# ASSIGNMENT-7

| | |
|---|---|
| **Student Name:** Ananya Kanwar | **UID:** 22BCS10579 |
| **Branch:** CSE | **Section/Group:** 22BCS_IOT-609/B |
| **Semester:** 6th | **Subject Code:** 22CSP-351 |
| **Subject Name**: Advanced Programming Lab-II | **Date:** 28-3-25 |

1. **Problem Statement :**

**Maximum Depth of Binary Tree**

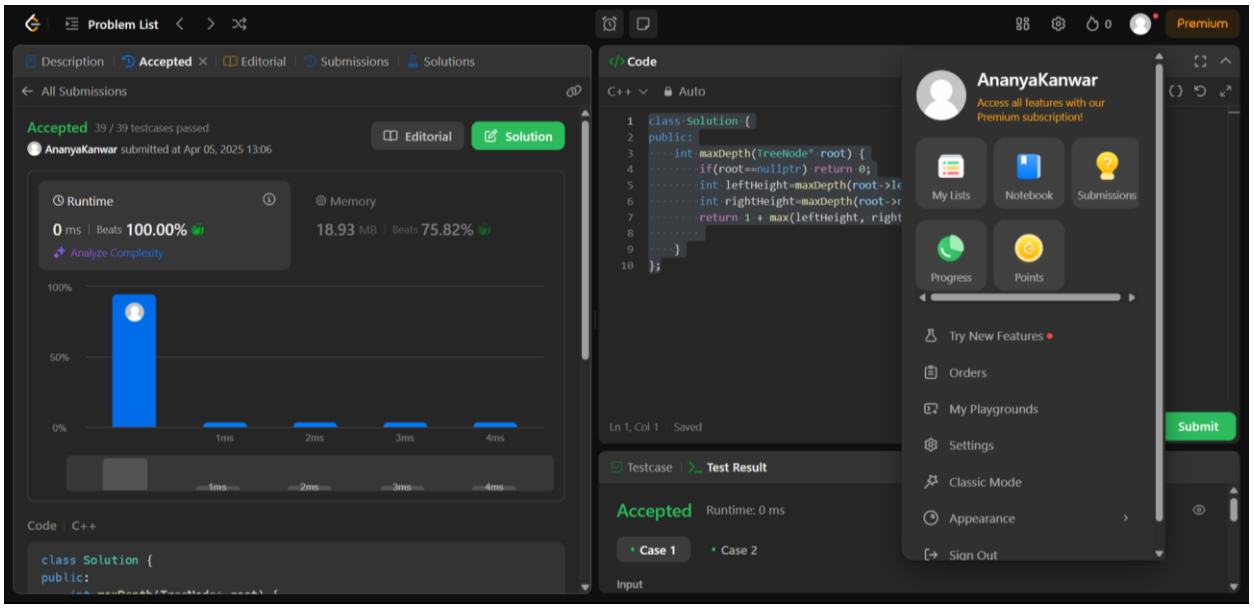
https://leetcode.com/problems/maximum-depth-of-binary-tree/

**Code:**

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

    }
};
```

**OUTPUT:**



2. **Problem Statement:**
   **Validate Binary Search Tree**
   https://leetcode.com/problems/validate-binary-search-tree/

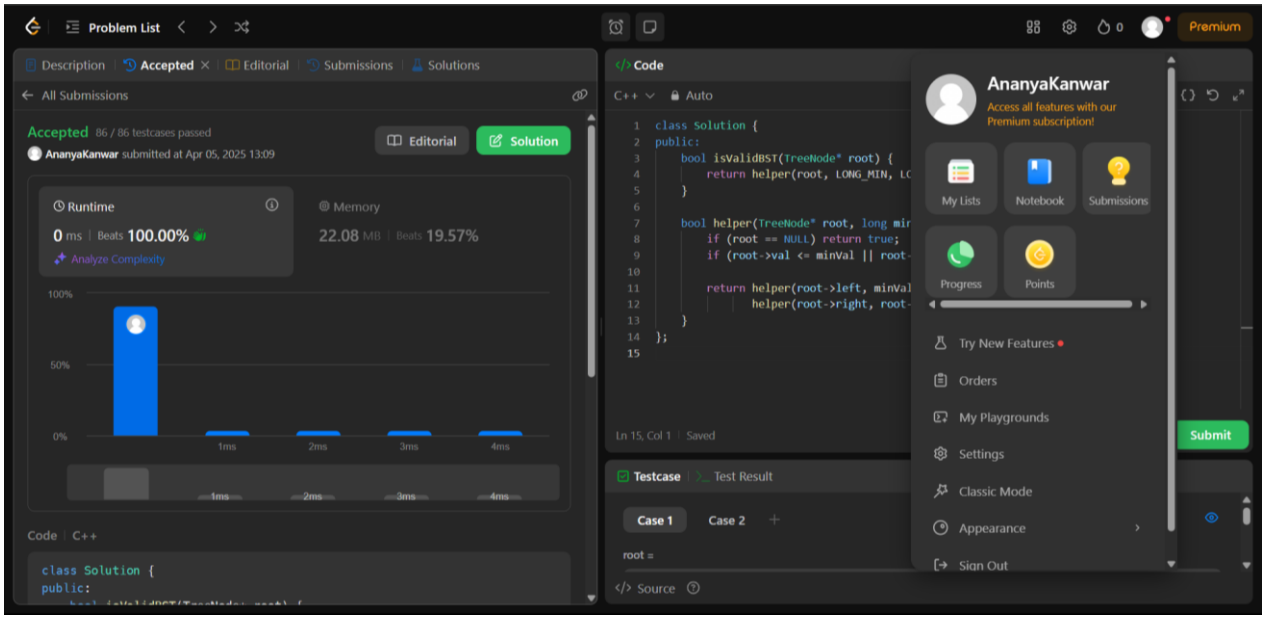
**Code:**

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return helper(root, LONG_MIN, LONG_MAX);
    }

    bool helper(TreeNode* root, long minVal, long maxVal) {
        if (root == NULL) return true;
        if (root->val <= minVal || root->val >= maxVal) return false;

        return helper(root->left, minVal, root->val) &&
               helper(root->right, root->val, maxVal);
    }
};
```

**OUTPUT:**



**3. Problem Statement:**

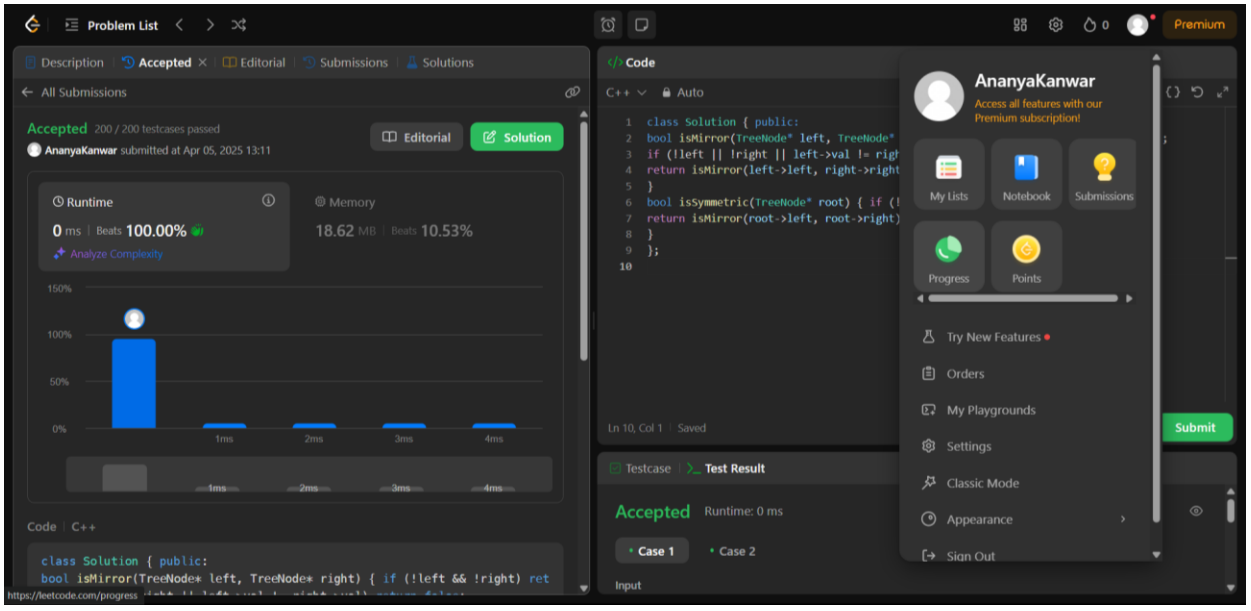
**Symmetric Tree**

https://leetcode.com/problems/symmetric-tree/

**CODE:**

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

**OUTPUT:**



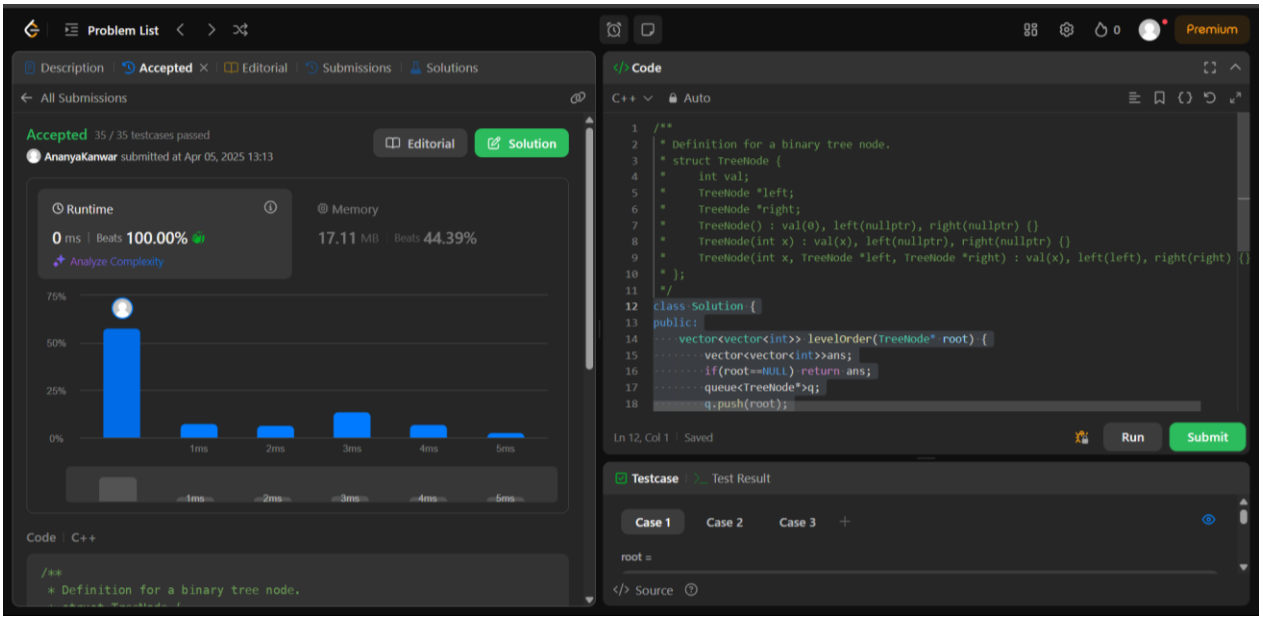
4. **Problem Statement:**
   **Binary Tree Level Order Traversal**
   https://leetcode.com/problems/binary-tree-level-order-traversal/

   **CODE:**

```
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 size=q.size();
            vector<int>level;
            for(int i=0;i<size;i++){
                TreeNode*node=q.front();
                q.pop();
                if(node->left !=NULL) q.push(node->left);
                if(node->right !=NULL) q.push(node->right);
                level.push_back(node->val);
            }
            ans.push_back(level);
        }

        return ans;
    }
};
```

**OUTPUT:**




5. **Problem Statement:**

**Convert Sorted Array to Binary Search Tree**

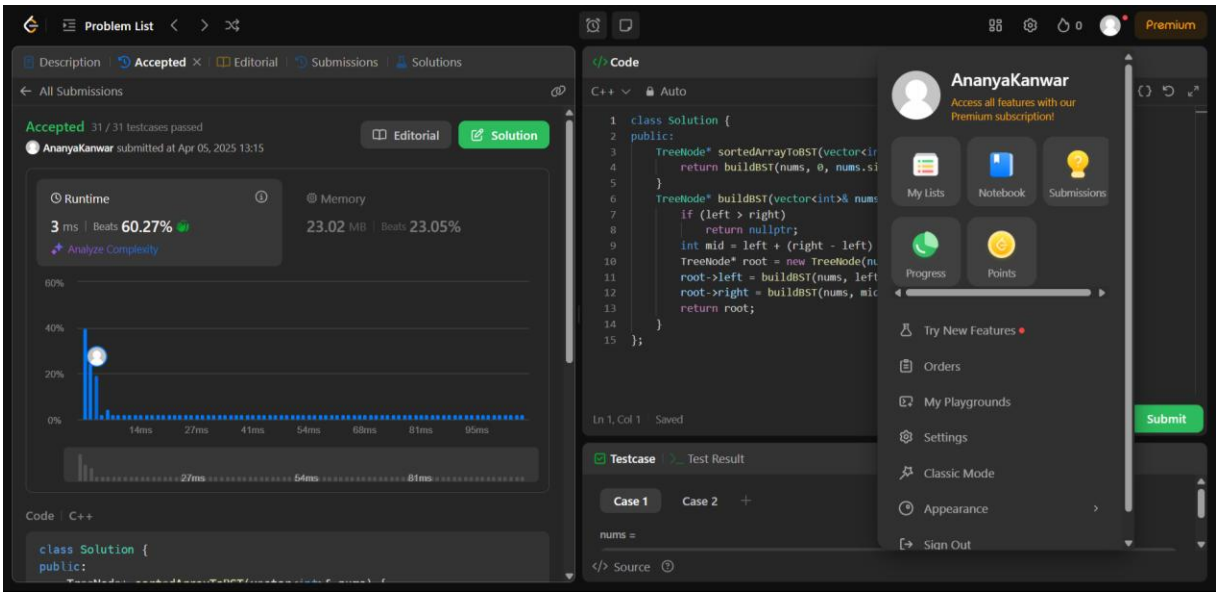
https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

**CODE:**

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

};

**OUTPUT:**




## 6. Problem Statement:

**Binary Tree Inorder Traversal**

https://leetcode.com/problems/binary-tree-inorder-traversal/

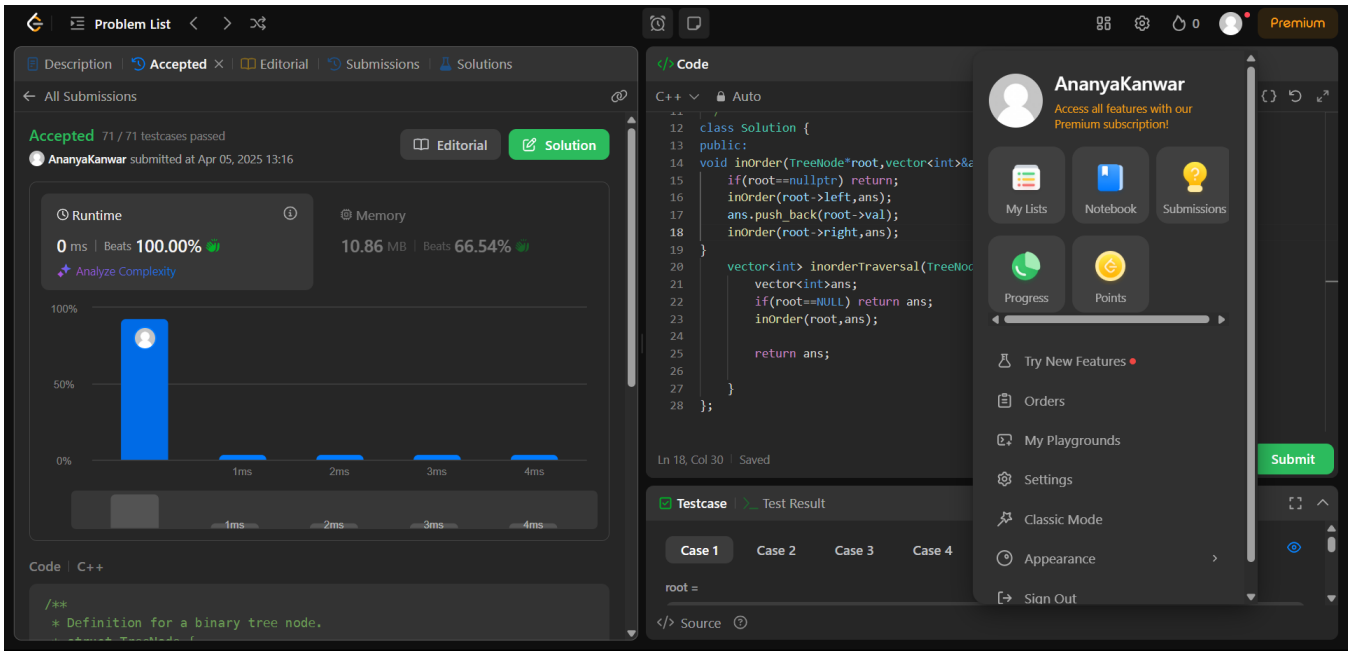
**CODE:**

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

        return ans;

    }
};
```

**OUTPUT:**




7. **Problem Statement:**
   **Binary Zigzag Level Order Traversal**
   https://leetcode.com/problems/binary-zigzag-level-order-traversal/

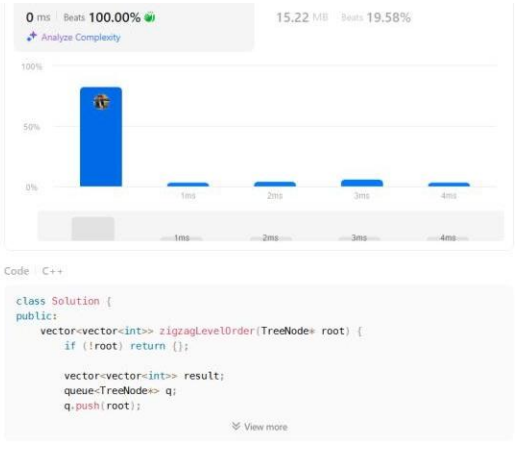
   **CODE:**

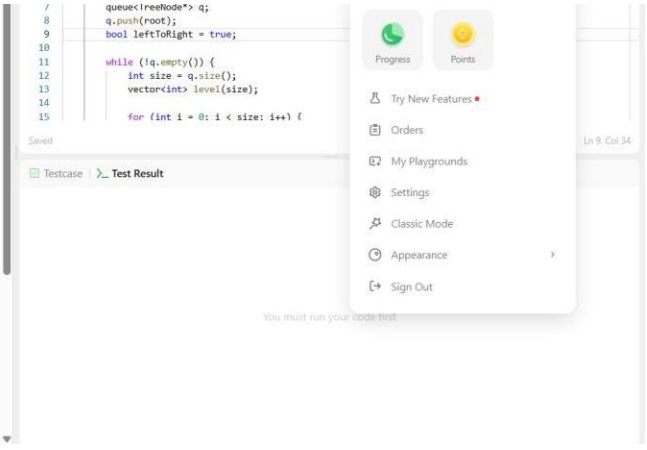
```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        if (!root) return {};

        vector<vector<int>> result;
        queue<TreeNode*> q;
        q.push(root);
```

```
        bool leftToRight = true;
        while (!q.empty()) {
            int size = q.size();
            vector<int> level(size);
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                int index = leftToRight ? i : (size - 1 - i);
                level[index] = node->val;
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(move(level));
            leftToRight = !leftToRight;
        }
        return result;
    }
};
```

**OUTPUT:**



8. **Problem Statement:**
**Construct Binary Tree from Inorder and Postorder Traversal**
https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/

**CODE:**

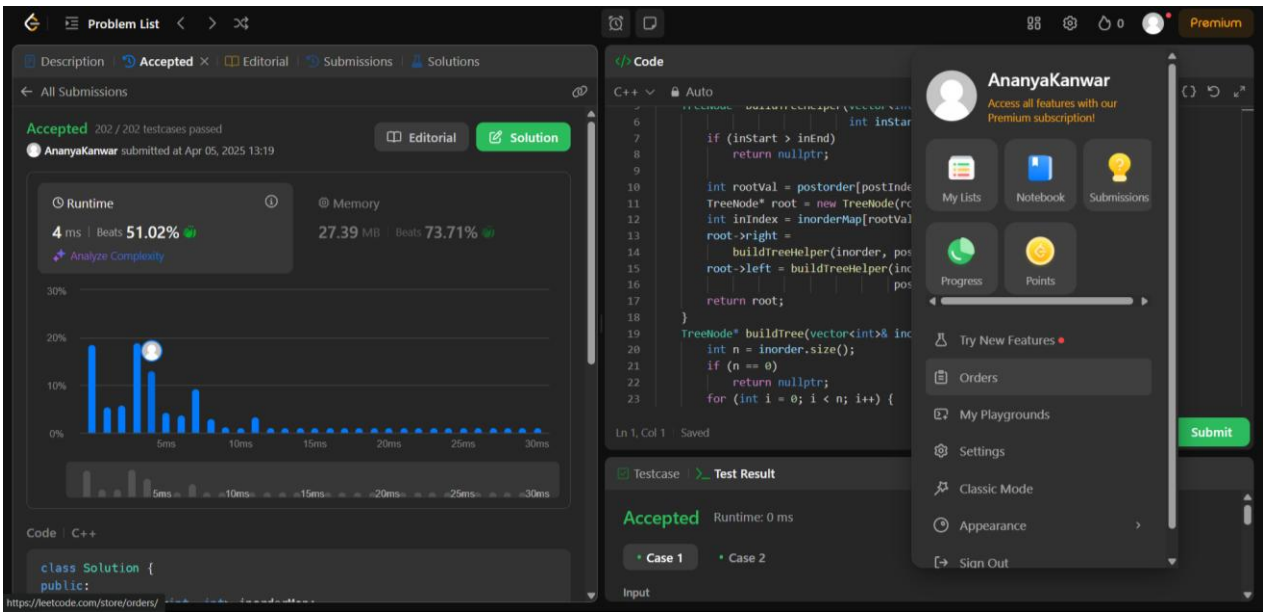
```
class Solution {
public:
    unordered_map<int, int> inorderMap;

    TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder,
                              int inStart, int inEnd, int& postIndex) {
        if (inStart > inEnd) return nullptr;

        int rootVal = postorder[postIndex--];
        TreeNode* root = new TreeNode(rootVal);
        int inIndex = inorderMap[rootVal];
        root->right = buildTreeHelper(inorder, postorder, inIndex + 1, inEnd, postIndex);
        root->left = buildTreeHelper(inorder, postorder, inStart, inIndex - 1, postIndex);
        return root;
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int n = inorder.size();
        if (n == 0) return nullptr;
        for (int i = 0; i < n; i++) {
            inorderMap[inorder[i]] = i;
        }
```

```
        int postIndex = n - 1;
        return buildTreeHelper(inorder, postorder, 0, n - 1, postIndex);
    }
};
```

**OUTPUT:**




9. **Problem Statement:**
   **Kth Smallest element in a BST**
   https://leetcode.com/problems/kth-smallest-element-in-a-bst/

**CODE:**

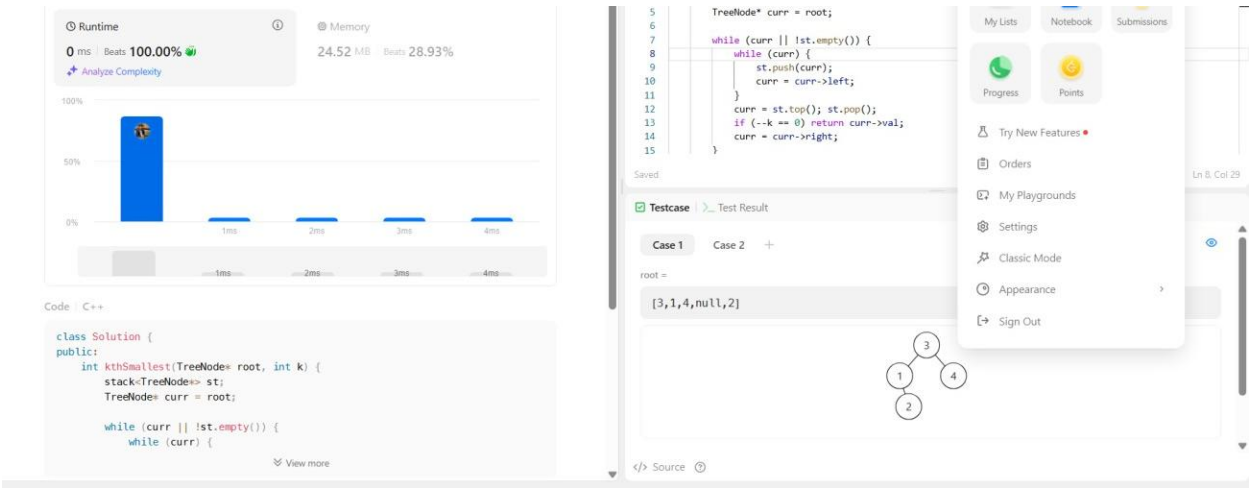
```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> st;
        TreeNode* curr = root;

        while (curr || !st.empty()) {
```

```
            while (curr) {
                st.push(curr);
                curr = curr->left;
            }
            curr = st.top(); st.pop();
            if (--k == 0) return curr->val;
            curr = curr->right;
        }

        return -1;
    }
};
```

**OUTPUT:**




## 10. Problem Statement

**Populating Next Right Pointers in Each Node**

https://leetcode.com/problems/populating-next-right-pointers-in-each-node/

**CODE:**

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root) return nullptr;
        queue<Node*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                Node* node = q.front();
                q.pop();
                if (i < size - 1) node->next = q.front();
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return root;
    }
};
```

**OUTPUT:**