

AP Assignment 7

Name-> Rahul Gupta

UID-> 22BCS10469

Section-> 609-B

Q1.) Maximum Depth of Binary Tree

```
class Solution {  
public:  
    int maxDepth(TreeNode* root) {  
        if(root == NULL) return 0;  
        int left = maxDepth(root->left);  
        int right = maxDepth(root->right);  
        return max(left, right) + 1;  
    }  
};
```

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

- Top Bar:** Includes navigation icons, a 'Problem List' dropdown, and status indicators for 'Run', 'Submit', and 'Premium'.
- Left Panel:**
 - Accepted:** 39 / 39 testcases passed. Submitted by Rahul Gupta on Mar 28, 2025 at 12:15.
 - Runtime:** 0 ms | Beats 100.00%.
 - Memory:** 19.16 MB | Beats 13.09%.
 - Bar Chart:** A chart showing runtime performance across different time intervals (1ms to 4ms).
- Code Editor:** Displays the C++ code for the solution, with line numbers 1 through 9. The code is saved.
- Testcase Section:**
 - Case 1:** Input: `root = [3,9,20,null,null,15,7]`.
 - Diagram:** A binary tree diagram with root 3, left child 9, and right child 20. Node 20 has left child 15 and right child 7.

Q2.) Validate Binary Search Tree

```
class Solution {
```

```
public:
```

```
    bool isValidBST(TreeNode* root) {  
        return valid(root, LONG_MIN, LONG_MAX);  
    }
```

```
private:
```

```
    bool valid(TreeNode* node, long minimum, long maximum) {  
        if (!node) return true;  
        if (!(node->val > minimum && node->val < maximum)) return false;  
        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);  
    }  
};
```

The screenshot shows a coding platform interface with a dark theme. The top navigation bar includes 'Problem List', 'Run', 'Submit', and 'Premium' (70% off). The left sidebar shows the problem status: 'Accepted' (86 / 86 testcases passed) and the user 'Rahul Gupta' submitted on Mar 28, 2025. The 'Runtime' section shows '0 ms' and 'Beats 100.00%'. The 'Memory' section shows '21.70 MB' and 'Beats 99.76%'. A bar chart shows the performance of the solution compared to other submissions. The main code editor displays the C++ solution for the 'Validate Binary Search Tree' problem. The code defines a 'Solution' class with a public 'isValidBST' method and a private 'valid' method. The 'valid' method uses recursive logic to check if the tree is a valid BST. The 'Testcase' section shows 'Case 1' with the input 'root = [2,1,3]' and a diagram of a binary tree with root 2, left child 1, and right child 3.

```
class Solution {  
public:  
    bool isValidBST(TreeNode* root) {  
        return valid(root, LONG_MIN, LONG_MAX);  
    }  
private:  
    bool valid(TreeNode* node, long minimum, long maximum) {  
        if (!node) return true;  
        if (!(node->val > minimum && node->val < maximum)) return false;  
        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);  
    }  
};
```

Testcase: Case 1
root = [2,1,3]
Diagram: A binary tree with root 2, left child 1, and right child 3.

Q3.) [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 code editor interface for a C++ solution. The left sidebar shows submission details: 'Accepted 200 / 200 testcases passed', submitted by 'Rahul Gupta' on 'Mar 31, 2025 10:35'. Performance metrics are listed: 'Runtime 0 ms | Beats 100.00%' and 'Memory 18.45 MB | Beats 60.43%'. A bar chart illustrates memory usage across various test cases, with the highest usage around 18.4 MB. The main editor area contains the C++ code for the 'Symmetric Tree' problem, which uses a recursive approach to check if a tree is a mirror of itself. The right sidebar shows the 'Testcase' tab with 'Case 1' selected, displaying the input array '[1,2,2,3,4,4,3]' and a visual representation of the resulting binary tree structure.

Accepted 200 / 200 testcases passed
Rahul Gupta submitted at Mar 31, 2025 10:35

Runtime
0 ms | Beats 100.00%

Memory
18.45 MB | Beats 60.43%

Analyze Complexity

Code | C++

Testcase | Test Result

Case 1 Case 2 +

root =

[1,2,2,3,4,4,3]

1

Q4.) 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:

- Problem List:** Navigation tabs for Description, Editorial, Solutions, Submissions, and Accepted.
- Accepted Submissions:** A list of accepted solutions, including one by Rahul Gupta submitted at Mar 31, 2025 10:42.
- Runtime and Memory Performance:** A bar chart showing performance metrics. The runtime is 0 ms (Beats 100.00%) and memory usage is 16.99 MB (Beats 88.27%).
- Code Editor:** A C++ code editor showing the solution for the problem. The code is a class Solution with a public method levelOrder that uses a queue to traverse the tree level by level.
- Testcase and Test Result:** A section for testing the solution. It shows a testcase with the input root = [3,9,20,null,null,15,7] and a corresponding binary tree diagram.

Q6.) Convert Sorted Array to Binary Search Tree

```
class Solution {  
  
public:  
  
    TreeNode* sortedArrayToBST(vector<int>& nums) {  
        return sortedArrayToBST(nums, 0, nums.size() - 1);  
    }  
  
    TreeNode* sortedArrayToBST(vector<int>& nums, int start, int end) {  
        if(start > end) {  
            return NULL;  
        }  
  
        int mid = (start + end) / 2;  
  
        TreeNode* root = new TreeNode(nums[mid]);  
  
        root->left = sortedArrayToBST(nums, start, mid - 1);  
  
        root->right = sortedArrayToBST(nums, mid + 1, end);  
  
        return root;  
    }  
};
```

The screenshot displays the LeetCode submission page for the problem "Convert Sorted Array to Binary Search Tree". The submission is marked as "Accepted" with 31/31 testcases passed. The user "Rahul Gupta" submitted it on Mar 31, 2025 at 11:11. The runtime is 3 ms, beating 59.84% of solutions, and the memory usage is 22.82 MB, beating 81.15%. A bar chart shows the runtime distribution across different time limits (27ms, 54ms, 80ms). The C++ code is shown in the editor, implementing the recursive function `sortedArrayToBST`.

Accepted 31 / 31 testcases passed
Rahul Gupta submitted at Mar 31, 2025 11:11

Runtime
3 ms | Beats 59.84%
[Analyze Complexity](#)

Memory
22.82 MB | Beats 81.15%

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

 TreeNode* sortedArrayToBST(vector<int>& nums, int start, int end) {
 if(start > end) {
 return NULL;
 }

 int mid = (start + end) / 2;
 TreeNode* root = new TreeNode(nums[mid]);

 root->left = sortedArrayToBST(nums, start, mid - 1);
 root->right = sortedArrayToBST(nums, mid + 1, end);
 return root;
 }
};

Testcase Test Result
Case 1 Case 2
nums =
[-10,-3,0,5,9]

Q7.) 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 a C++ IDE interface with the following components:

- Top Bar:** Includes navigation icons, a "Problem List" tab, and a "Premium" badge.
- Left Panel (Submissions):**
 - Buttons: Description, Editorial, Solutions, Submissions, Accepted.
 - Tab: All Submissions.
 - Status: Accepted 71 / 71 testcases passed.
 - User: Rahul Gupta submitted at Mar 31, 2025 17:28.
 - Buttons: Editorial, Solution.
 - Runtime: 0 ms | Beats 100.00%.
 - Memory: 12.38 MB | Beats 10.65%.
 - Bar chart showing performance comparison.
 - Code editor snippet: `class Solution {`
- Right Panel (Code):**
 - Language: C++.
 - Code content: The C++ solution for Binary Tree Inorder Traversal.
 - Testcase tab: Shows Case 1 with input `root = [1,null,2,3]` and a diagram of the binary tree.

Q8.) Binary Tree Zigzag Level Order Traversal

```
class Solution {  
public:  
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {  
        vector<vector<int>> ans;  
        if(root==NULL) return ans;  
        queue<TreeNode*> q;  
        q.push(root);  
        int cnt = 1;  
        while(!q.empty()){  
            vector<int> temp;  
            int n = q.size();  
            for(int i = 0 ;i<n;i++){  
                TreeNode* node = q.front();  
                q.pop();  
                temp.push_back(node->val);  
                if(node->left!=NULL)q.push(node->left);  
                if(node->right!=NULL)q.push(node->right);  
            }  
            if(cnt%2==0) reverse(temp.begin(),temp.end());  
            ans.push_back(temp);  
            cnt++;  
        }  
        return ans; }  
};
```

The screenshot shows a C++ IDE with the following components:

- Problem List:** Zigzag Level Order Traversal
- Description:** Accepted, 33 / 33 testcases passed. Submitted by Rahul Gupta on Apr 03, 2025 at 10:43.
- Runtime:** 0 ms, Beats 100.00%.
- Memory:** 15.21 MB, Beats 20.13%.
- Code:** C++ code implementing the zigzag level order traversal algorithm.
- Testcase:** Case 1: root = [3,9,20,null,null,15,7]. The output shows a binary tree with root 3, left child 9, right child 20, and 20's left child 15.

Q9.) Construct Binary Tree from Inorder and Postorder Traversal

class Solution {

public:

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

unordered_map<int, int> index;

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

index[inorder[i]] = i;

}

return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1, 0, postorder.size() - 1, index); }

TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int inorderStart, int inorderEnd, int postorderStart, int postorderEnd, unordered_map<int, int>& index) {

if (inorderStart > inorderEnd || postorderStart > postorderEnd) {

return nullptr; }

int rootVal = postorder[postorderEnd];

TreeNode* root = new TreeNode(rootVal);

int inorderRootIndex = index[rootVal];

int leftSubtreeSize = inorderRootIndex - inorderStart;

root->left = buildTreeHelper(inorder, postorder, inorderStart, inorderRootIndex - 1, postorderStart, postorderStart + leftSubtreeSize - 1, index);

root->right = buildTreeHelper(inorder, postorder, inorderRootIndex + 1, inorderEnd, postorderStart + leftSubtreeSize, postorderEnd - 1, index);

return root; } };

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

- Problem List:** A navigation bar at the top with icons for problem list, description, editorial, solutions, and submissions.
- Accepted Submissions:** A section showing the user's submission status as 'Accepted' for 202/202 testcases passed, submitted by 'Rahul Gupta' on Apr 03, 2025 at 10:45.
- Runtime and Memory Performance:** A bar chart showing the user's performance compared to others. Runtime is 1 ms (Beats 81.54%) and Memory is 27.40 MB (Beats 73.64%).
- Code Editor:** A C++ code editor showing the solution for building a binary tree from inorder and postorder traversals. The code uses a recursive helper function and an unordered map for index lookup.
- Testcase Input:** A section for testing the solution with specific inputs. The 'inorder' array is [9, 3, 15, 20, 7] and the 'postorder' array is [9, 15, 7, 20, 3].

Q10.) [Kth Smallest Element in a BST](#)

```
class Solution {  
  
public:  
  
    int kthSmallest(TreeNode* root, int k) {  
  
        TreeNode* node=root;  
  
        stack<TreeNode*> st;  
  
        int cnt=0;  
  
        while(true){  
  
            if(node!=NULL){  
  
                st.push(node);  
  
                node=node->left;  
  
            }else{  
  
                node=st.top();  
  
                st.pop();  
  
                cnt++;  
  
                if(cnt==k){  
  
                    return node->val; }  
  
                node=node->right; } }  
  
        return -1; } };
```

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

- Problem List:** A navigation bar at the top with icons for problem list, editor, solutions, submissions, and accepted status.
- Accepted Status:** A green banner indicating "Accepted 93 / 93 testcases passed" by "Rahul Gupta" submitted at "Mar 31, 2025 17:52".
- Runtime and Memory:** Performance metrics showing "0 ms" runtime (Beats 100.00%) and "24.36 MB" memory (Beats 67.74%).
- Code Editor:** A C++ code editor showing the implementation of the `kthSmallest` function. The code uses a stack to traverse the BST in-order.
- Testcase:** A section for testing the solution with "Case 1" showing the input `root = [3,1,4,null,2]` and a corresponding binary tree diagram.
- Test Result:** A section for viewing the test results, currently showing a "Source" icon.

Q11.) Populating Next Right Pointers in Each Node

```
class Solution {  
  
public:  
  
    Node* connect(Node* root) {  
  
        if (!root) return nullptr;  
  
        Node* leftMost = root;  
  
        while (leftMost->left) {  
  
            Node* currNode = leftMost;  
  
  
            while (currNode) {  
  
                currNode->left->next = currNode->right;  
  
                if (currNode->next) {  
  
                    currNode->right->next = currNode->next->left;  
  
                }  
  
                currNode = currNode->next; }  
  
            leftMost = leftMost->left; }  
  
        return root; } };
```

The screenshot displays a coding platform interface with a dark theme. The top navigation bar includes 'Problem List', 'Run', 'Submit', and a 'Premium' badge. The left sidebar shows the 'Accepted' tab with a submission by 'Rahul Gupta' at Mar 31, 2025 17:45. Performance metrics are shown: Runtime 16 ms (Beats 32.68%) and Memory 19.04 MB (Beats 62.69%). A bar chart visualizes the runtime performance across various test cases. The main editor shows the C++ code for the 'connect' function, which populates the next right pointers in a binary tree. The bottom right section displays 'Testcase 1' with the input array [1, 2, 3, 4, 5, 6, 7] and a corresponding tree diagram with a root node labeled 1.

Problem List < > Run Submit 70 Premium

Description Editorial Solutions Submissions Accepted x

< All Submissions

Accepted 59 / 59 testcases passed

Rahul Gupta submitted at Mar 31, 2025 17:45

Editorial Solution

Runtime

16 ms | Beats 32.68%

Analyze Complexity

Memory

19.04 MB | Beats 62.69%

15%
10%
5%
0%

7ms 12ms 17ms

4ms 6ms 8ms 10ms 12ms 14ms 16ms 18ms

Code C++

```
class Solution {  
public:  
    Node* connect(Node* root) {  
        if (!root) return nullptr;  
        Node* leftMost = root;  
        while (leftMost->left) {  
            Node* currNode = leftMost;  
            while (currNode) {  
                currNode->left->next = currNode->right;  
                if (currNode->next) {  
                    currNode->right->next = currNode->next->left;  
                }  
                currNode = currNode->next; }  
            leftMost = leftMost->left; }  
        return root; } };
```

Saved Ln 9, Col 1

Testcase Test Result

Case 1 Case 2 +

root =

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

1

</> Source