

Assignment 7

Problem 1 - [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));
    }
};
```

The screenshot displays a coding platform interface for the problem "104. Maximum Depth of Binary Tree". The interface is divided into several sections:

- Submissions:** A tab at the top left showing a 3D bar chart and the text "No data".
- Code:** A code editor on the top right showing the C++ solution for the problem.
- Testcase:** A section on the bottom right showing the test result as "Accepted" with a runtime of 0 ms. It includes input and output fields for Case 1.
- Description:** A section on the bottom left providing the problem statement, an example diagram, and statistics (13.4K likes, 163 comments, 304 Online).

Problem Statement: Given the `root` of a binary tree, return its maximum depth. A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1: A diagram shows a root node with value 3, which has two children, both of which are null. The maximum depth is 1.

Testcase Details:

- Case 1:** Input: `root = [3,9,20,null,null,15,7]`, Output: `3`, Expected: `3`.

Statistics: 13.4K likes, 163 comments, 304 Online.

Problem 2 - [Validate Binary Search Tree](#)

```
class Solution {
```

```
bool isPossible(TreeNode* root, long long l, long long r){
    if(root == nullptr) return true;
    if(root->val < r and root->val > l)
        return isPossible(root->left, l, root->val) and
                isPossible(root->right, root->val, r);
    else return false;
}
```

```
public:
```

```
bool isValidBST(TreeNode* root) {
    long long int min = -1000000000000, max = 1000000000000;
    return isPossible(root, min, max);
}
};
```

The screenshot shows a LeetCode submission interface. The top left pane displays submission statistics: 'Accepted 86 / 86 testcases passed', 'Ayush submitted at Mar 28, 2025 12:01', and performance metrics: 'Runtime 0 ms Beats 100.00%', 'Memory 21.86 MB Beats 76.17%'. A bar chart shows the user's performance relative to others. The top right pane shows the C++ code for the solution. The bottom left pane shows the problem description for '98. Validate Binary Search Tree', including the definition of a valid BST and an example. The bottom right pane shows the test case input '[2,1,3]' and a diagram of the resulting binary tree with root 2, left child 1, and right child 3.

Submissions Accepted ×

All Submissions

Accepted 86 / 86 testcases passed

Ayush submitted at Mar 28, 2025 12:01

Editorial Solution

Runtime 0 ms Beats 100.00%

Memory 21.86 MB Beats 76.17%

Analyze Complexity

100%

0%

1ms 2ms 3ms 4ms

98. Validate Binary Search Tree Solved ✓

Medium Topics Companies

Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

- The left **subtree** of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

17.4K 234 209 Online

Code

```
bool isPossible(TreeNode* root, long long l, long long r){
    if(root == nullptr) return true;
    if(root->val < r and root->val > l)
        return isPossible(root->left, l, root->val) and
                isPossible(root->right, root->val, r);
    else return false;
}

public:
    bool isValidBST(TreeNode* root) {
        long long int min = -1000000000000, max = 1000000000000;
        return isPossible(root, min, max);
    }
};
```

Saved Ln 16, Col 3

Testcase Test Result

Case 1 Case 2 +

root =

[2,1,3]

2

1 3

Source

Problem 3 – [Symmetric Tree](#)

```
class Solution {
```

```
public:
```

```
    bool isSymmetric(TreeNode* root) {  
        return isMirror(root->left, root->right);  
    }
```

```
private:
```

```
    bool isMirror(TreeNode* n1, TreeNode* n2) {  
        if (n1 == nullptr && n2 == nullptr) {  
            return true;  
        }
```

```
        if (n1 == nullptr || n2 == nullptr) {  
            return false;  
        }
```

```
        return n1->val == n2->val && isMirror(n1->left, n2->right) && isMirror(n1->right, n2->left);  
    }  
};
```

The screenshot displays a coding platform interface with four main panels:

- Submissions:** Shows the submission status as 'Accepted' for 200/200 testcases. The user 'Ayush' submitted on Mar 28, 2025 at 12:02. Performance metrics include 0 ms runtime (beats 100.00%) and 18.48 MB memory (beats 60.17%). A bar chart shows the user's performance relative to others.
- Code:** Displays the C++ code for the solution, including the `isMirror` helper function and the `isSymmetric` method.
- Description:** Contains the problem statement for '101. Symmetric Tree', which asks to check if a binary tree is a mirror of itself. It includes an 'Easy' difficulty tag and an example diagram of a symmetric tree with root 1 and children 2 and 2, which have children 3 and 4 respectively.
- Testcase:** Shows the input for Case 1 as the array `[1, 2, 2, 3, 4, 4, 3]` and a corresponding tree diagram.

Problem 4 - [Binary Tree Level Order Traversal](#)

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i<s;i++){
                TreeNode *node=q.front();
                q.pop();
                if(node->left!=NULL)q.push(node->left);
                if(node->right!=NULL)q.push(node->right);
                v.push_back(node->val);
            }
            ans.push_back(v);
        }
        return ans;
    }
};
```

The screenshot displays a coding platform interface with the following components:

- Submissions Panel:** Shows 'Accepted' status for 200/200 testcases. The submission was made by 'Ayush' on Mar 28, 2025 at 12:02. It includes a 'Runtime' section (0 ms, 100.00% beats) and a 'Memory' section (18.48 MB, 60.17% beats).
- Code Editor:** Contains C++ code for a recursive function `isMirror` that checks if a binary tree is a mirror of itself. The code uses a helper function `isMirror` to compare nodes and their children recursively.
- Testcase Panel:** Shows 'Case 1' with the input `root = [1,2,2,3,4,4,3]`. Below the input is a diagram of a binary tree with root 1, left child 2, right child 2, and further children 3, 4, 4, 3 respectively.
- Problem Description:** Titled '101. Symmetric Tree', it asks to check if a binary tree is a mirror of itself. It includes an 'Example 1' with a diagram of a root node 1 and its children.

Problem 5- [Convert Sorted Array to Binary Search Tree](#)

```
#include <vector>
```

```
using namespace std;
```

```
class Solution {
```

```
public:
```

```
    TreeNode* sortedArrayToBST(vector<int>& nums) {  
        return helper(nums, 0, nums.size() - 1);  
    }
```

```
private:
```

```
    TreeNode* helper(vector<int>& nums, int left, int right) {  
        if (left > right) return nullptr;  
        int mid = left + (right - left) / 2;  
        TreeNode* root = new TreeNode(nums[mid]);  
        root->left = helper(nums, left, mid - 1);  
        root->right = helper(nums, mid + 1, right);  
        return root;  
    }
```

```
};
```

The screenshot displays a coding platform interface with the following components:

- Submissions Panel:** Shows 'Accepted' status for 31/31 testcases. A bar chart indicates runtime performance, with the user's solution at 3 ms (59.52% beats) and 23.05 MB memory (22.90% beats).
- Code Editor:** Contains the C++ solution code for the problem, including the `sortedArrayToBST` method and a recursive `helper` function.
- Testcase Panel:** Shows 'Case 1' with the input array `nums = [-10, -3, 0, 5, 9]`.
- Problem Description:** Titled '108. Convert Sorted Array to Binary Search Tree', it explains the task: converting a sorted array into a height-balanced BST. It includes an example of a tree structure for the array `[3, 0, 9]`.

Problem 6 - [Binary Tree Inorder Traversal](#)

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        if (root == NULL) return ans;
        vector<int> left = inorderTraversal(root->left);
        ans.insert(ans.end(), left.begin(), left.end());
        ans.push_back(root->val);
        vector<int> right = inorderTraversal(root->right);
        ans.insert(ans.end(), right.begin(), right.end());
        return ans;
    }
};
```

The screenshot shows the LeetCode submission page for problem 94, "Binary Tree Inorder Traversal". The interface is in dark mode. At the top, the problem is marked as "Accepted" with 71/71 testcases passed. The user "Ayush" submitted the solution on Mar 28, 2025 at 12:05. The solution is in C++ and is marked as "Solved".

Runtime Performance: 0 ms, Beats 100.00%. A bar chart shows the runtime performance relative to other submissions, with the user's solution being the fastest.

Memory Performance: 12.60 MB, Beats 7.24%. A bar chart shows the memory usage performance.

Example 1:
Input: root = [1,null,2,3]
Output: [1,3,2]
Explanation: The inorder traversal of the binary tree [1,null,2,3] is [1,3,2].

Testcase 1: The test result shows the input root = [1,null,2,3] and the expected output [1,3,2]. A diagram of the binary tree is shown: root (1) has a right child (2), which has a left child (3).

```
graph TD
    root((1)) --> node2((2))
    node2 --> node3((3))
```