

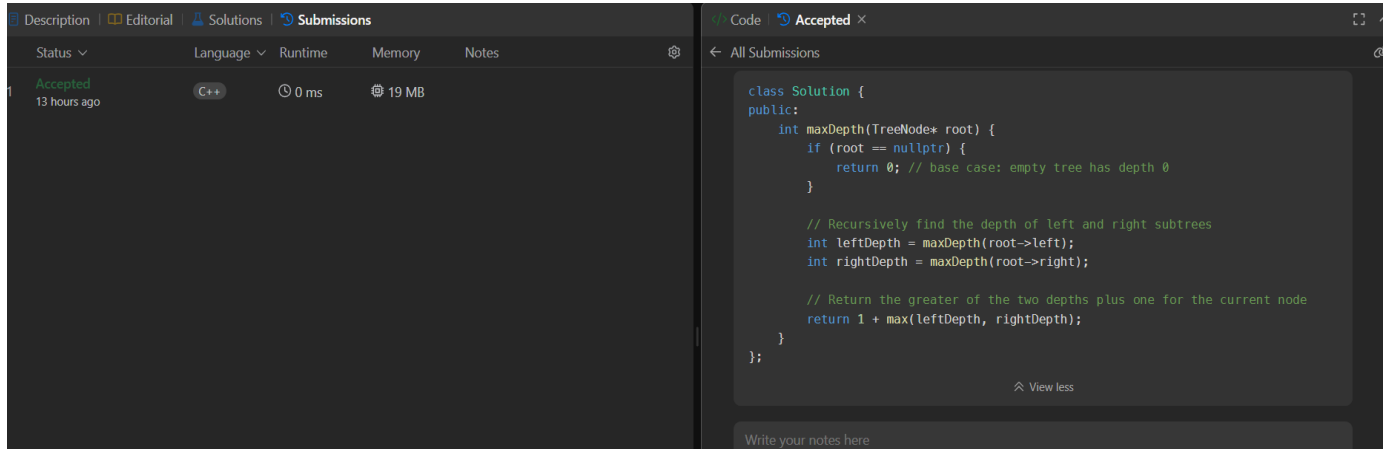
Assignment

Name – JIGYA JAIN

UID – 22BCS50101

Section – 609-A

1. Maximum Depth of Binary Tree –



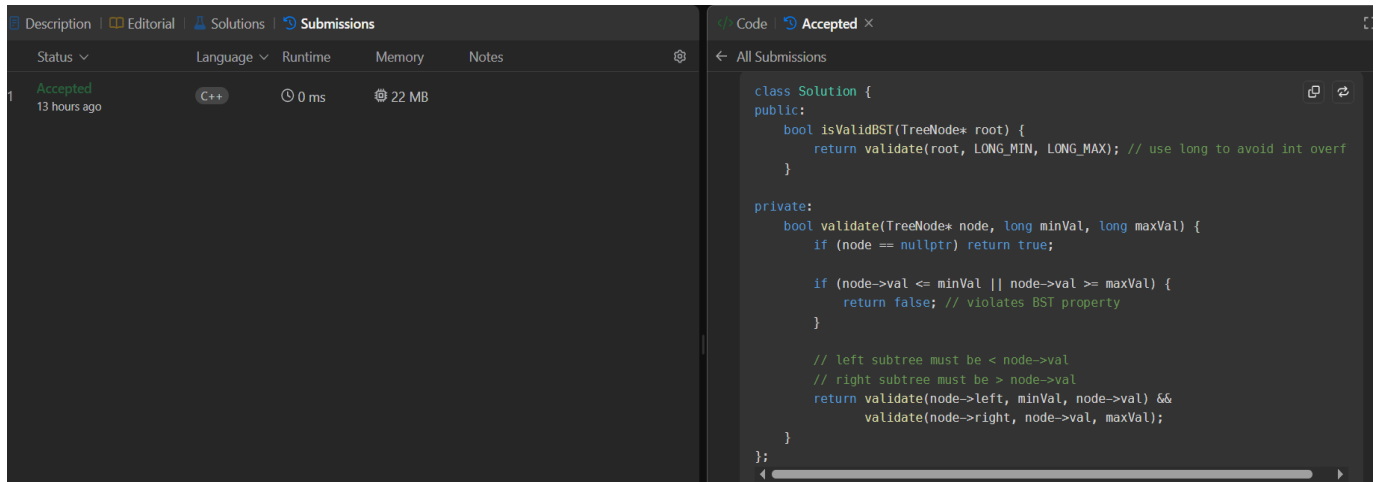
The screenshot shows a submission interface with a table of submissions and a code editor. The table has columns for Status, Language, Runtime, Memory, and Notes. The first submission is 'Accepted' 13 hours ago, using C++ with 0 ms runtime and 19 MB memory. The code editor shows the following C++ code:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (root == nullptr) {
            return 0; // base case: empty tree has depth 0
        }

        // Recursively find the depth of left and right subtrees
        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);

        // Return the greater of the two depths plus one for the current node
        return 1 + max(leftDepth, rightDepth);
    }
};
```

2. Validate Binary Search Tree –



The screenshot shows a submission interface with a table of submissions and a code editor. The table has columns for Status, Language, Runtime, Memory, and Notes. The first submission is 'Accepted' 13 hours ago, using C++ with 0 ms runtime and 22 MB memory. The code editor shows the following C++ code:

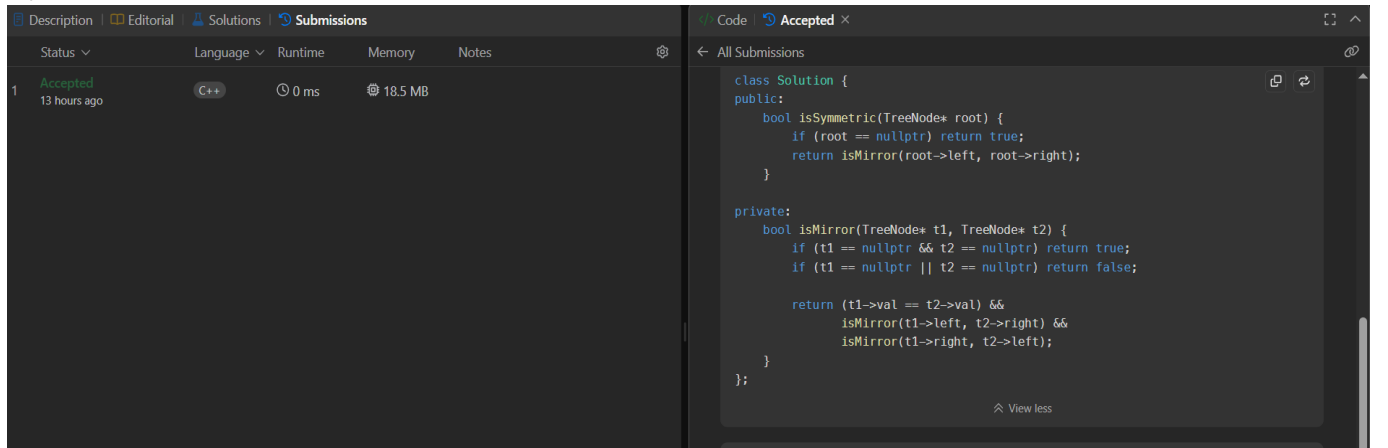
```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return validate(root, LONG_MIN, LONG_MAX); // use long to avoid int overflow
    }

private:
    bool validate(TreeNode* node, long minVal, long maxVal) {
        if (node == nullptr) return true;

        if (node->val <= minVal || node->val >= maxVal) {
            return false; // violates BST property
        }

        // left subtree must be < node->val
        // right subtree must be > node->val
        return validate(node->left, minVal, node->val) &&
            validate(node->right, node->val, maxVal);
    }
};
```

3. Symmetric Tree –



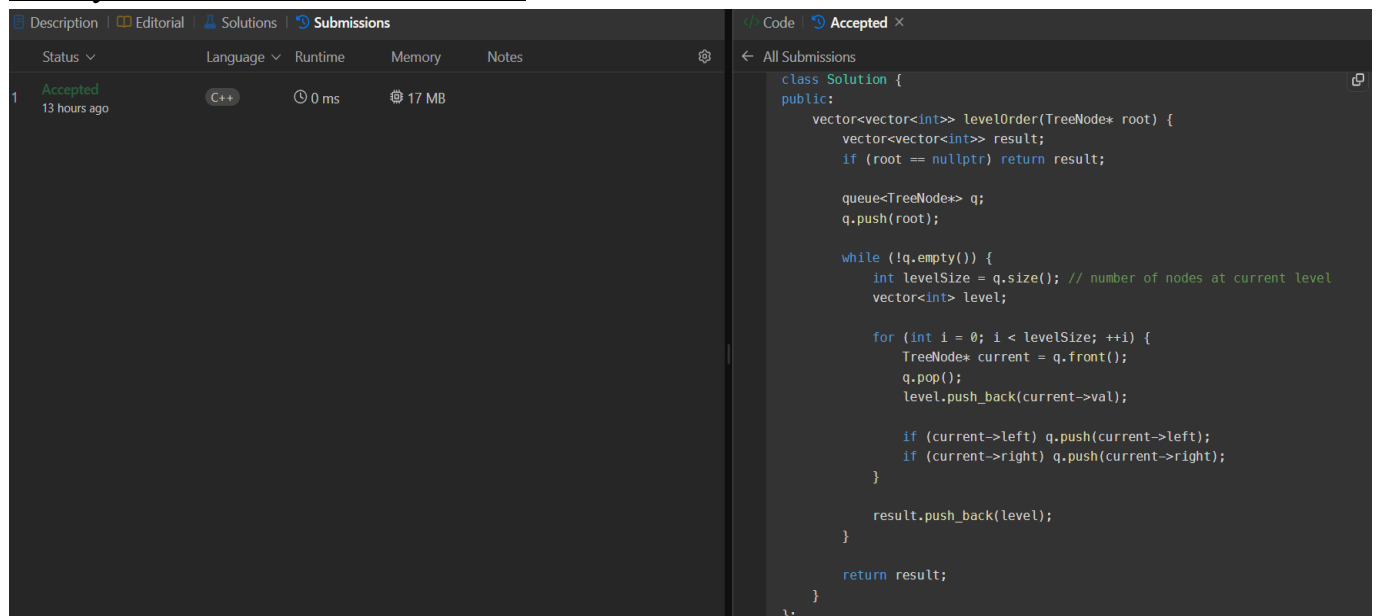
The screenshot shows a submission interface with a table of submissions and a code editor. The table has columns for Status, Language, Runtime, Memory, and Notes. The first submission is 'Accepted' 13 hours ago, using C++ with 0 ms runtime and 18.5 MB memory. The code editor shows the following C++ code:

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (root == nullptr) return true;
        return isMirror(root->left, root->right);
    }

private:
    bool isMirror(TreeNode* t1, TreeNode* t2) {
        if (t1 == nullptr && t2 == nullptr) return true;
        if (t1 == nullptr || t2 == nullptr) return false;

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

4. Binary Tree Level Order Traversal –



Submission details: Accepted, 13 hours ago, C++, 0 ms, 17 MB.

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) return result;

        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int levelSize = q.size(); // number of nodes at current level
            vector<int> level;

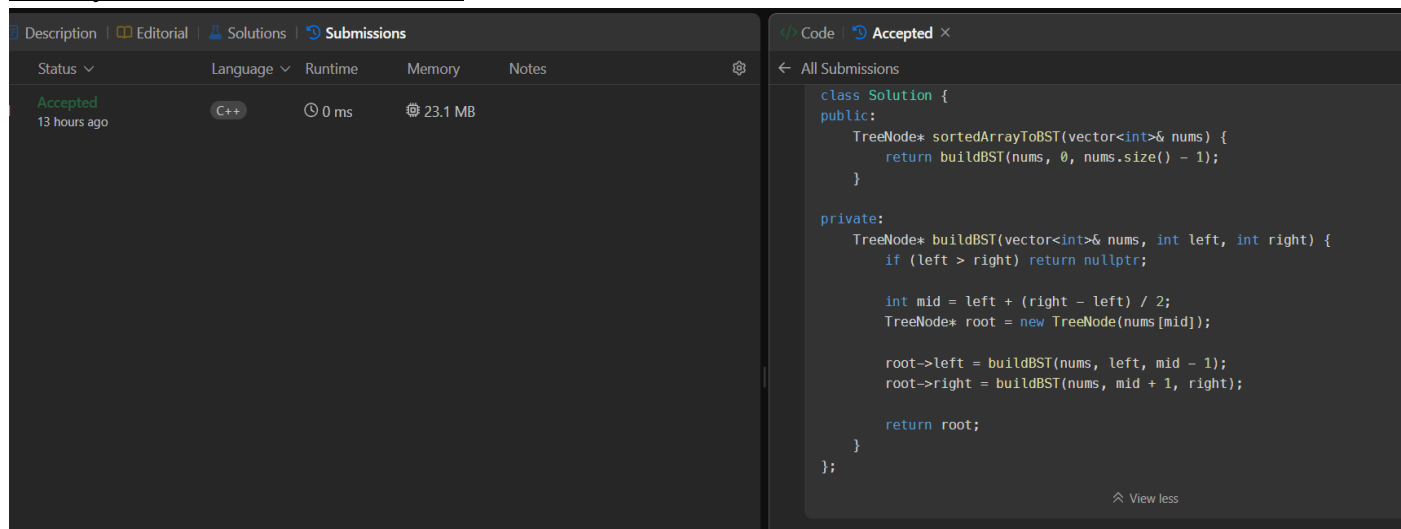
            for (int i = 0; i < levelSize; ++i) {
                TreeNode* current = q.front();
                q.pop();
                level.push_back(current->val);

                if (current->left) q.push(current->left);
                if (current->right) q.push(current->right);
            }

            result.push_back(level);
        }

        return result;
    }
};
```

5. Binary Tree Inorder Traversal –



Submission details: Accepted, 13 hours ago, C++, 0 ms, 23.1 MB.

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildBST(nums, 0, nums.size() - 1);
    }

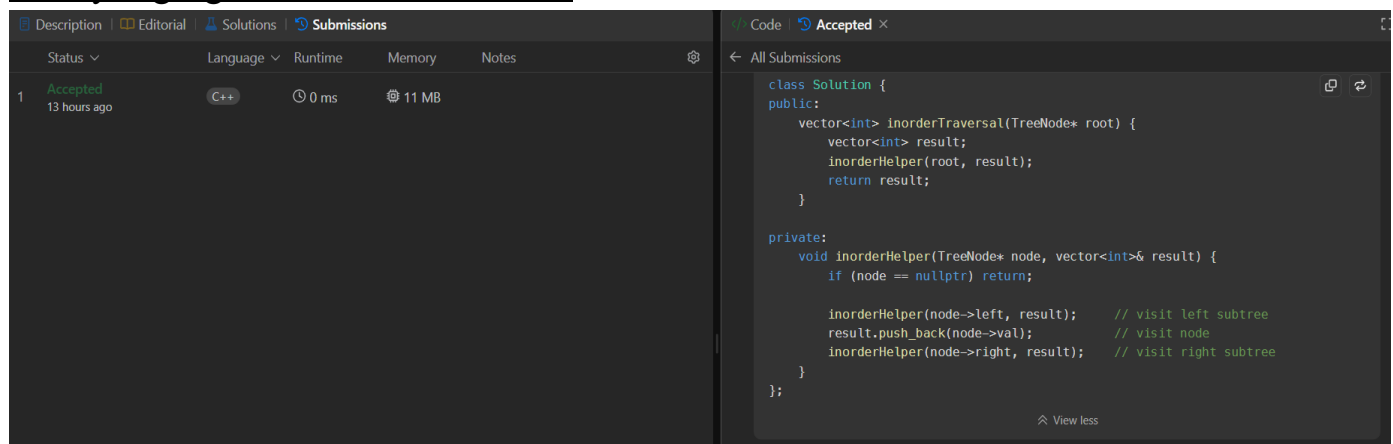
private:
    TreeNode* buildBST(vector<int>& nums, int left, int right) {
        if (left > right) return nullptr;

        int mid = left + (right - left) / 2;
        TreeNode* root = new TreeNode(nums[mid]);

        root->left = buildBST(nums, left, mid - 1);
        root->right = buildBST(nums, mid + 1, right);

        return root;
    }
};
```

6. Binary Zigzag Level Order Traversal –



Submission details: Accepted, 13 hours ago, C++, 0 ms, 11 MB.

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderHelper(root, result);
        return result;
    }

private:
    void inorderHelper(TreeNode* node, vector<int>& result) {
        if (node == nullptr) return;

        inorderHelper(node->left, result); // visit left subtree
        result.push_back(node->val);       // visit node
        inorderHelper(node->right, result); // visit right subtree
    }
};
```

7. Postorder Traversal –

Description
Accepted
Editorial
Solutions
Submissions

All Submissions

Accepted 232 / 232 testcases passed
Jigyajain submitted at Mar 06, 2025 22:27

Editorial
Solution

Runtime
0 ms | Beats 100.00%

Memory
42.76 MB | Beats 36.07%

```

10  */
11  class Solution {
12      public ListNode rotateRight(ListNode head, int k) {
13          if (head == null || head.next == null) {
14              return head;
15          }
16
17          int nodes = 1;
18          ListNode beg = head;
19          ListNode end = head;
20
21          while (end.next != null) {
22              end = end.next;
23              nodes++;
24          }
25
26          int rotate = k % nodes;
27          if (rotate == 0) {
28              return head;
29          }
30
31          end.next = head;

```

8. Kth Smallest element in a BST –

Status
Language
Runtime
Memory
Notes

Accepted 13 hours ago
C++
1 ms
27.5 MB

```

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 postIndex = postorder.size() - 1;
        return build(inorder, postorder, inorderIndex, 0, inorder.size() - 1, postIndex);
    }

private:
    TreeNode* build(const vector<int>& inorder, const vector<int>& postorder,
                    unordered_map<int, int>& inorderIndex,
                    int inStart, int inEnd, int& postIndex) {
        if (inStart > inEnd) return nullptr;

        int rootVal = postorder[postIndex--];
        TreeNode* root = new TreeNode(rootVal);

        int inRoot = inorderIndex[rootVal];

        // Important: build right subtree first (because we go backwards in postorder)
        root->right = build(inorder, postorder, inorderIndex, inRoot + 1, inEnd, postIndex);
        root->left = build(inorder, postorder, inorderIndex, inStart, inRoot - 1, postIndex);

        return root;
    }
};

```

9. Populating Next Right –

Description
Editorial
Solutions
Submissions

Status
Language
Runtime
Memory
Notes

Accepted 13 hours ago
C++
0 ms
24.2 MB

```

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0;
        int result = -1;
        inorder(root, k, count, result);
        return result;
    }

private:
    void inorder(TreeNode* node, int k, int& count, int& result) {
        if (!node) return;

        inorder(node->left, k, count, result);

        count++;
        if (count == k) {
            result = node->val;
            return;
        }

        inorder(node->right, k, count, result);
    }
};

```

10. Pointers in Each Node -

Description | Editorial | Solutions | Submissions

Status ▾

Language ▾

Runtime

Memory

Notes

⚙

1

Accepted
13 hours ago

C++

13 ms

18.8 MB

Code Accepted ×

← All Submissions

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root) return nullptr;

        Node* level_start = root;

        while (level_start->left) {
            Node* curr = level_start;

            while (curr) {

                curr->left->next = curr->right;

                if (curr->next) {
                    curr->right->next = curr->next->left;
                }

                curr = curr->next;
            }

            level_start = level_start->left;
        }

        return root;
    }
};
```