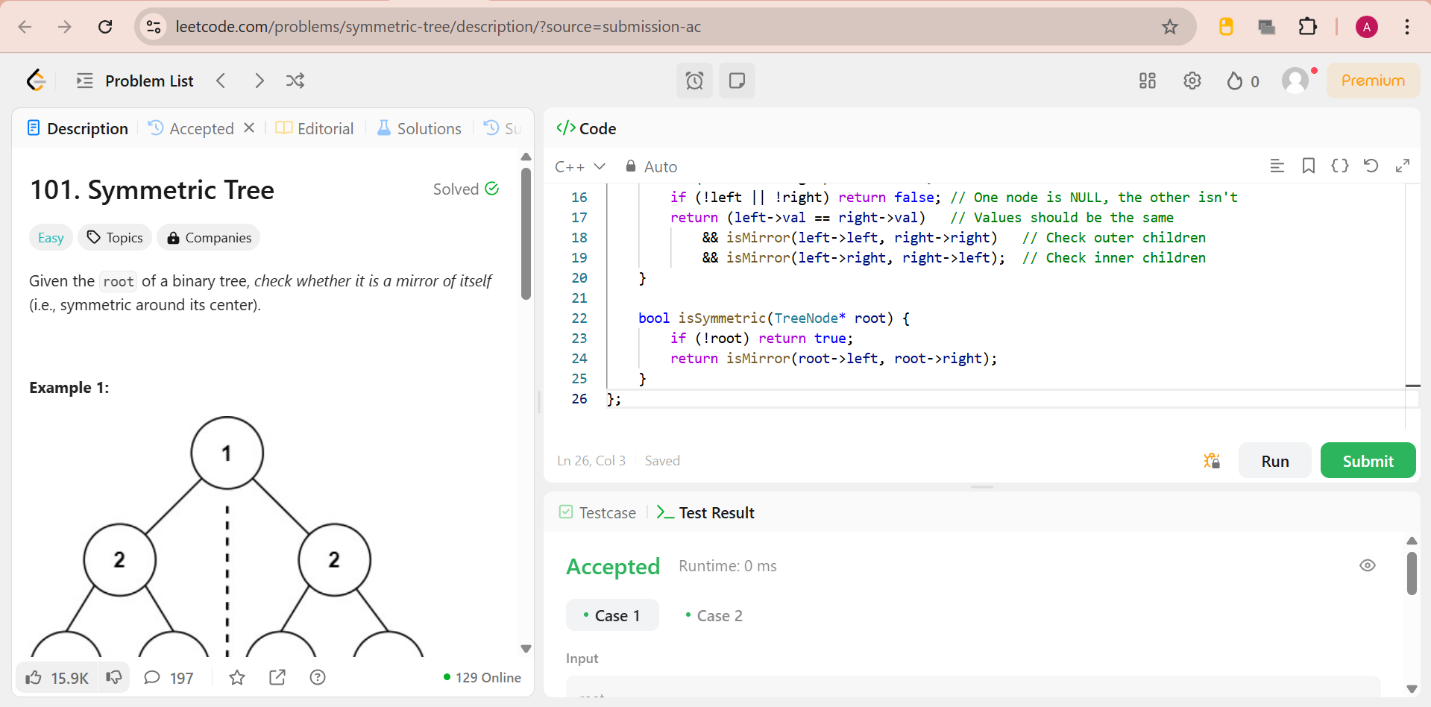
        if (!root) return true;

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

    }

};

**OUTPUT:**



**Question 4:** Binary Tree Zigzag Order Traversal

**CODE:**

class Solution {

public:

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

        vector<vector<int>> res;

        if (root == NULL) {

            return res;

        }

        queue<TreeNode\*> q;

        q.push(root);

        bool leftToRight = true;

        while (!q.empty()) {

            int n = q.size();

            deque<int> levelNodes;

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

                TreeNode\* temp = q.front();

                q.pop();

                if (leftToRight) {

                    levelNodes.push\_back(temp->val);

                } else {

                    levelNodes.push\_front(temp->val);

                }

                if (temp->left) {

                    q.push(temp->left);

                }

                if (temp->right) {

                    q.push(temp->right);

                }

            }

            res.push\_back(vector<int>(levelNodes.begin(), levelNodes.end()));

            leftToRight = !leftToRight; // Toggle direction

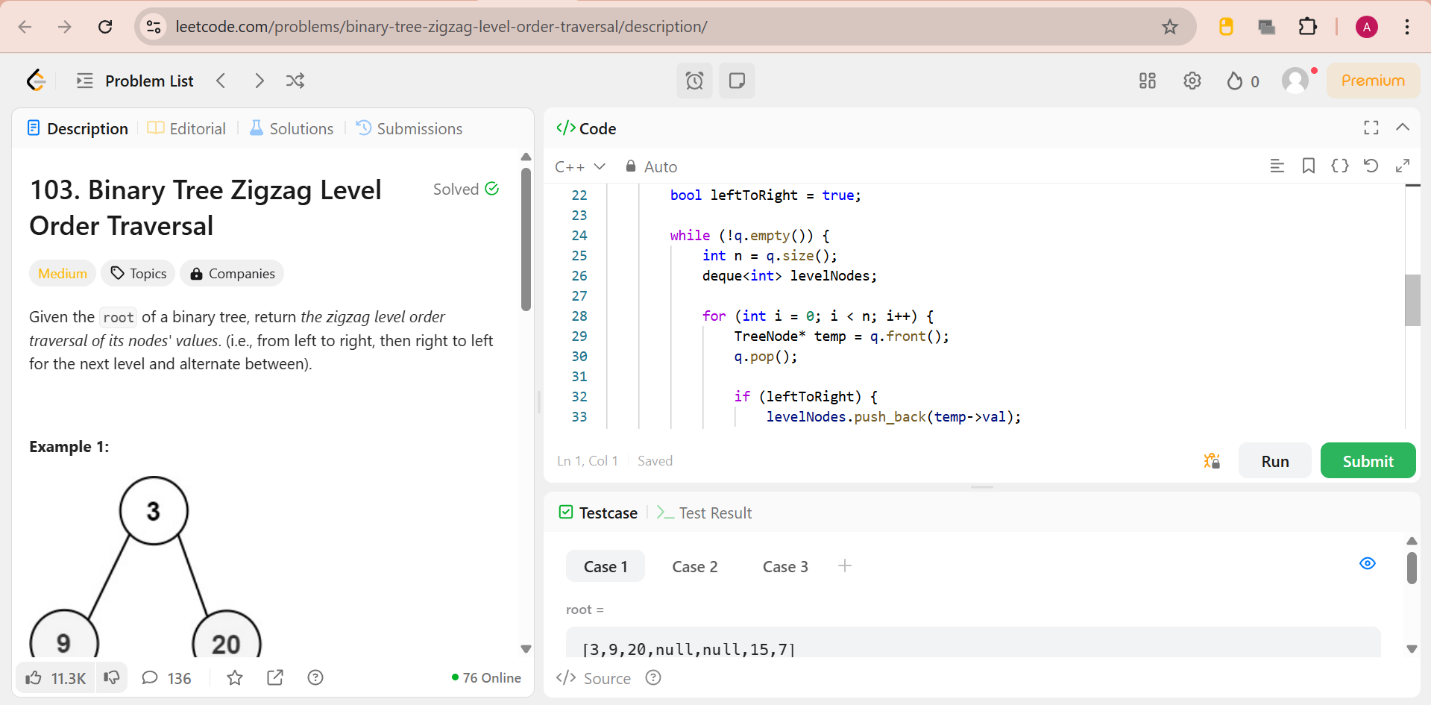
        }

        return res;

    }

};

**OUTPUT:**

****

**Question 5:** Lowest Common Ancestor of binary tree

**CODE:**

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if (!root || root == p || root == q) return root;  // Base case

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

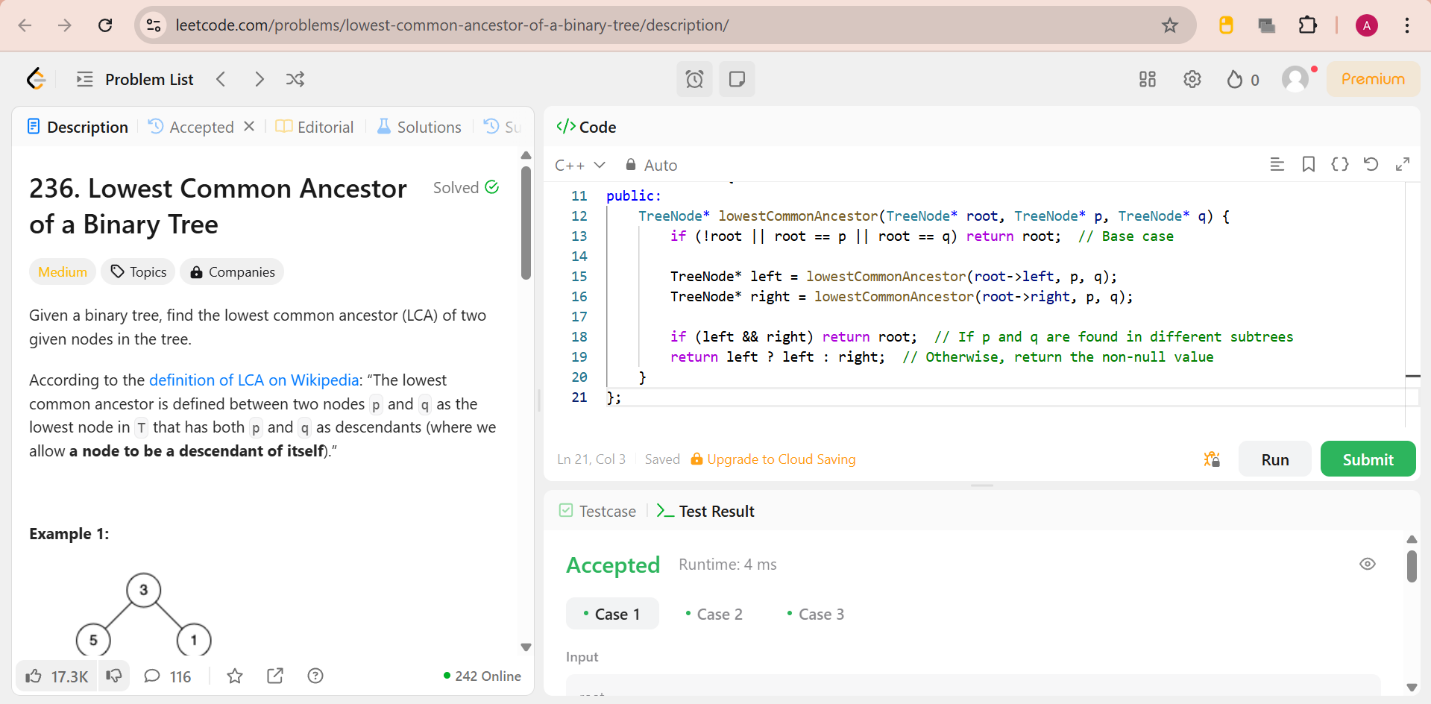
        if (left && right) return root;  // If p and q are found in different subtrees

        return left ? left : right;  // Otherwise, return the non-null value

    }

};

**OUTPUT:**

****

**Question 6:** Binary tree Inorder traversal

**CODE:**

class Solution {

public:

void inorder(TreeNode\* root, vector<int>& result) {

if (!root) return;

inorder(root->left, result); // Visit left subtree

result.push\_back(root->val); // Visit root

inorder(root->right, result); // Visit right subtree

}

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

vector<int> result;

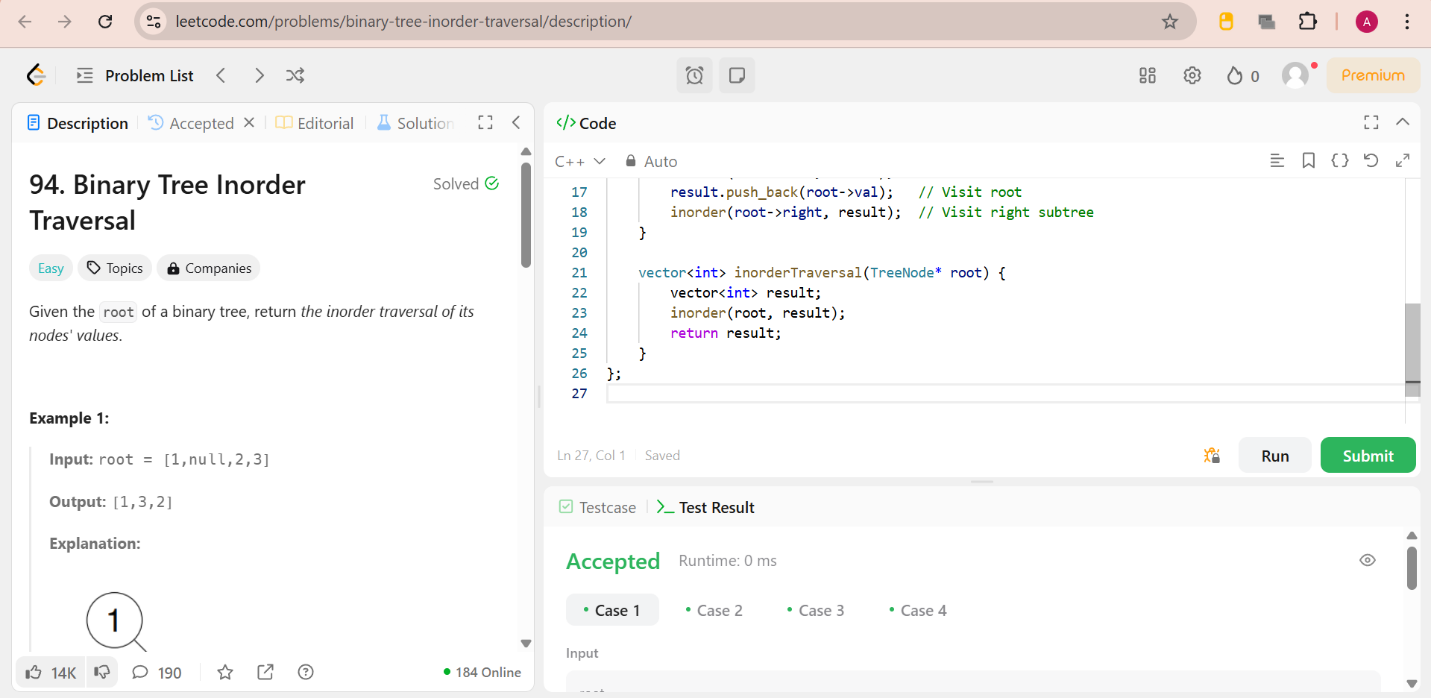
inorder(root, result);

return result;

}

};

**OUTPUT:**

****

**Question 7:** Binary Tree Level Order traversal

**CODE:**

class Solution {

public:

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

        vector<vector<int>>res;

        if(root==NULL)

        {

            return res;

        }

        queue<TreeNode\*>q;

        q.push(root);

        while(!q.empty())

        {

            int n = q.size();

            vector<int> ans;

            while(n--)

            {

                TreeNode\* temp = q.front();

                q.pop();

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

                if(temp->left)

                {

                    q.push(temp->left);

                }

                if(temp->right)

                {

                    q.push(temp->right);

                }

            }

            res.push\_back(ans);

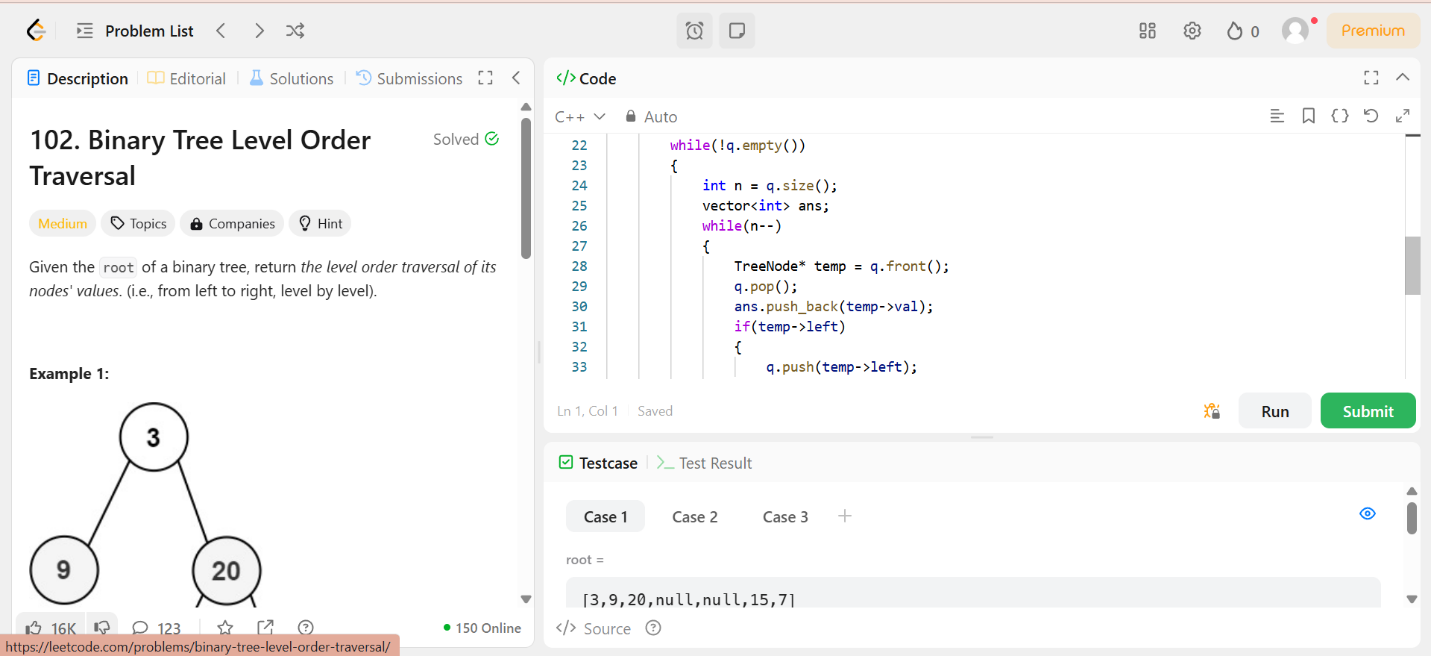
        }

        return res;

    }

};

**OUTPUT:**



**Question 8:** Kth Smallest element in a BST

**CODE:**

class Solution {

public:

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

        stack<TreeNode\*> st;

        TreeNode\* curr = root;

        int count = 0;

        while (curr || !st.empty()) {

            while (curr) {  // Reach the leftmost node

                st.push(curr);

                curr = curr->left;

            }

            curr = st.top();  // Process node

            st.pop();

            count++;

            if (count == k) return curr->val;  // Found kth smallest

            curr = curr->right;  // Move to the right subtree

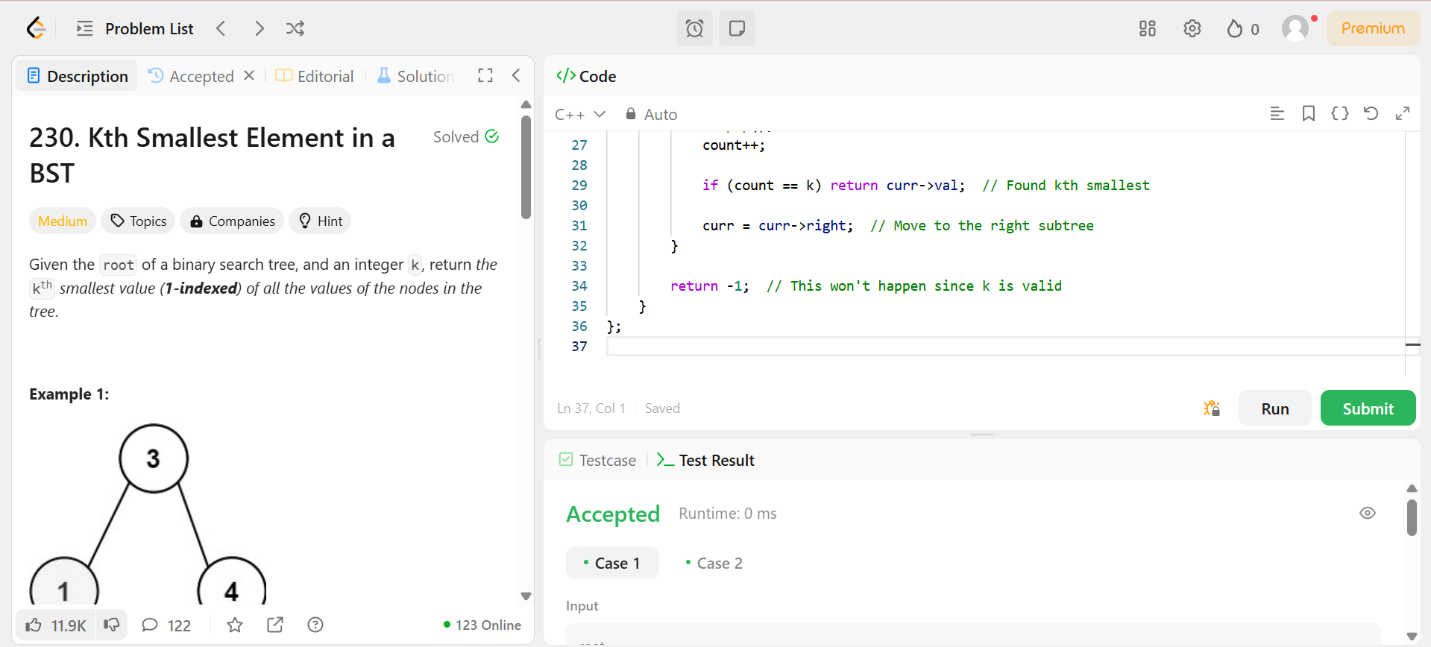
        }

        return -1;  // This won't happen since k is valid

    }

};

**OUTPUT:**

****