

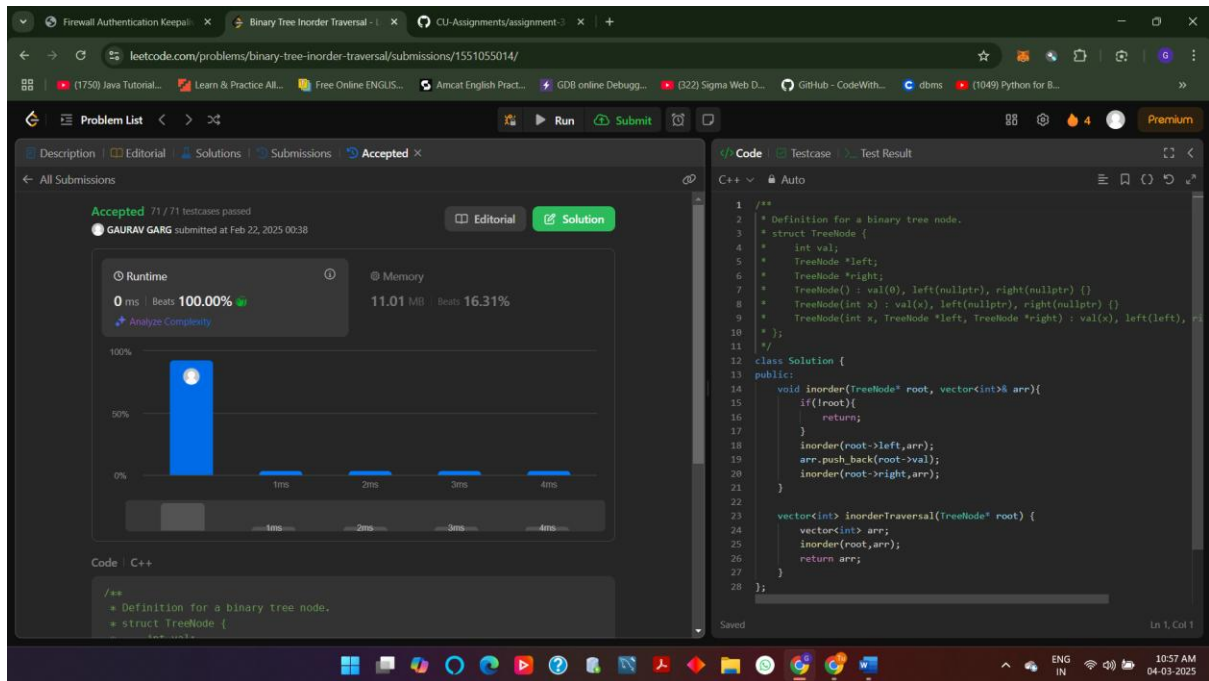
ASSIGNMENT 3

GAURAV GARG

22BCS15846

603/A

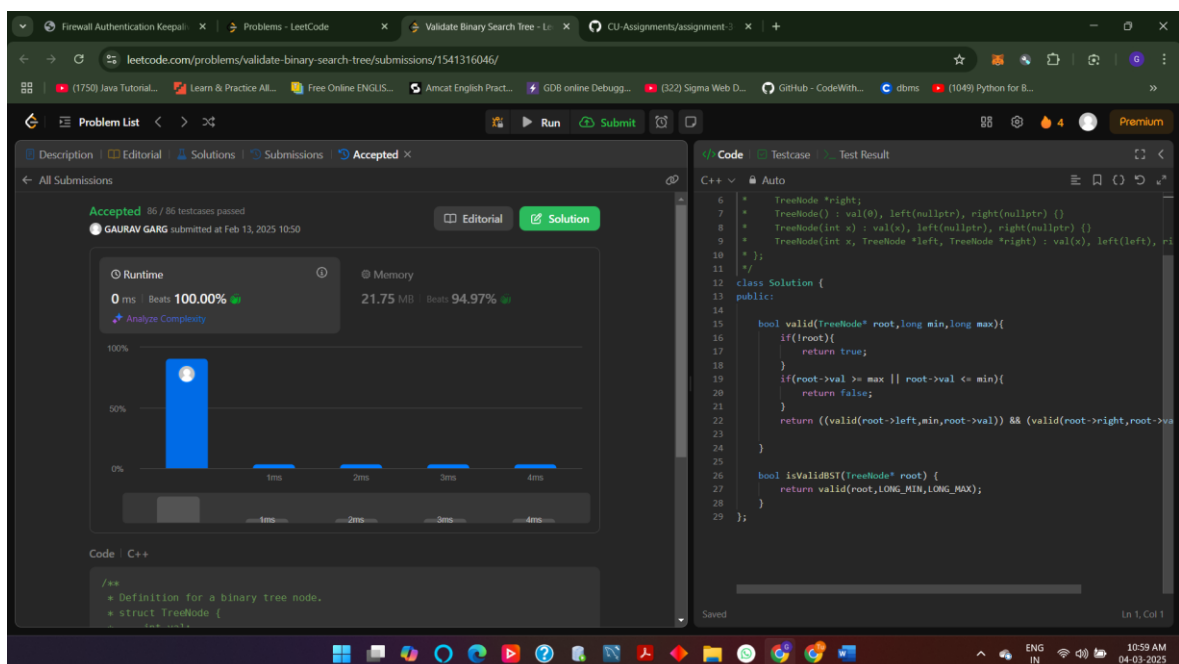
94. Binary Tree Inorder Traversal



The screenshot shows the LeetCode interface for the problem "Binary Tree Inorder Traversal". The submission is by GAURAV GARG, submitted on Feb 22, 2025, 00:38. The status is "Accepted" with 71/71 testcases passed. The runtime is 0 ms, beating 100.00% of submissions, and the memory is 11.01 MB, beating 16.31% of submissions. The code is written in C++ and implements an inorder traversal using a recursive function.

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 class Solution {
13 public:
14     void inorder(TreeNode* root, vector<int>& arr){
15         if(!root){
16             return;
17         }
18         inorder(root->left, arr);
19         arr.push_back(root->val);
20         inorder(root->right, arr);
21     }
22
23     vector<int> inorderTraversal(TreeNode* root) {
24         vector<int> arr;
25         inorder(root, arr);
26         return arr;
27     }
28 };
```

98. Validate Binary Search Tree



The screenshot shows the LeetCode interface for the problem "Validate Binary Search Tree". The submission is by GAURAV GARG, submitted on Feb 13, 2025, 10:50. The status is "Accepted" with 86/86 testcases passed. The runtime is 0 ms, beating 100.00% of submissions, and the memory is 21.75 MB, beating 94.97% of submissions. The code is written in C++ and implements a validation function for a Binary Search Tree using a recursive approach.

```
6 * struct TreeNode {
7 *     int val;
8 *     TreeNode *left;
9 *     TreeNode *right;
10 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
11 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
13 * };
14
15 class Solution {
16 public:
17     bool valid(TreeNode* root, long min, long max){
18         if(!root){
19             return true;
20         }
21         if(root->val >= max || root->val <= min){
22             return false;
23         }
24         return ((valid(root->left, min, root->val)) && (valid(root->right, root->val, max)));
25     }
26
27     bool isValidBST(TreeNode* root) {
28         return valid(root, LONG_MIN, LONG_MAX);
29     }
30 };
```

230. Kth Smallest Element in a BST

The screenshot shows a C++ solution for the problem "Kth Smallest Element in a BST". The code is as follows:

```
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}

class Solution {
public:
    int count = 0;
    int result = -1;
    void answer(TreeNode* root, int k) {
        if(!root) return;
        answer(root->left, k);
        count++;
        if(count == k) {
            result = root->val;
            return;
        }
        answer(root->right, k);
    }
    int kthSmallest(TreeNode* root, int k) {
        answer(root, k);
        return result;
    }
};
```

The submission is accepted, with 93/93 testcases passed. The runtime is 0 ms, beating 100.00% of submissions. The memory usage is 24.27 MB, beating 90.11% of submissions. A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming.

102. Binary Tree Level Order Traversal

The screenshot shows a Java solution for the problem "Binary Tree Level Order Traversal". The code is as follows:

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> q = new LinkedList<>();
        List<List<Integer>> finalAns = new ArrayList<>();
        if(root==null){
            return finalAns;
        }
        q.add(root);
        while(!q.isEmpty()){
            int levels = q.size();
            List<Integer> sublevels = new ArrayList<>();
            for(int i=0; i<levels; i++){
                if(q.peek().left!=null){
                    q.add(q.peek().left);
                }
                if(q.peek().right!=null){
                    q.add(q.peek().right);
                }
                sublevels.add(q.remove().val);
            }
            finalAns.add(sublevels);
        }
        return finalAns;
    }
}
```

The submission is accepted, with 35/35 testcases passed. The runtime is 1 ms, beating 89.76% of submissions. The memory usage is 44.88 MB, beating 87.59% of submissions. A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming.

987. Vertical Order Traversal of a Binary Tree

The screenshot shows a LeetCode submission for the problem "Vertical Order Traversal of a Binary Tree". The submission is accepted, with a runtime of 3 ms and memory usage of 42.78 MB. A bar chart shows the runtime distribution across different time intervals. The code editor displays the following Java code:

```
class Solution {  
    Map<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map;  
  
    public List<List<Integer>> verticalTraversal(TreeNode root) {  
        if (root == null)  
            return null;  
        map = new TreeMap<>();  
        dfs(root, 0, 0);  
        List<List<Integer>> res = new LinkedList<>();  
        for (int key : map.keySet()) {  
            List<Integer> list = new LinkedList<>();  
            TreeMap<Integer, PriorityQueue<Integer>> tm = map.get(key);  
            for (int k : tm.keySet()) {  
                PriorityQueue<Integer> pq = tm.get(k);  
                while (!pq.isEmpty()) {  
                    list.add(pq.poll());  
                }  
            }  
            res.add(list);  
        }  
        return res;  
    }  
  
    private void dfs(TreeNode root, int index, int level) {  
        if (root == null)  
            return;  
        map.putIfAbsent(index, new TreeMap<>());  
        map.get(index).putIfAbsent(level, new PriorityQueue<>());  
        map.get(index).get(level).add(root.val);  
        dfs(root.left, index - 1, level + 1);  
        dfs(root.right, index + 1, level + 1);  
    }  
}
```