



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

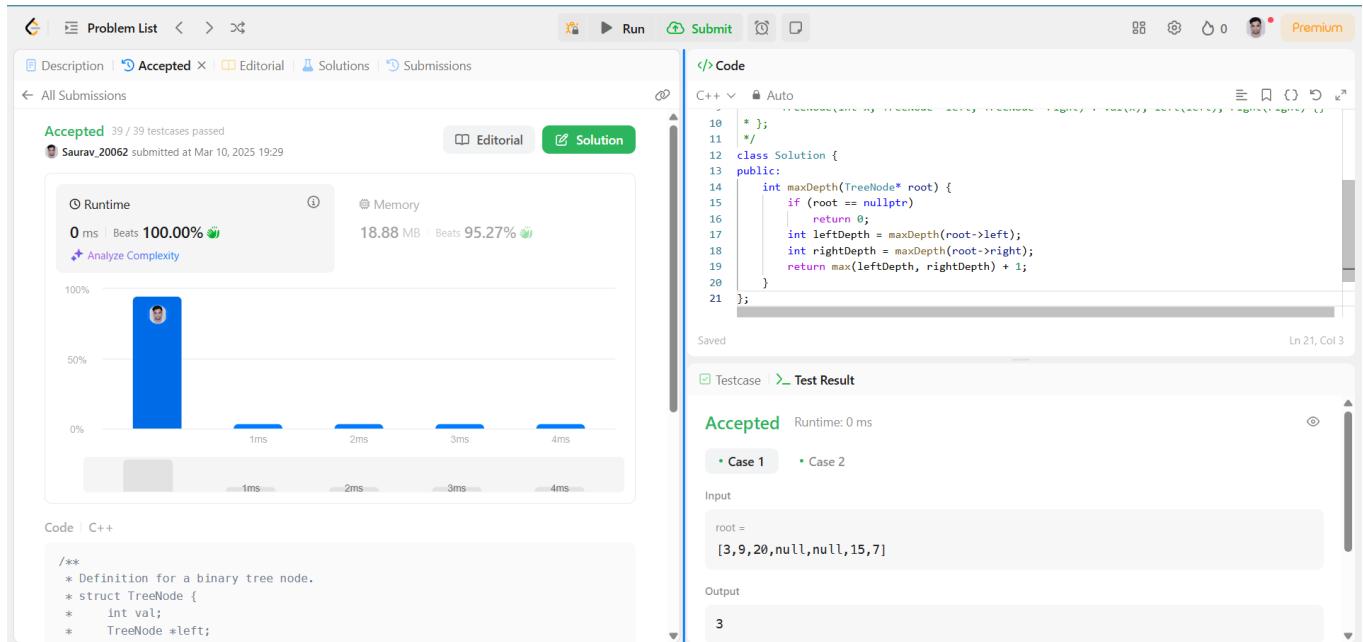
Discover. Learn. Empower.

## Experiment 5

**Student Name:** Saurav Ashiwal  
**Branch:** CSE  
**Semester:** 6  
**Subject Name:** AP lab-2

**UID:** 22BCS13250  
**Section/Group:** 614/B  
**Date of Performance:** 9/03/2025  
**Subject Code:** 22CSP-351

### **Q 1. Maximum Depth of Binary Tree**



The screenshot shows a LeetCode submission page for the problem "Maximum Depth of Binary Tree". The code is written in C++ and has been accepted. The runtime is 0 ms (beats 100.00%) and the memory usage is 18.88 MB (beats 95.27%). The code itself is a recursive function to calculate the maximum depth of a binary tree.

```
C++ v Auto
10     * );
11 */
12 class Solution {
13 public:
14     int maxDepth(TreeNode* root) {
15         if (root == nullptr)
16             return 0;
17         int leftDepth = maxDepth(root->left);
18         int rightDepth = maxDepth(root->right);
19         return max(leftDepth, rightDepth) + 1;
20     }
21 };
```

The test case input is `root = [3,9,20,null,null,15,7]` and the output is `3`.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Q 2. Validate Binary Search Tree

Accepted 86 / 86 testcases passed

Saurav\_20062 submitted at Mar 10, 2025 19:31

Runtime: 0 ms | Memory: 21.80 MB

```
10 * };
11 */
12 class Solution {
13 public:
14     bool isValidBST(TreeNode* root, long long minValue = LLONG_MIN, long long maxValue = LLONG_MAX) {
15         if (root == nullptr)
16             return true;
17         if (root->val <= minValue || root->val >= maxValue)
18             return false;
19         return isValidBST(root->left, minValue, root->val) && isValidBST(root->right, root->val, maxValue);
20     }
21 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
root = [2,1,3]
```

Output

```
true
```

## Q 3. Symmetric Tree

Accepted 200 / 200 testcases passed

Saurav\_20062 submitted at Mar 10, 2025 19:32

Runtime: 0 ms | Memory: 18.61 MB

```
12 class Solution {
13 public:
14     bool isSymmetric(TreeNode* root) {
15         if (root == nullptr)
16             return true;
17         return isMirror(root->left, root->right);
18     }
19
20     bool isMirror(TreeNode* left, TreeNode* right) {
21         if (left == nullptr && right == nullptr)
22             return true;
23         if (left == nullptr || right == nullptr)
24             return false;
25         return (left->val == right->val) &&
26                isMirror(left->left, right->right) &&
27                isMirror(left->right, right->left);
28     }
29 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
root = [1,2,2,3,4,4,3]
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Q 4. Binary Tree Zigzag Level Order Traversal

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 103. Binary Tree Zigzag Level Order Traversal

Medium | Topics | Companies

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

**Example 1:**

```
graph TD; 3((3)) --> 9((9)); 3 --> 20((20)); 20 --> 15((15)); 20 --> 7((7))
```

**Input:** root = [3,9,20,null,null,15,7]  
**Output:** [[3],[20,9],[15,7]]

11.3K | 136 | 87 Online

Code

```
C++ v Auto
12 class Solution {
13 public:
14     vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
15         vector<vector<int>> result;
16         if (root == nullptr) return result;
17
18         queue<TreeNode*> q;
19         q.push(root);
20         bool leftToRight = true;
21
22         while (!q.empty()) {
23             int size = q.size();
24             vector<int> level(size);
25             for (int i = 0; i < size; ++i) {
26                 TreeNode* node = q.front();
```

Saved | Ln 45, Col 1

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 | Case 2 | Case 3

Input

```
root =
[3,9,20,null,null,15,7]
```

Output

## Q 5. Lowest Common Ancestor of a Binary Tree

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 236. Lowest Common Ancestor of a Binary Tree

Medium | Topics | Companies

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the [definition of LCA on Wikipedia](#): "The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself)."

**Example 1:**

```
graph TD; 3((3)) --> 5((5)); 3 --> 1((1)); 5 --> 6((6)); 5 --> 2((2)); 1 --> 0((0)); 1 --> 8((8)); 2 --> 7((7)); 2 --> 4((4))
```

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1  
**Output:** 3  
**Explanation:** The LCA of nodes 5 and 1 is 3.

17.3K | 116 | 281 Online

Code

```
C++ v Auto
10 class Solution {
11 public:
12     TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
13         if (root == nullptr || root == p || root == q)
14             return root;
15
16         TreeNode* left = lowestCommonAncestor(root->left, p, q);
17         TreeNode* right = lowestCommonAncestor(root->right, p, q);
18
19         if (left != nullptr && right != nullptr)
20             return root;
21
22         return (left != nullptr) ? left : right;
23     }
24 };
```

Saved | Ln 25, Col 1

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 | Case 2 | Case 3

Input

```
root =
[3,5,1,6,2,0,8,null,null,7,4]
```

p =



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Q 6. Binary Tree Inorder Traversal

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 94. Binary Tree Inorder Traversal

Easy | Topics | Companies

Given the `root` of a binary tree, return the *inorder traversal* of its nodes' values.

**Example 1:**

**Input:** root = [1,null,2,3]  
**Output:** [1,3,2]

**Explanation:**

```
graph TD; 1((1)) --> 2((2)); 1 --> 3((3))
```

14K | 190 | 184 Online

C++ v Auto

```
12 class Solution {
13 public:
14     void inorder(TreeNode* root, vector<int>& result) {
15         if (root == nullptr) return;
16         inorder(root->left, result); // Visit left subtree
17         result.push_back(root->val); // Visit node
18         inorder(root->right, result); // Visit right subtree
19     }
20
21     vector<int> inorderTraversal(TreeNode* root) {
22         vector<int> result;
23         inorder(root, result);
24         return result;
25     }
26
27 }
```

Saved | Ln 27, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4

Input  
root = [1,null,2,3]

Output

## Q7. Binary Tree Level Order Traversal

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 102. Binary Tree Level Order Traversal

Medium | Topics | Companies | Hint

Given the `root` of a binary tree, return the *level order traversal* of its nodes' values. (i.e., from left to right, level by level).

**Example 1:**

```
graph TD; 3((3)) --> 9((9)); 3 --> 20((20)); 20 --> 15((15)); 20 --> 7((7))
```

**Input:** root = [3,9,20,null,null,15,7]  
**Output:** [[3],[9,20],[15,7]]

16K | 123 | 170 Online

C++ v Auto

```
12 class Solution {
13 public:
14     vector<vector<int>> levelOrder(TreeNode* root) {
15         vector<vector<int>> result;
16         if (!root) return result;
17         queue<TreeNode*> q;
18         q.push(root);
19         while(!q.empty()){
20             int n = q.size();
21             vector<int> ans;
22             while(n--){
23                 TreeNode* temp = q.front();
24                 q.pop();
25                 ans.push_back (temp->val);
26                 if(temp->left){
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input  
root = [3,9,20,null,null,15,7]

Output



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Q 8. Kth Smallest Element in a BST

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 230. Kth Smallest Element in a BST

Medium Topics Companies Hint

Given the root of a binary search tree, and an integer k, return the  $k^{\text{th}}$  smallest value (*1-indexed*) of all the values of the nodes in the tree.

Example 1:

```
graph TD; 3((3)) --> 1((1)); 3 --> 4((4)); 1 --> 2((2))
```

Input: root = [3,1,4,null,2], k = 1  
Output: 1

11.9K 122 116 Online

Code

```
C++ v Auto
12 class Solution {
13 public:
14     int kthSmallest(TreeNode* root, int k) {
15         int count = 0;
16         int result = -1;
17         inorder(root, k, count, result);
18         return result;
19     }
20
21     void inorder(TreeNode* node, int k, int& count, int& result) {
22         if (node == nullptr) return;
23
24         inorder(node->left, k, count, result); // Traverse the left subtree
25
26         count++; // Increment the node count
27         if (count == k) result = node->val;
28
29         inorder(node->right, k, count, result); // Traverse the right subtree
30     }
31 }
```

Saved Ln 35, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =  
[3,1,4,null,2]

k =

## Q 9. Populating Next Right Pointers in Each Node

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

### 116. Populating Next Right Pointers in Each Node

Medium Topics Companies

You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Example 1:

```
graph TD; 1((1)) --- 2((2)); 1 --- 3((3)); 2 --- 4((4)); 2 --- 5((5)); 3 --- 6((6)); 3 --- 7((7))
```

10K 73 48 Online

Code

```
C++ v Auto
19 class Solution {
20 public:
21     Node* connect(Node* root) {
22         if (root == nullptr) return nullptr;
23
24         Node* leftmost = root;
25
26         while (leftmost->left != nullptr) {
27             Node* head = leftmost;
28
29             while (head != nullptr) {
30                 // Connect the left and right child of the current node
31                 head->left->next = head->right;
32
33                 // Connect the right child of current node to the left child of next node
34                 head = head->right;
35             }
36             leftmost = leftmost->left;
37         }
38     }
39 }
```

Saved Ln 49, Col 1

Testcase | Test Result

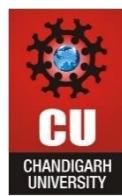
Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =  
[1,2,3,4,5,6,7]

Output



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Q 10. Sum of Left Leaves

Problem List < > Run Submit Premium

Description Editorial Solutions Submissions

### 404. Sum of Left Leaves

Easy Topics Companies

Given the `root` of a binary tree, return *the sum of all left leaves*.

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

Example 1:

```
C++ v Auto
11 */
12 class Solution {
13 public:
14     int sumOfLeftLeaves(TreeNode* root) {
15         if (root == nullptr) return 0;
16
17         int sum = 0;
18
19         // Check if the left child is a leaf
20         if (root->left && !root->left->left && !root->left->right) {
21             sum += root->left->val;
22         }
23
24         // Recursively find the sum of left leaves in the left and right subtrees
25         sum += sumOfLeftLeaves(root->left);
26         sum += sumOfLeftLeaves(root->right);
}
Ln 31, Col 1
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
root =
[3,9,20,null,null,15,7]
```

Output

5.6K 92 21 Online