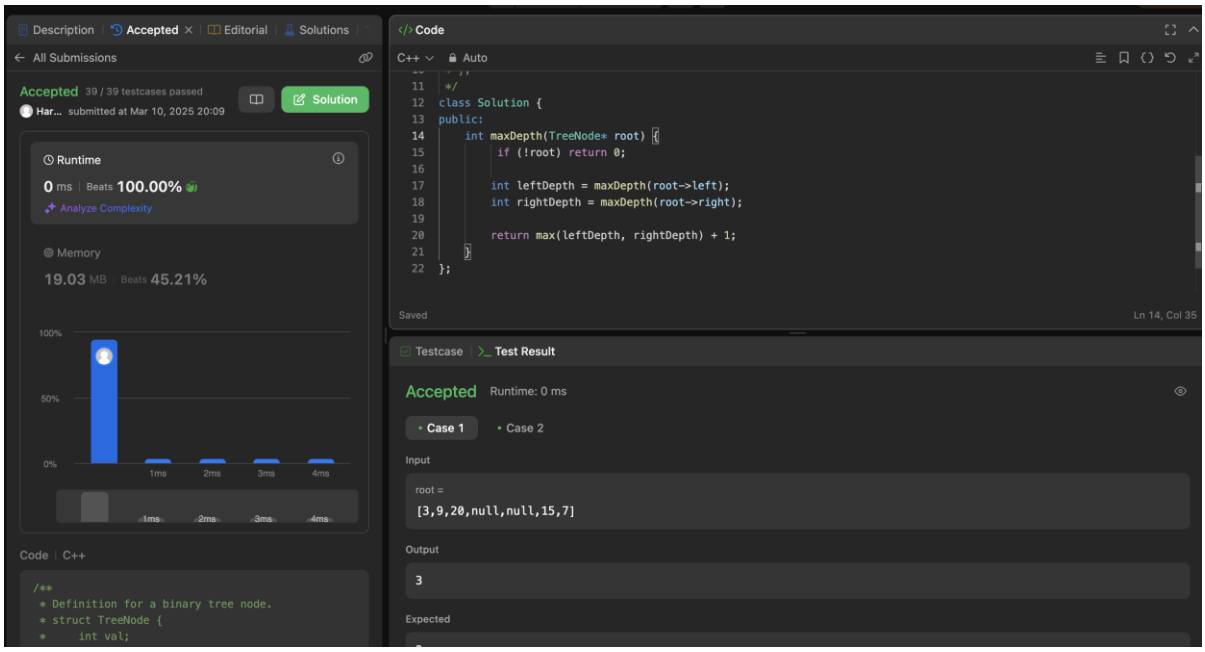
Ap-Assignment 5

Name: Harmandeep Singh **UID:**22BCS14975

Leetcode Profile: https://leetcode.com/u/Harman001/

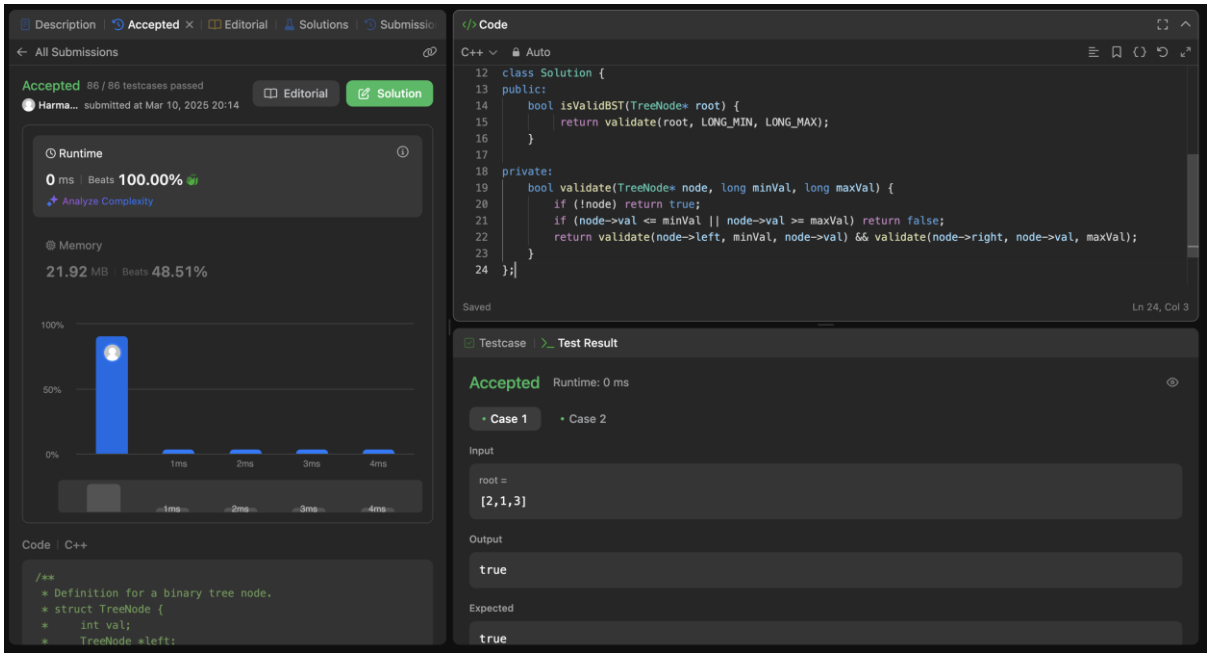
Ques 1:Max depth of binary trees

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0; // Base case: if the tree is empty

        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);

        return max(leftDepth, rightDepth) + 1;
    }
};
```



Ques2: Validate Binary Search tree

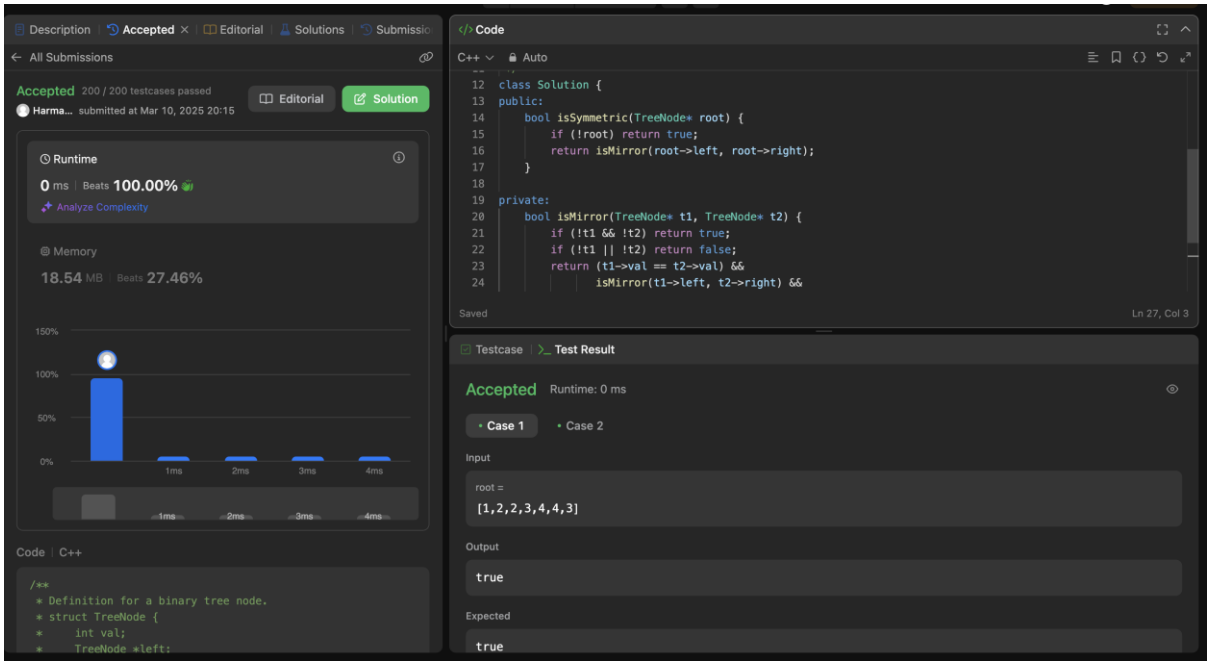
```
class Solution {
public:
bool isValidBST(TreeNode* root) {
return validate(root, LONG_MIN, LONG_MAX);
}

private:
bool validate(TreeNode* node, long minVal, long maxVal) {
if (!node) return true;
if (node->val <= minVal || node->val >= maxVal) return false;
return validate(node->left, minVal, node->val) && validate(node->right, node->val, maxVal);
}
};
```



Ques 3:Symmetric tree

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

private:
bool isMirror(TreeNode* t1, TreeNode* t2) {
if (!t1 && !t2) return true;
if (!t1 || !t2) return false;
return (t1->val == t2->val) &&
isMirror(t1->left, t2->right) &&
isMirror(t1->right, t2->left);
```

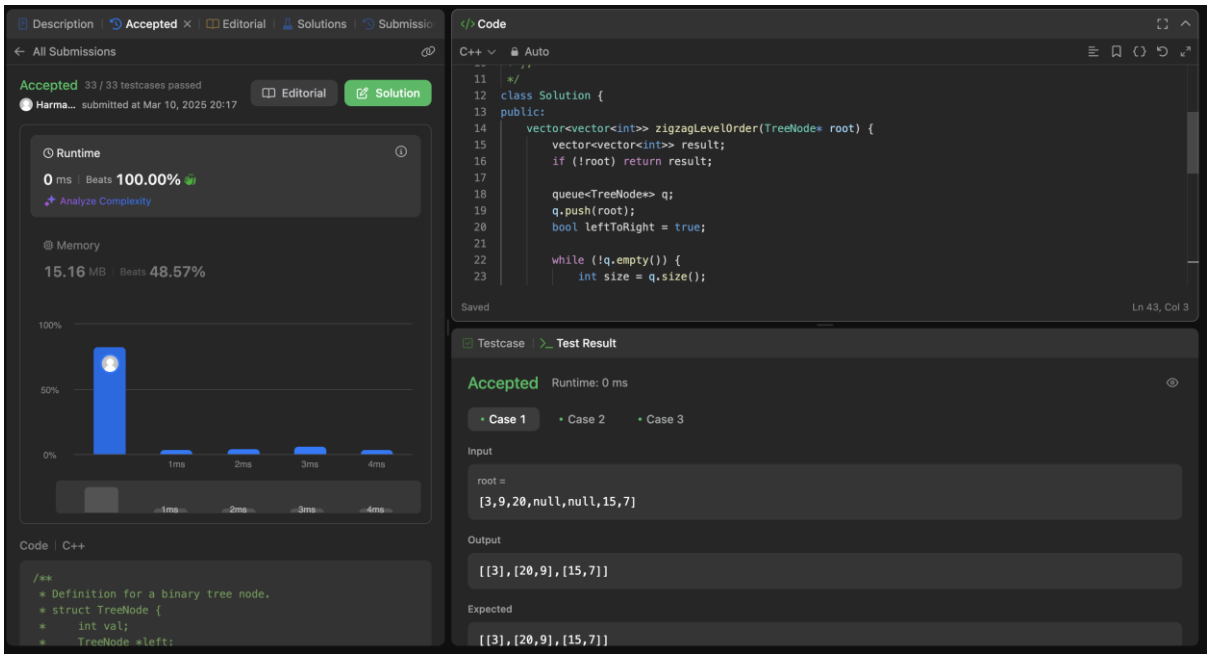
```
}
};
```



Ques4:Binary tree Zigzag level order traversal

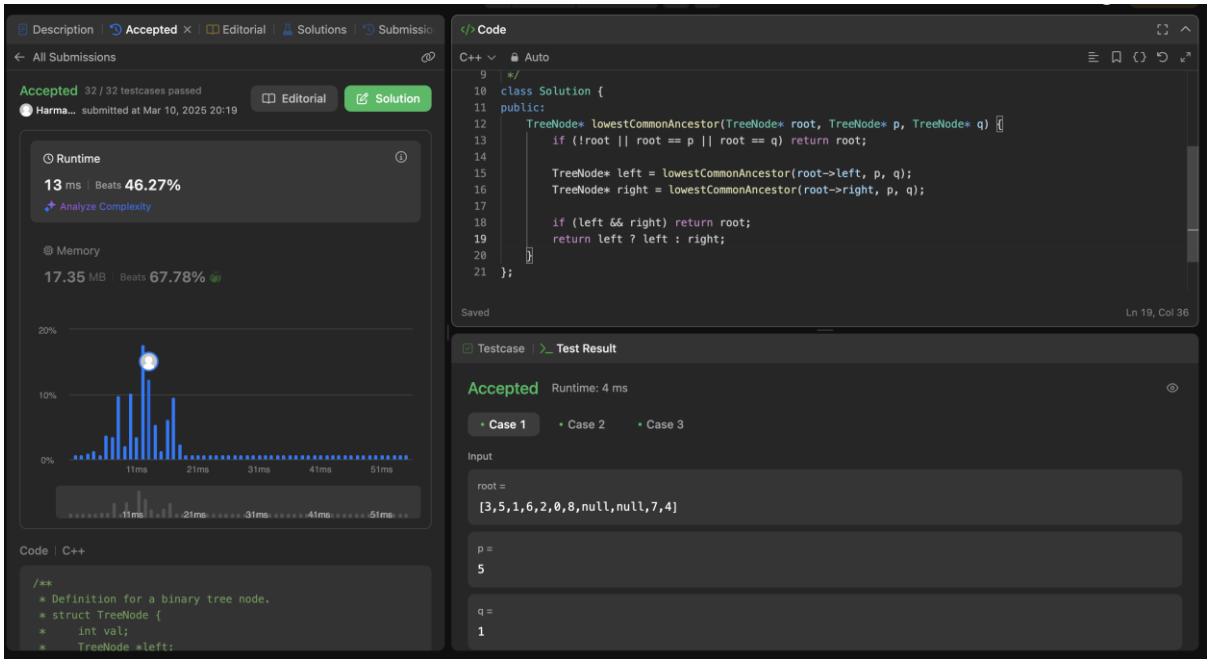
```
class Solution {
public:
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
vector<vector<int>> result;
if (!root) return result;

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(level);
leftToRight = !leftToRight;
}

return result;
}
};
```



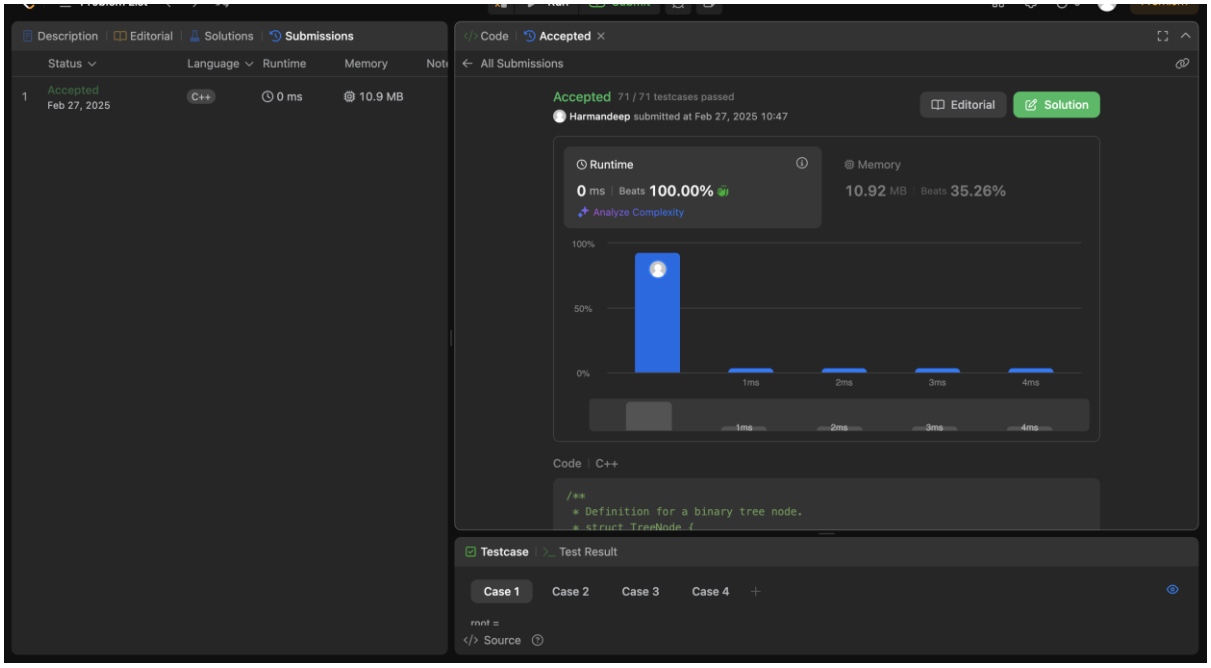
Ques 5: Lowest Common ancestors of Binary tree

```
class Solution {
public:
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
if (!root || root == p || root == q) return root;
TreeNode* left = lowestCommonAncestor(root->left, p, q);
TreeNode* right = lowestCommonAncestor(root->right, p, q);
if (left && right) return root;
return left ? left : right;
}
};
```



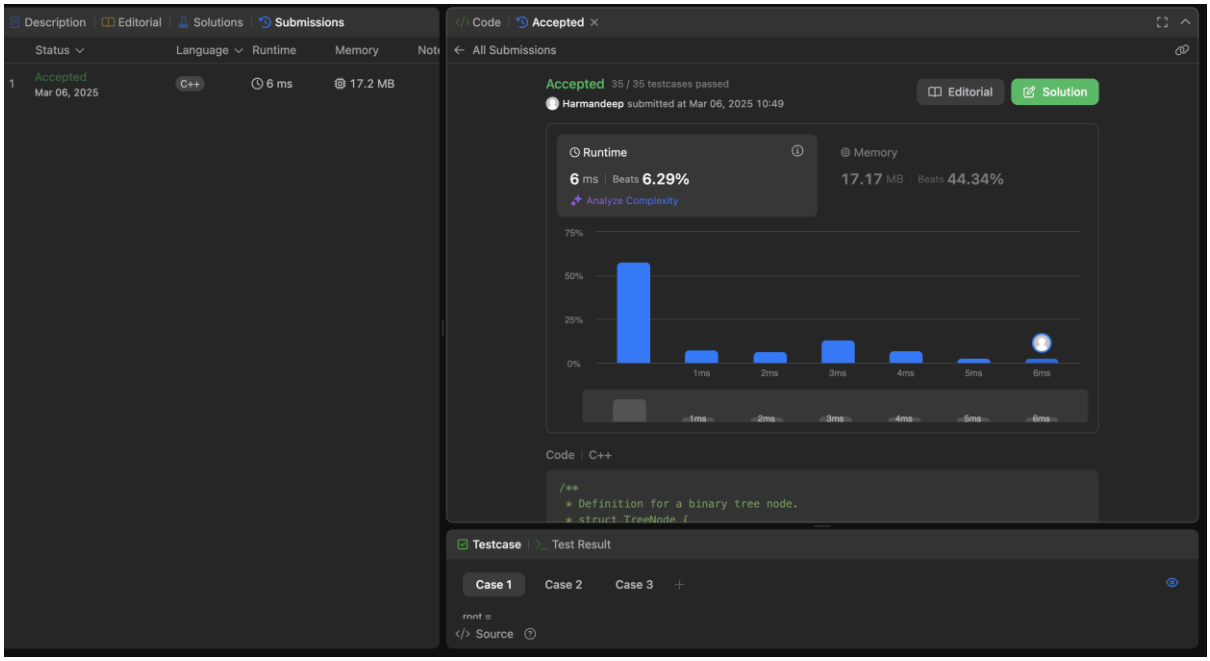
Ques: Binary tree Inorder traversal

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

vector<int> inorderTraversal(TreeNode* root) {
vector<int> result;
inorder(root, result);
return result;
}
};
```



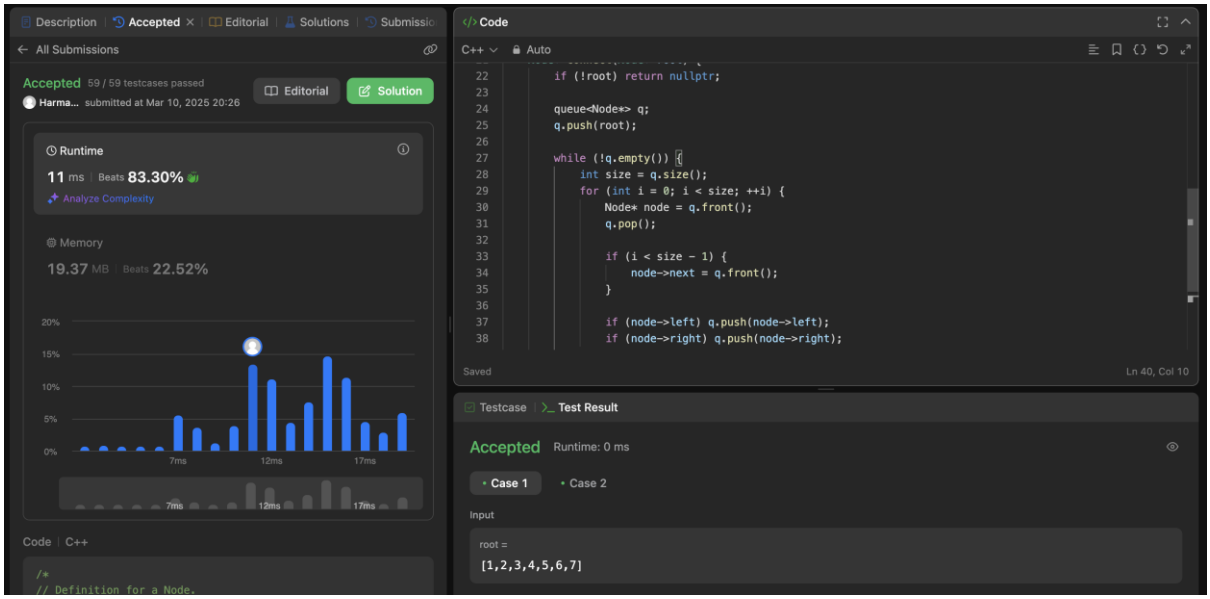
Ques : Binary tree Level order traversal

```
class Solution {
public:
vector<vector<int>> levelOrder(TreeNode* root) {
vector<vector<int>>res;
if(!root) return res;
queue<TreeNode*>q;
q.push(root);
while(!q.empty()){
int n=q.size();
vector<int>ans;
while(n--){
TreeNode*temp=q.front();
q.pop();
ans.push_back(temp->val);
if(temp->left) q.push(temp->left);
if(temp->right)q.push(temp->right);
}res.push_back(ans);
}
return res;


}
};
```



Ques : Population next right Pointer in each node

```
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;
}
};
```



Ques :Sum of the left leaves

```
class Solution {
public:
int sumOfLeftLeaves(TreeNode* root) {
if (!root) return 0;
int sum = 0;
if (root->left && !root->left->left && !root->left->right) {
sum += root->left->val;
}
return sum + sumOfLeftLeaves(root->left) + sumOfLeftLeaves(root->right);
}
};
```