**ADVANCE PROGRAMMING LAB-2**

**ASSIGNMENT-3**

**NAME: MOHIT KUMAR**

**UID:22BCS10208**

94.

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);

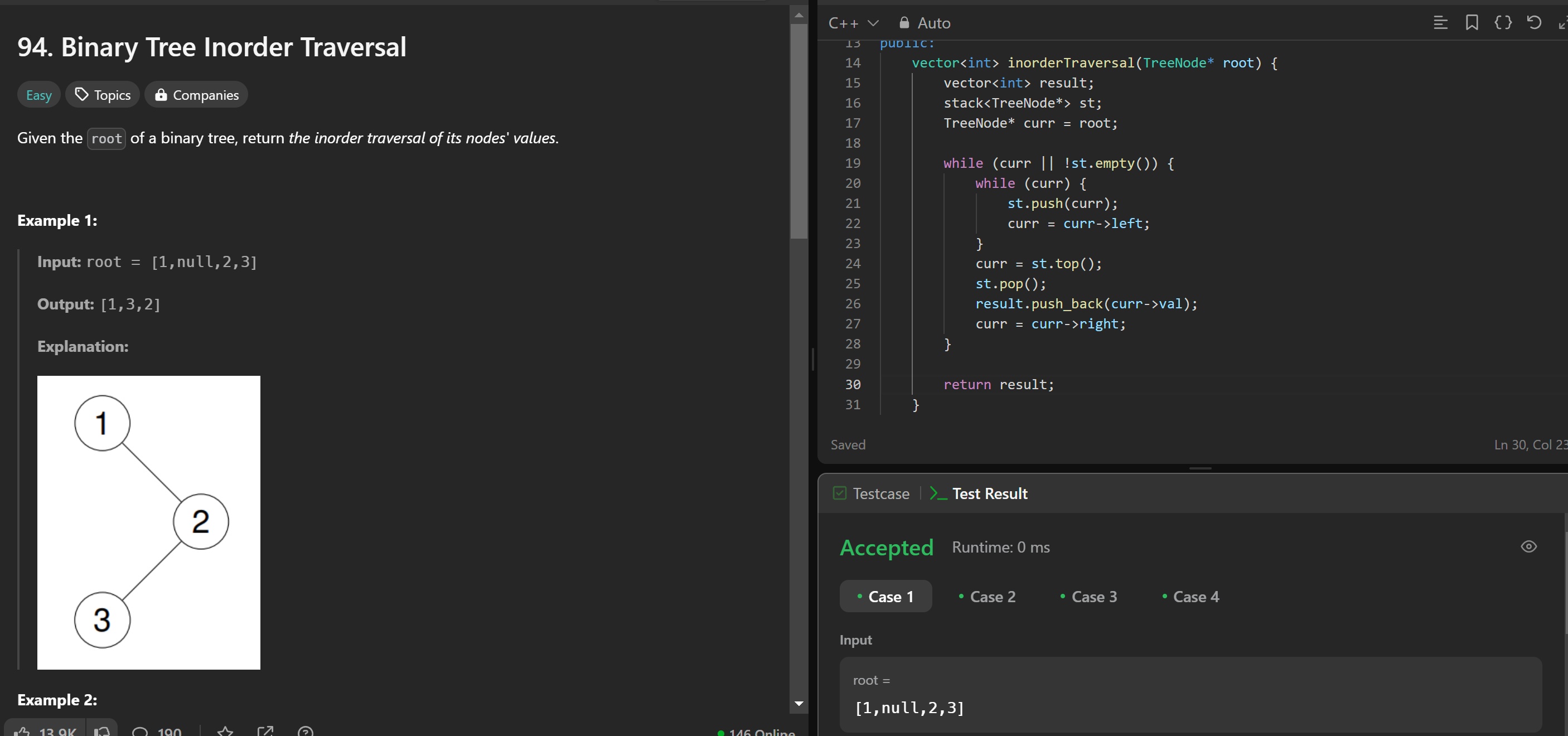
curr = curr->right;

}

return result;

}

};



101

class Solution {

public:

bool isSymmetric(TreeNode\* root) {

return isMirror(root, root);

}

bool isMirror(TreeNode\* t1, TreeNode\* t2) {

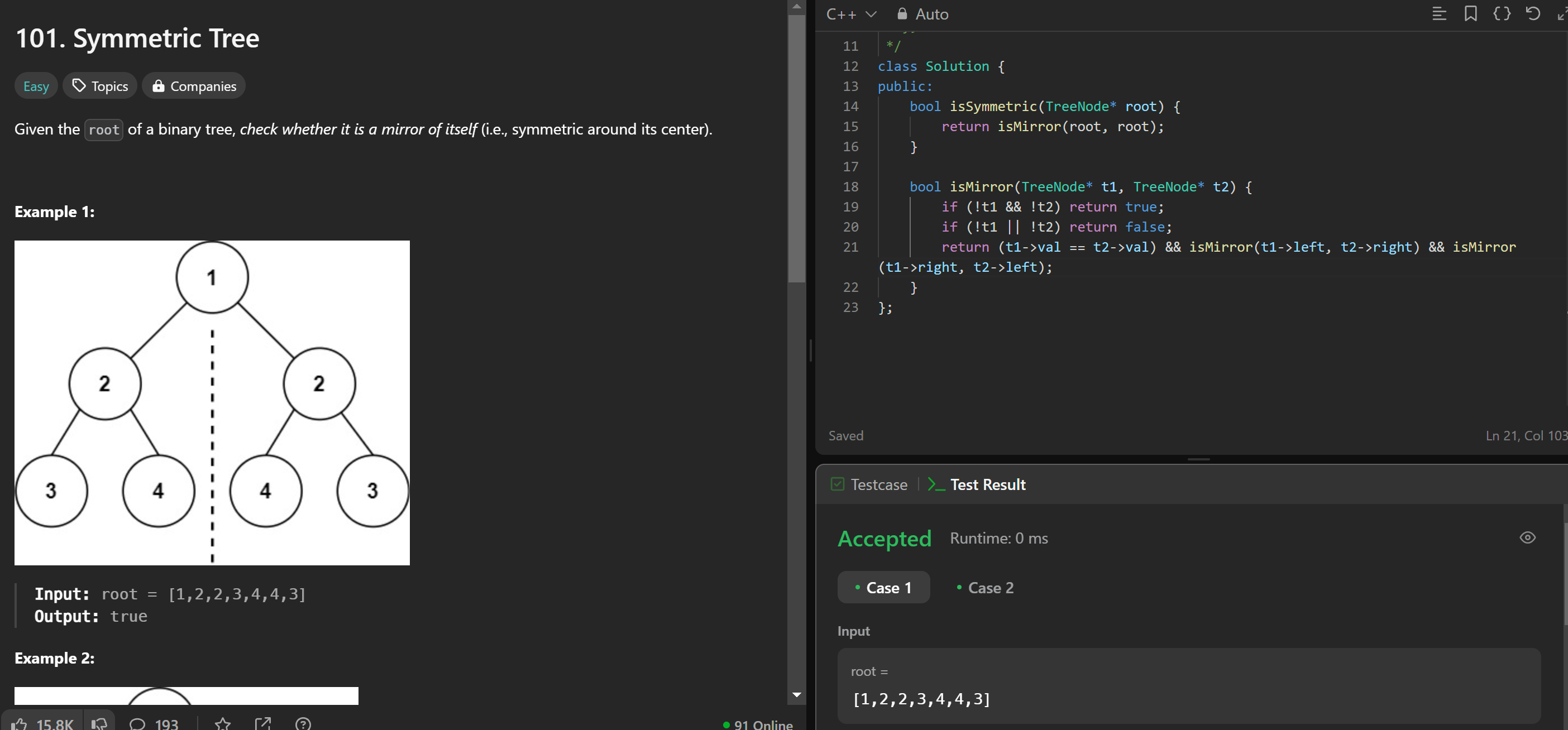
if (!t1 && !t2) return true;

if (!t1 || !t2) return false;

return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);

}

};



104

class Solution {

public:

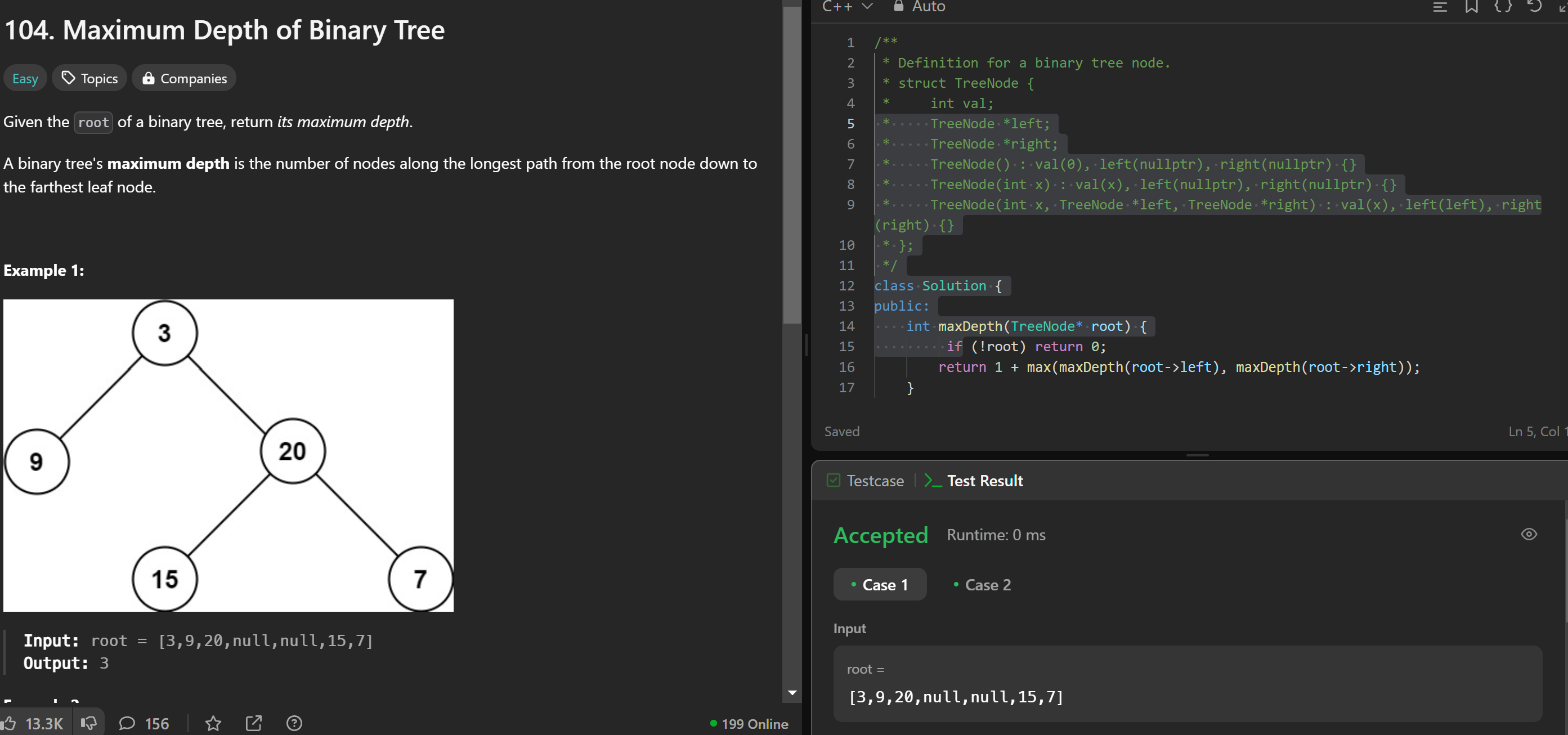
int maxDepth(TreeNode\* root) {

if (!root) return 0;

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

}

};



98.

class Solution {

public:

bool isValidBST(TreeNode\* root) {

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

}

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

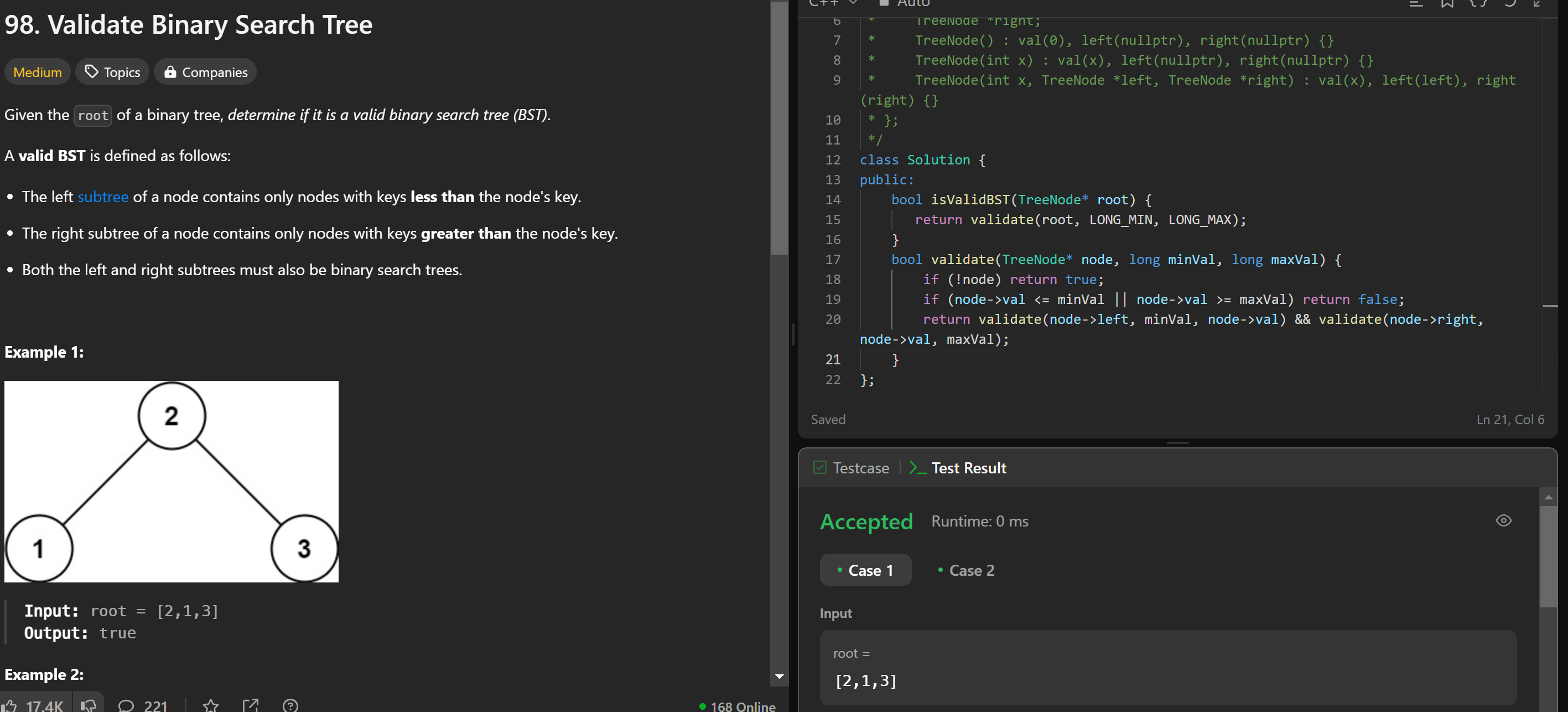
if (!node) return true;

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

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

}

};



230.

class Solution {

public:

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

stack<TreeNode\*> s;

while (true) {

while (root) {

s.push(root);

root = root->left;

}

root = s.top();

s.pop();

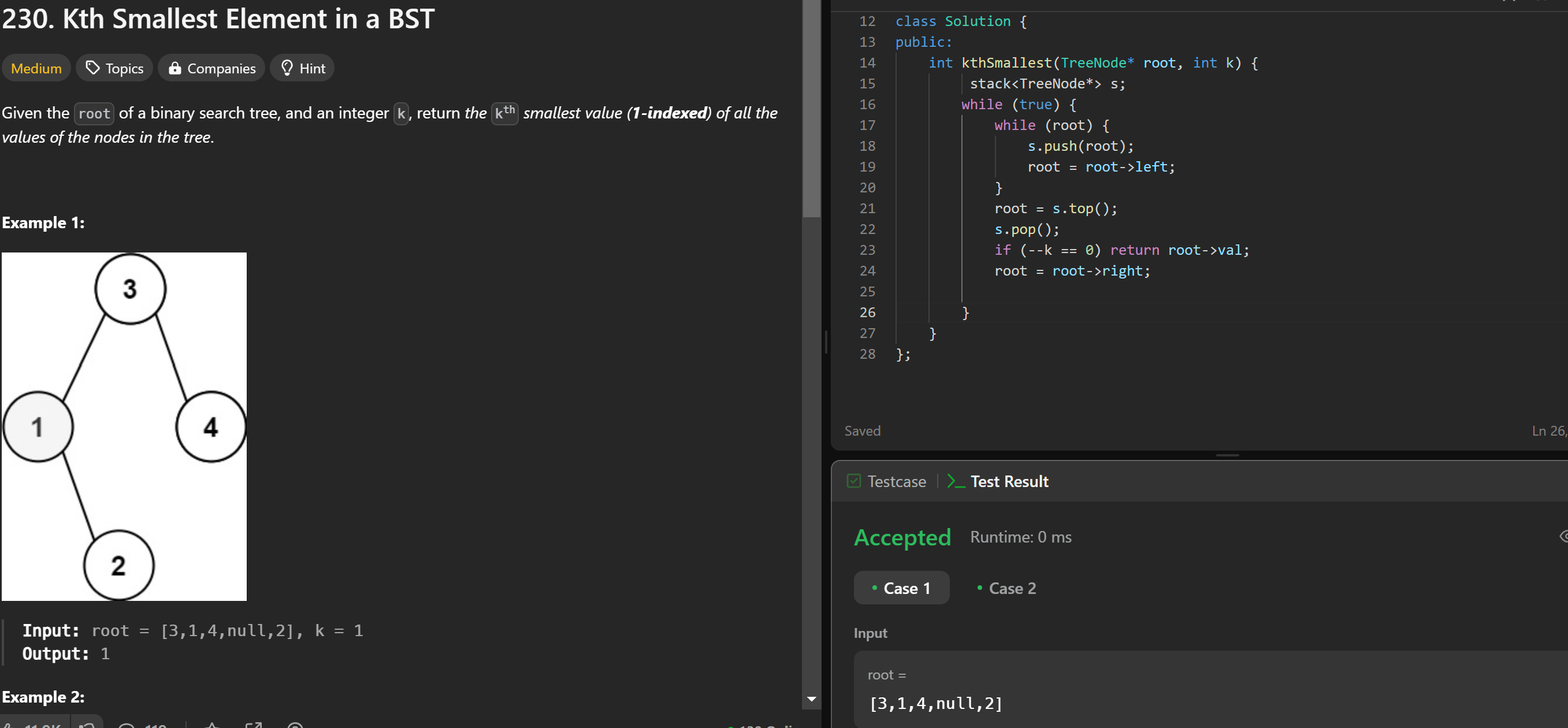
if (--k == 0) return root->val;

root = root->right;

}

}

};



102.

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);

}

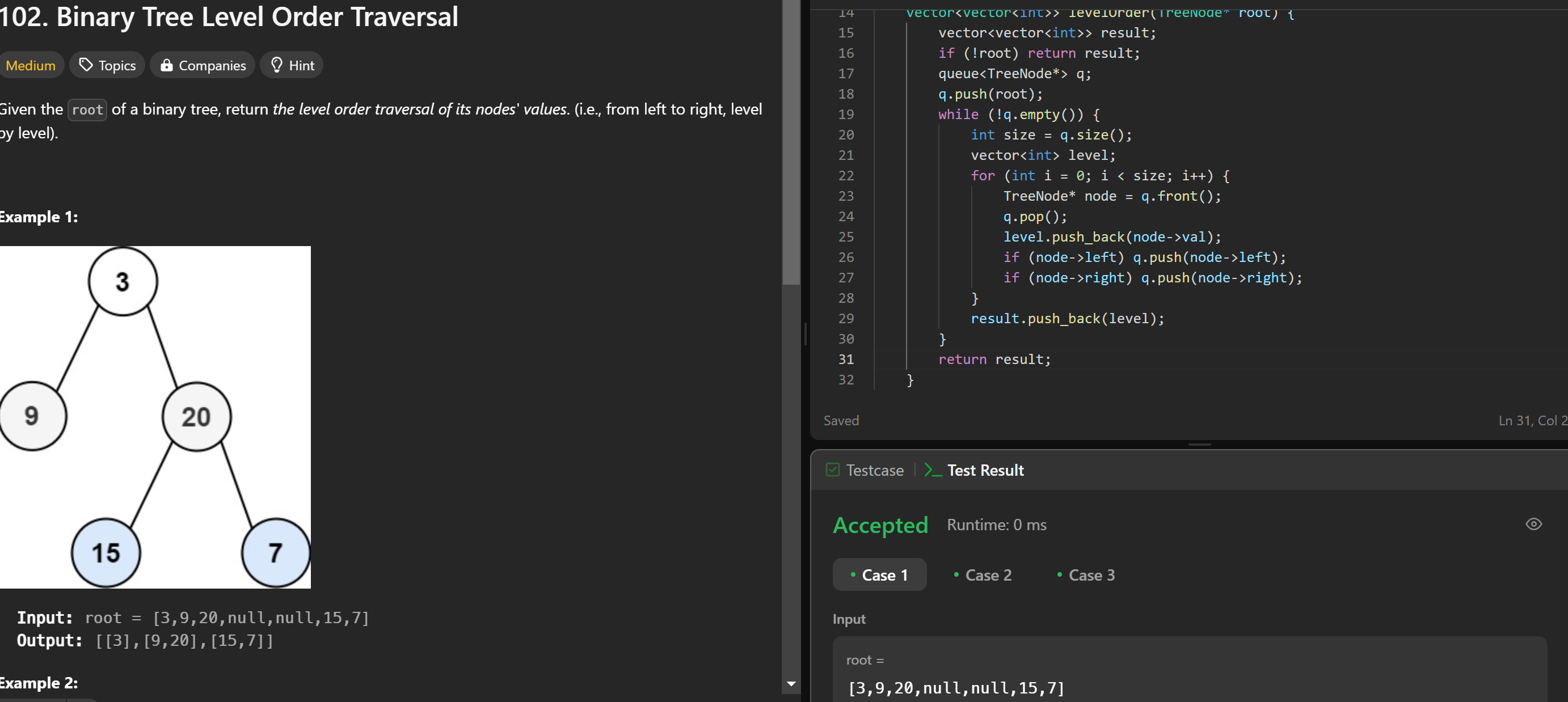
result.push\_back(level);

}

return result;

}

};



107.

class Solution {

public:

vector<vector<int>> levelOrderBottom(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);

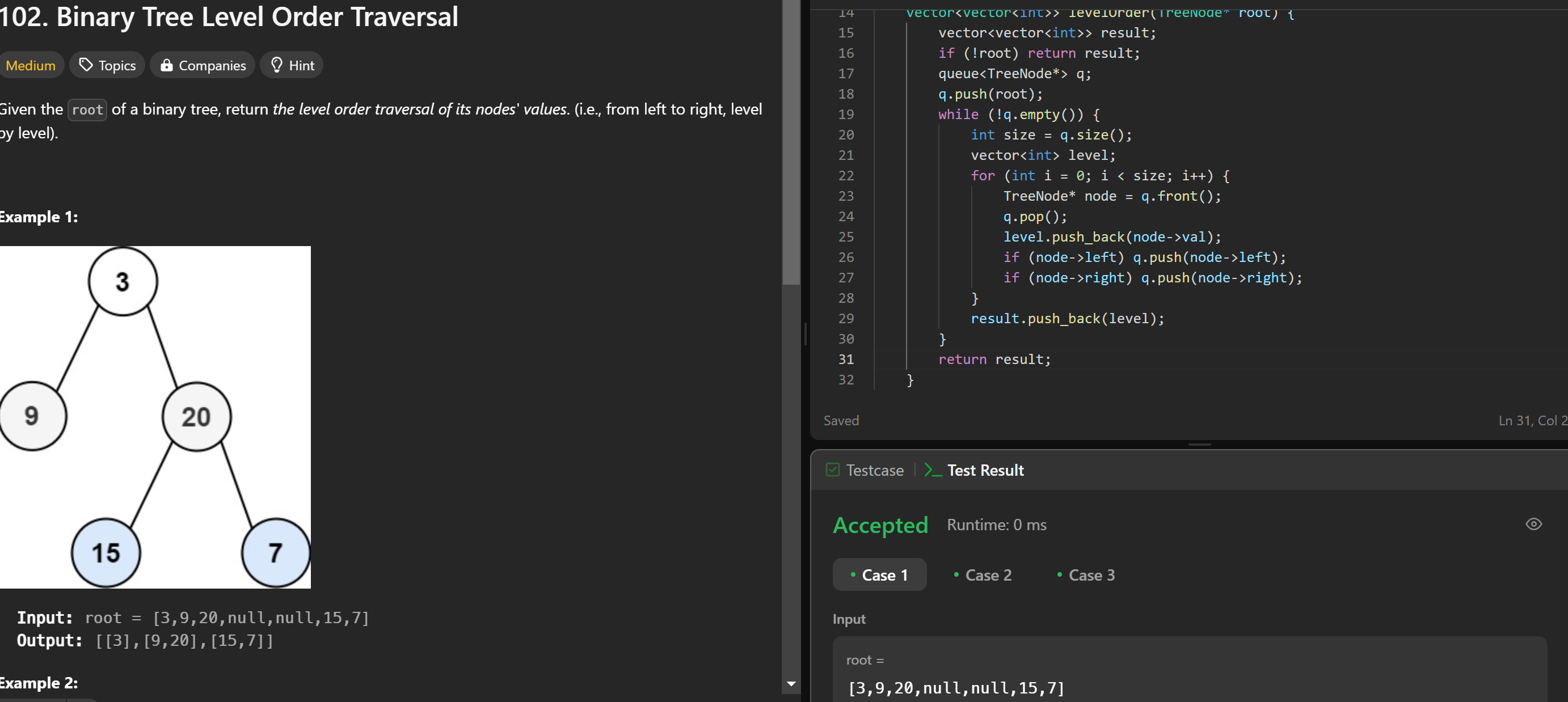
}

reverse(result.begin(), result.end());

return result;

}

};



103.

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> level(size);

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

TreeNode\* node = q.front();

q.pop();

int index = leftToRight ? i : (size - 1 - i);

level[index] = node->val;

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

}

result.push\_back(level);

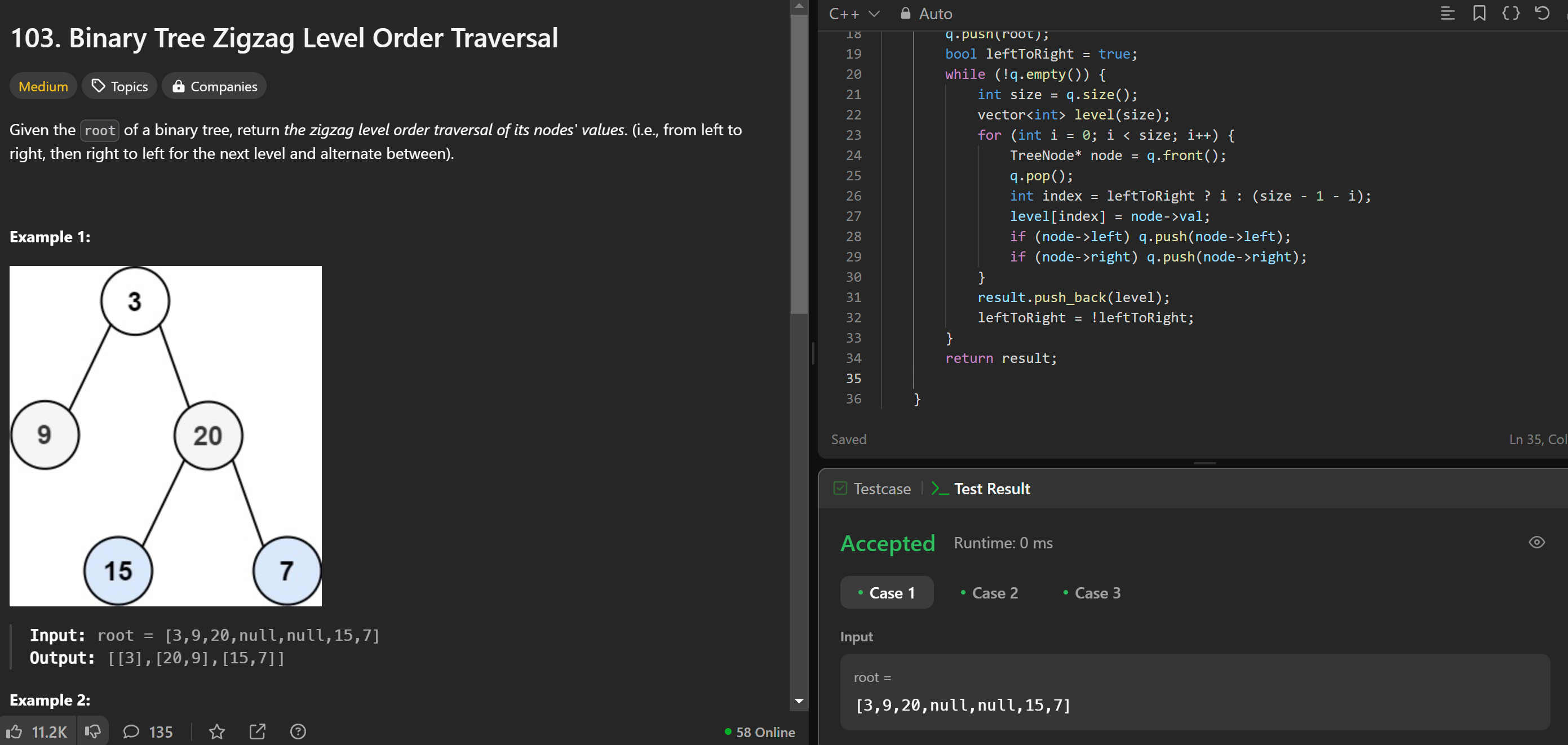
leftToRight = !leftToRight;

}

return result;

}

};



199.

class Solution {

public:

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

vector<int> result;

if (!root) return result;

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

int size = q.size();

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

TreeNode\* node = q.front();

q.pop();

if (i == size - 1) result.push\_back(node->val);

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

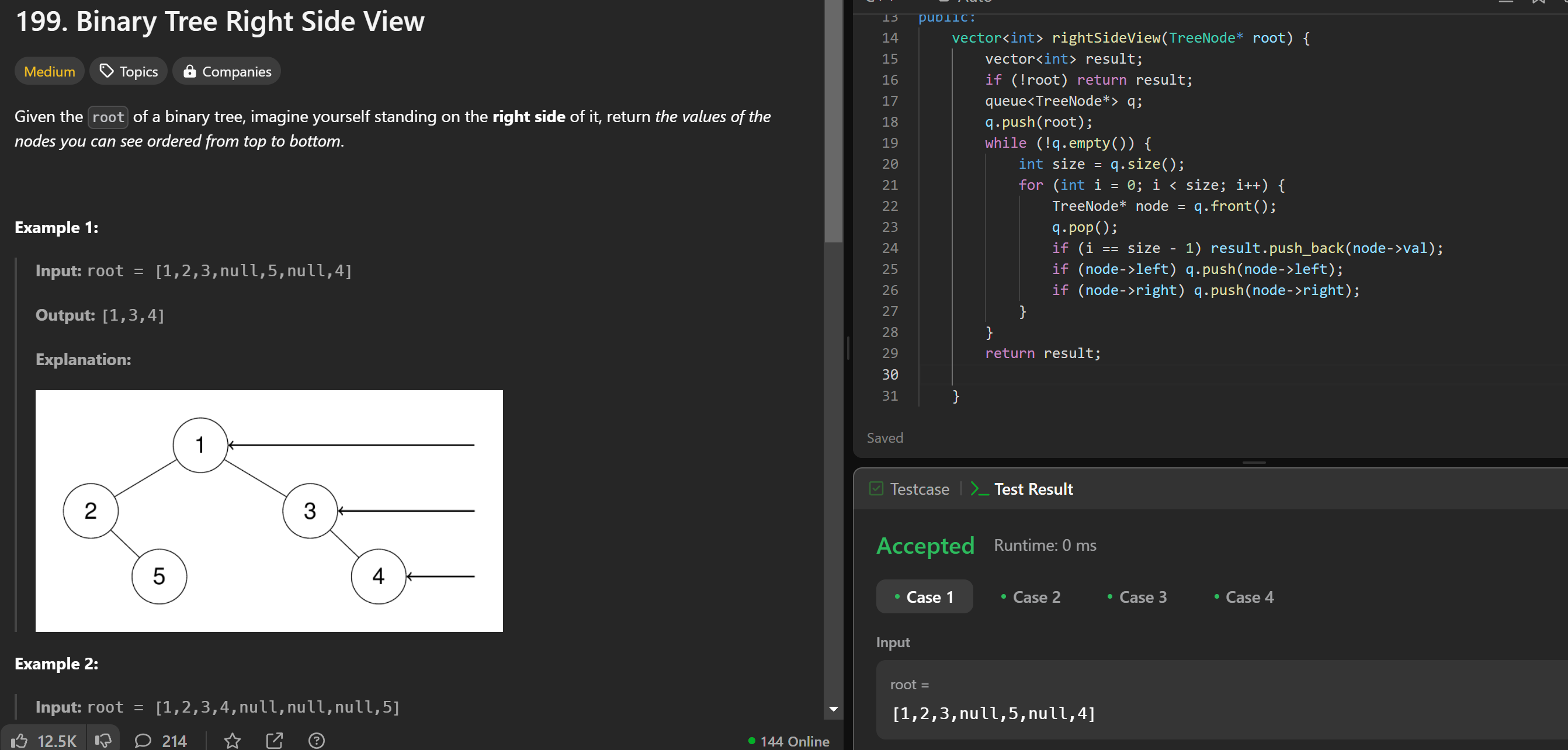
}

}

return result;

}

};



106.

class Solution {

public:

TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

unordered\_map<int, int> inorderIndex;

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

inorderIndex[inorder[i]] = i;

}

int postIdx = postorder.size() - 1;

return build(inorder, postorder, inorderIndex, postIdx, 0, inorder.size() - 1);

}

TreeNode\* build(vector<int>& inorder, vector<int>& postorder, unordered\_map<int, int>& inorderIndex, int& postIdx, int left, int right) {

if (left > right) return nullptr;

int rootVal = postorder[postIdx--];

TreeNode\* root = new TreeNode(rootVal);

int rootIdx = inorderIndex[rootVal];

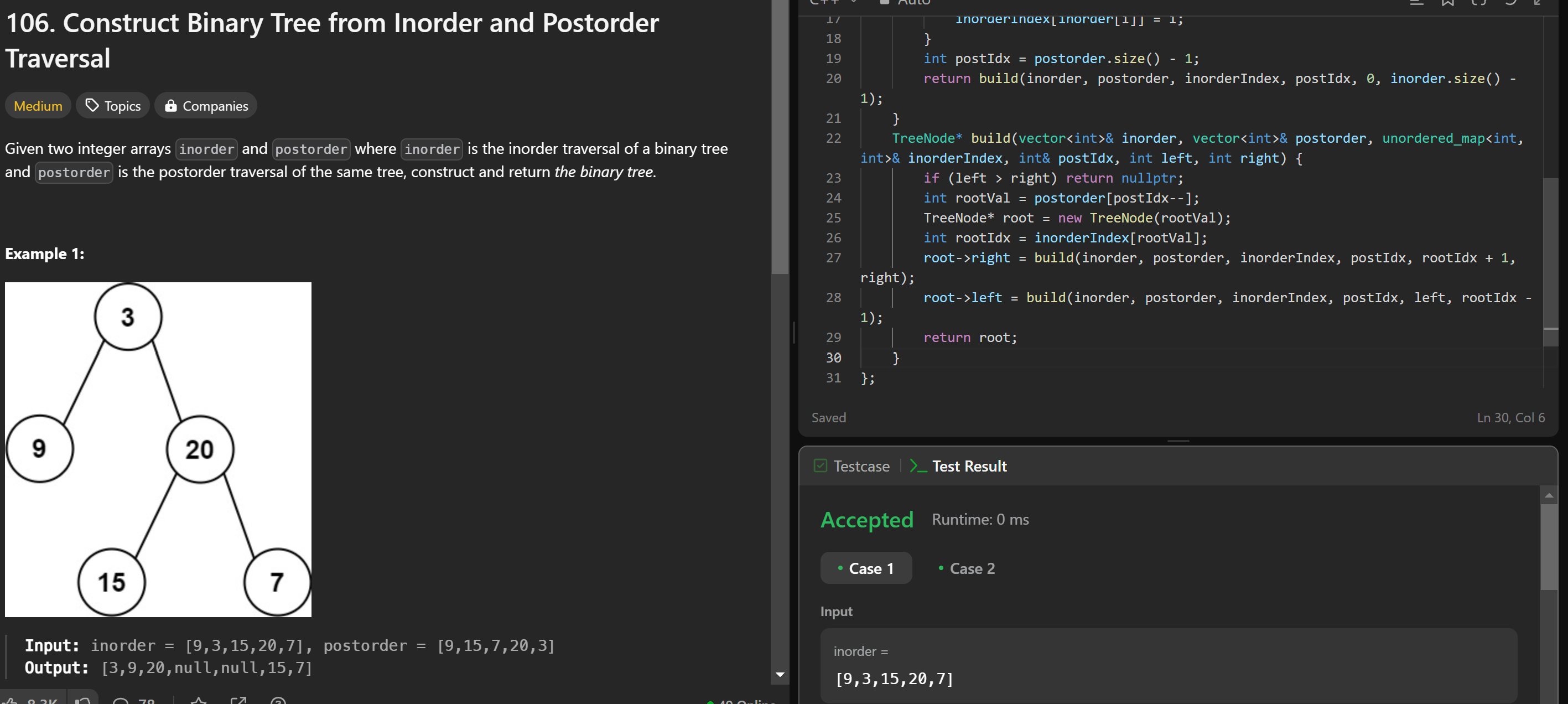
root->right = build(inorder, postorder, inorderIndex, postIdx, rootIdx + 1, right);

root->left = build(inorder, postorder, inorderIndex, postIdx, left, rootIdx - 1);

return root;

}

};



513.

class Solution {

public:

int findBottomLeftValue(TreeNode\* root) {

queue<TreeNode\*> q;

q.push(root);

int leftmostValue = root->val;

while (!q.empty()) {

int size = q.size();

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

TreeNode\* node = q.front();

q.pop();

if (i == 0) leftmostValue = node->val;

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

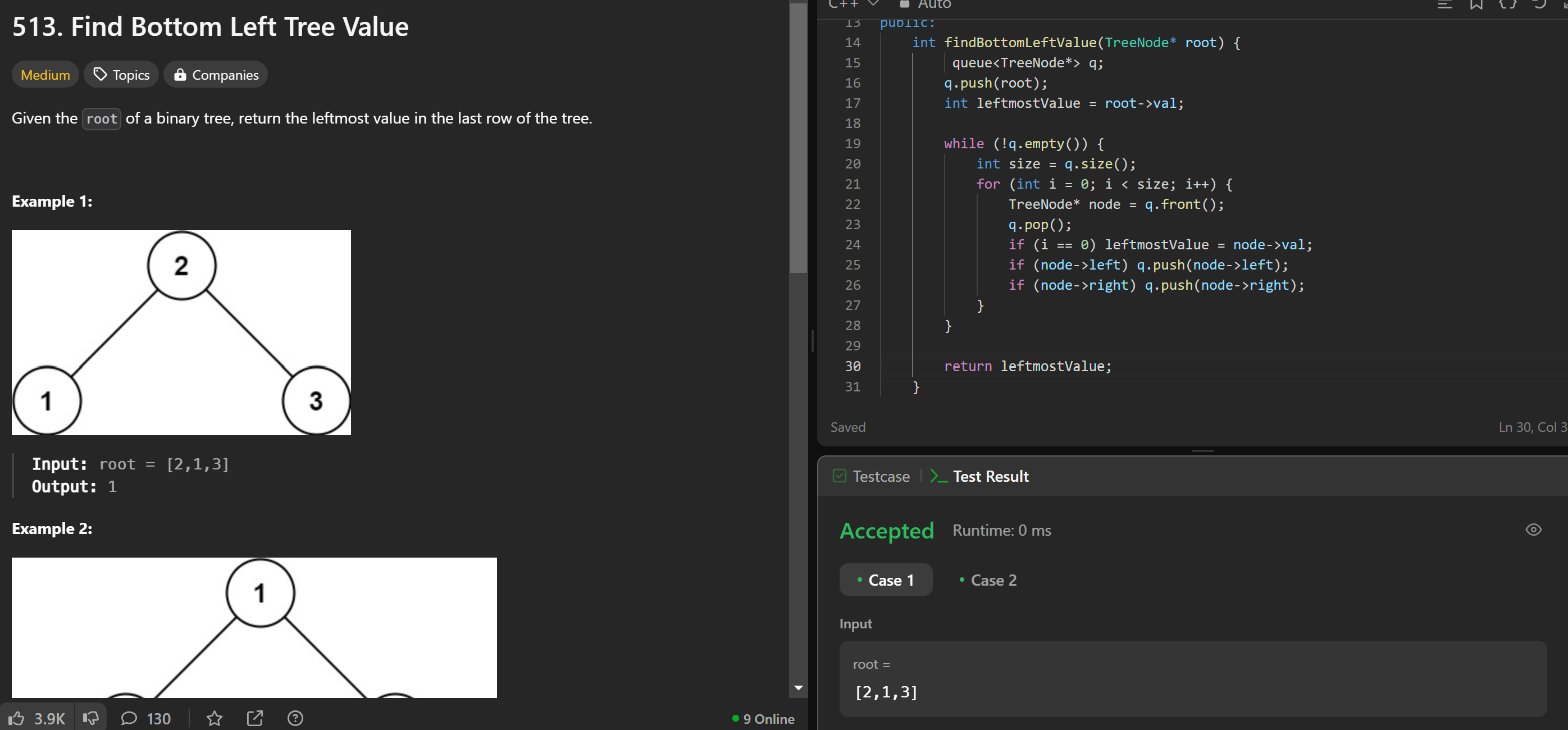
}

}

return leftmostValue;

}

};



124.

class Solution {

public:

int maxPathSum(TreeNode\* root) {

int maxSum = INT\_MIN;

maxPathSumHelper(root, maxSum);

return maxSum;

}

int maxPathSumHelper(TreeNode\* root, int &maxSum) {

if (!root) return 0;

int left = max(0, maxPathSumHelper(root->left, maxSum));

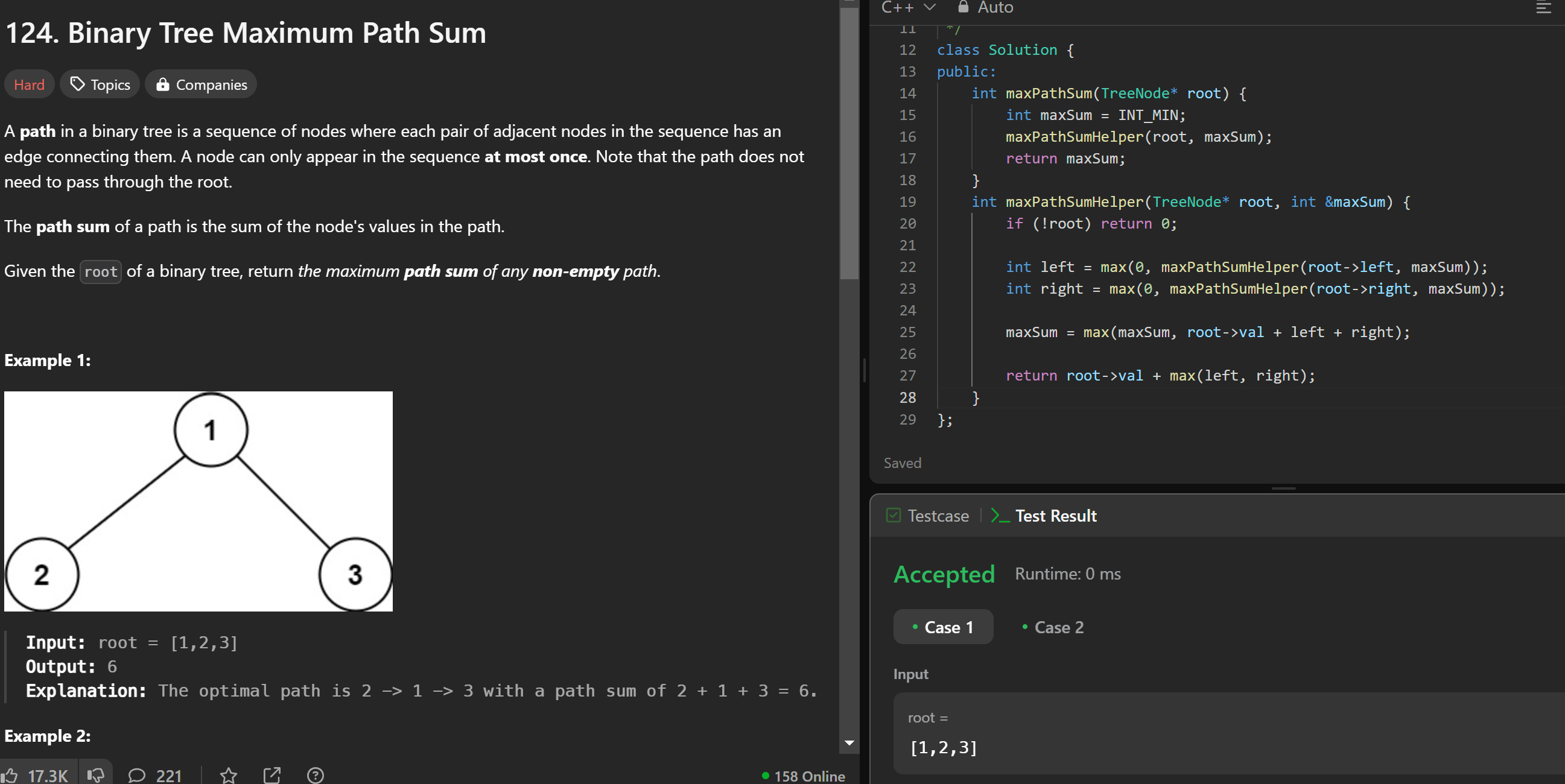
int right = max(0, maxPathSumHelper(root->right, maxSum));

maxSum = max(maxSum, root->val + left + right);

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

}

};



987.

class Solution {

public:

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

if (!root) return {};

map<int, map<int, vector<int>>> nodes;

queue<pair<TreeNode\*, pair<int, int>>> q; // {node, {row, col}}

q.push({root, {0, 0}});

while (!q.empty()) {

auto [node, position] = q.front();

q.pop();

int row = position.first, col = position.second;

nodes[col][row].push\_back(node->val);

if (node->left) q.push({node->left, {row + 1, col - 1}});

if (node->right) q.push({node->right, {row + 1, col + 1}});

}

vector<vector<int>> result;

for (auto& col : nodes) {

vector<int> colNodes;

for (auto& row : col.second) {

sort(row.second.begin(), row.second.end());

colNodes.insert(colNodes.end(), row.second.begin(), row.second.end());

}

result.push\_back(colNodes);

}

return result;

}

};

