

# AP Assignment 7

Name-> Ishu

UID-> 22BCS15695

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:

- Problem List:** Shows the problem status as 'Accepted' with 39/39 testcases passed. The user 'Rahul Gupta' submitted the solution on Mar 28, 2025 at 12:15.
- Runtime and Memory:** The runtime is 0 ms, beating 100.00% of other solutions. The memory usage is 19.16 MB, beating 13.09% of other solutions.
- Code Editor:** Contains the C++ code for the 'maxDepth' function, which recursively calculates the maximum depth of a binary tree.
- Testcase:** Case 1 shows the input 'root = [3,9,20,null,null,15,7]' and a visual representation of the binary tree structure with a root node 3, left child 9, right child 20, and further children 15 and 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 the following components:

- Problem List:** Navigation tabs for Description, Editorial, Solutions, Submissions, and Accepted.
- Submission Status:** "Accepted" with 86 / 86 testcases passed. Submitted by "Rahul Gupta" on Mar 28, 2025 at 12:38.
- Runtime:** 0 ms, Beats 100.00%.
- Memory:** 21.70 MB, Beats 99.76%.
- Code Editor:** Displays the C++ solution code for the problem.
- Testcase:** Shows "Case 1" with input "root = [2,1,3]" and a diagram of a binary tree with root 2, left child 1, and right child 3.

```
1 class Solution {  
2 public:  
3     bool isValidBST(TreeNode* root) {  
4         return valid(root, LONG_MIN, LONG_MAX);  
5     }  
6 private:  
7     bool valid(TreeNode* node, long minimum, long maximum) {  
8         if (!node) return true;  
9         if (!(node->val > minimum && node->val < maximum)) return false;  
10        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);  
11    }  
12 }  
13 }  
14 }  
15 }
```

Testcase 1: root = [2,1,3]

Diagram: A binary tree with root node 2. Node 2 has a left child node 1 and a right child node 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 corresponding tree diagram with root 1.

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:** Shows the problem name and navigation options.
- Accepted Submissions:** Indicates that 35 out of 35 test cases passed. The user 'Rahul Gupta' submitted the solution on Mar 31, 2025, at 10:42.
- Runtime Performance:** Shows a runtime of 0 ms, which is 100.00% better than other solutions.
- Memory Performance:** Shows a memory usage of 16.99 MB, which is 88.27% better than other solutions.
- Code Editor:** Contains the C++ code for the level order traversal algorithm.
- Testcase:** Shows the input for Case 1: root = [3, 9, 20, null, null, 15, 7].
- Test Result:** Displays a diagram of the binary tree structure for the input: root (3) has a left child (9) and a right child (20). Node 9 has a right child (15). Node 20 has a left child (7).

```
1 class Solution {  
2 public:  
3     vector<vector<int>> levelOrder(TreeNode* root) {  
4         vector<vector<int>>ans;  
5         if(root==NULL)return ans;  
6         queue<TreeNode*>q;  
7         q.push(root);  
8         while(!q.empty()){  
9             int s=q.size();  
10            vector<int>v;  
11            for(int i=0;i<s;i++){  
12                TreeNode *node=q.front();  
13                q.pop();  
14                if(node->left!=NULL)q.push(node->left);  
15                if(node->right!=NULL)q.push(node->right);  
16                v.push_back(node->val);  
17            }  
18            ans.push_back(v);  
19        }  
20        return ans;  
21    }  
22 };
```

## 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 displays a C++ IDE interface for a problem titled "Binary Tree Inorder Traversal".

**Left Panel (Submission Details):**

- Navigation: Description, Editorial, Solutions, Submissions, Accepted (selected).
- 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%.
- Graph: A bar chart showing performance comparison against other solutions.
- Code: C++

**Right Panel (Code Editor):**

```
1 class Solution {  
2 public:  
3     vector<int> inorderTraversal(TreeNode* root) {  
4         vector<int> ans;  
5         if (root == NULL) return ans;  
6         vector<int> left = inorderTraversal(root->left);  
7         ans.insert(ans.end(), left.begin(), left.end());  
8         ans.push_back(root->val);  
9         vector<int> right = inorderTraversal(root->right);  
10        ans.insert(ans.end(), right.begin(), right.end());  
11        return ans;  
12    }  
13  
14 };
```

**Testcase Section:**

- Case 1: root = [1,null,2,3]
- Case 2: (empty)
- Case 3: (empty)
- Case 4: (empty)

Source: </> Source

## 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 interface with the following components:

- Problem List:** Shows the problem name and status (Accepted).
- Submissions:** Shows the submission status (Accepted) and the user's name (Rahul Gupta).
- Runtime:** Shows the execution time (0 ms) and memory usage (15.21 MB).
- Code:** Displays the C++ code for the zigzag level order traversal.
- Testcase:** Shows the input for Case 1: `[3,9,20,null,null,15,7]`.
- Test Result:** Shows the output for Case 1: `[3,9,20,null,null,15,7]`.

## 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' with a green checkmark. It includes the user's name 'Rahul Gupta', the submission time 'submitted at Apr 03, 2025 10:45', and a 'Solution' button.
- Runtime and Memory:** Performance metrics showing '1 ms' runtime (Beats 81.54%) and '27.40 MB' memory (Beats 73.64%).
- Code Editor:** A C++ code editor showing the implementation of the buildTree function. The code uses an unordered\_map to store the index of each element in the inorder array and recursively constructs the binary tree.
- Testcase:** A section for testing the solution with input arrays: inorder = [9, 3, 15, 20, 7] and postorder = [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 Performance:** A bar chart showing "0 ms" runtime, which "Beats 100.00%" of other solutions. A link to "Analyze Complexity" is provided.
- Memory Performance:** A bar chart showing "24.36 MB" memory usage, which "Beats 67.74%" of other solutions.
- Code Editor:** A C++ code editor showing the implementation of the `kthSmallest` function. The code uses a stack to traverse the BST in-order until the kth element is found.
- 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 with root 3, left child 1, and right child 4 (which has a left child 2).
- Source:** A link to view the source code.

### 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 < > <img alt="refresh icon" data-bbox="178 581 188 591"/> Run Submit <img alt="share icon" data-bbox="541 581 551 591"/> <img alt="copy icon" data-bbox="561 581 571 591"/> 70 Premium

Description Editorial Solutions Submissions Accepted x

< All Submissions <img alt="share icon" data-bbox="391 621 401 631"/>

Accepted 59 / 59 testcases passed

Rahul Gupta submitted at Mar 31, 2025 17:45

Editorial Solution

Runtime <img alt="info icon" data-bbox="361 686 371 696"/>

16 ms | Beats 32.68%

Analyze Complexity

Memory

19.04 MB | Beats 62.69% <img alt="green bar" data-bbox="161 761 171 771"/>

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 + <img alt="refresh icon" data-bbox="931 851 941 861"/>

root =

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

1

</ Source <img alt="info icon" data-bbox="461 946 471 956"/>