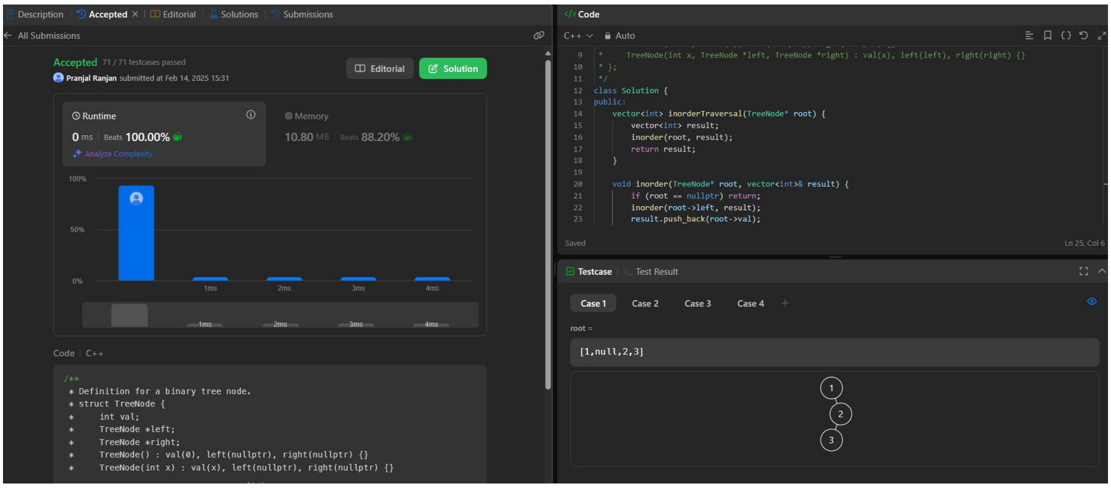
94.BINARY TREE INORDER TRAVERSAL

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     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:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }

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

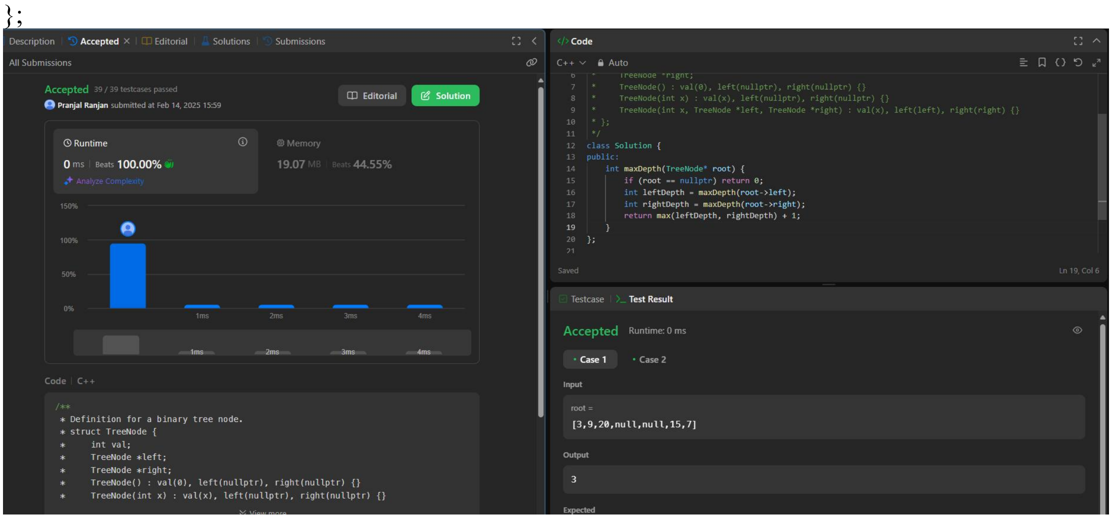


104. MAXIMUM DEPTH OF A BIANRY TREE

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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:
bool isSymmetric(TreeNode* root) {
return isMirror(root, root);
}

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

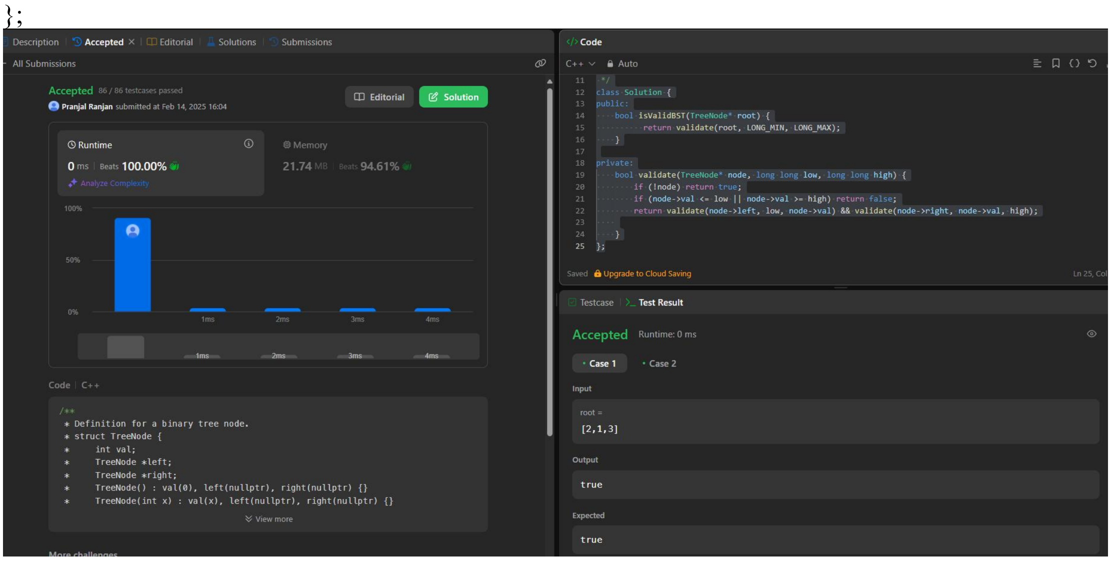
```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
* };
*/
class Solution {
public:
int maxDepth(TreeNode* root) {
if (root == nullptr) return 0;
int leftDepth = maxDepth(root->left);
int rightDepth = maxDepth(root->right);
return max(leftDepth, rightDepth) + 1;
}
};
```



98. VALIDATE BINARY SEARCH TREE

```
/**
* Definition for a binary tree node.
* struct TreeNode {
```

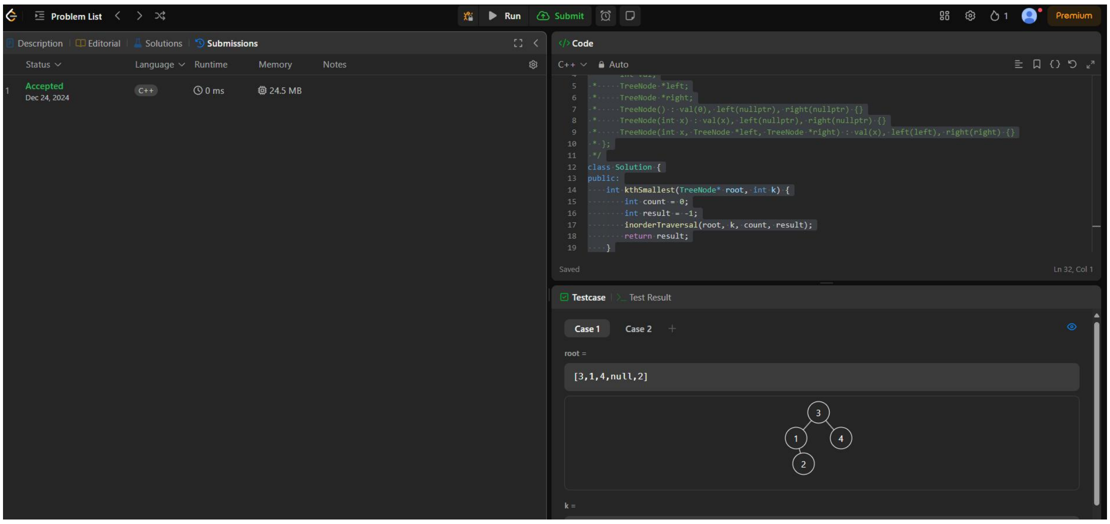
```
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
* };
*/
class Solution {
public:
bool isValidBST(TreeNode* root) {
return validate(root, LONG_MIN, LONG_MAX);
}

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

}
};
```



230.<u>KTH SMALLEST ELEMENT IN BST</u>

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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 kthSmallest(TreeNode* root, int k) {
int count = 0;
int result = -1;
```

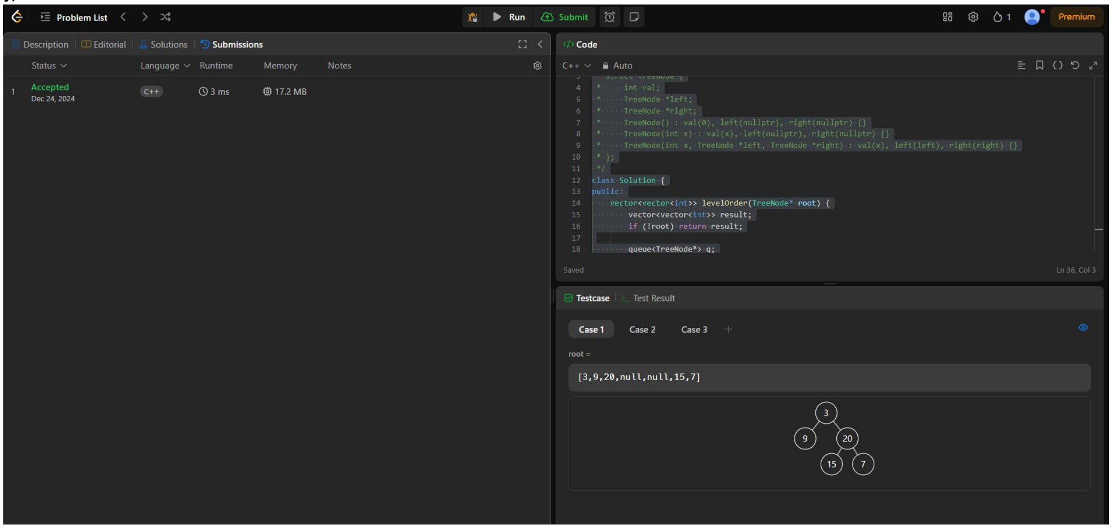
```
inorderTraversal(root, k, count, result);
return result;
}

void inorderTraversal(TreeNode* root, int k, int& count, int& result) {
if (root == nullptr) return;
inorderTraversal(root->left, k, count, result);
count++;
if (count == k) {
result = root->val;
return;
}
inorderTraversal(root->right, k, count, result);
}
};
```



107. BINARY TREE LEVEL ORDER TRAVERSAL II

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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:
vector<vector<int>> levelOrder(TreeNode* root) {
vector<vector<int>> result;
if (!root) return result;

queue<TreeNode*> q;
q.push(root);

while (!q.empty()) {
int levelSize = q.size();
```

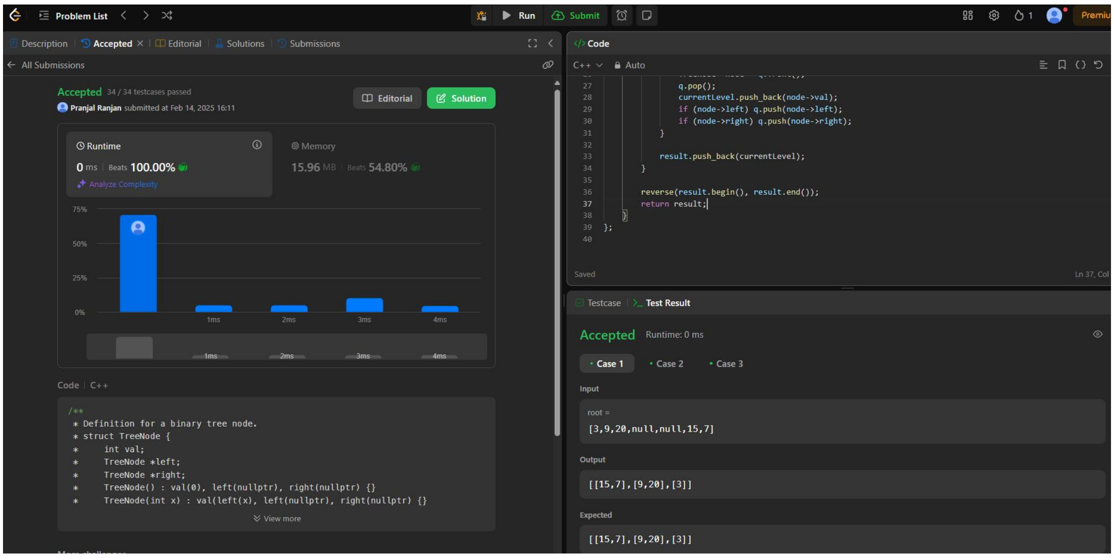
```
vector<int> level;
for (int i = 0; i < levelSize; ++i) {
TreeNode* node = q.front();
q.pop();
level.push_back(node->val);
if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
result.push_back(level);
}

return result;
}
};
```



102. BINARY TREE LEVEL ORDER TRAVERSAL

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     TreeNode(int x) : val(left(x), left(nullptr), right(nullptr) {}
*     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
* };
*/
class Solution {
public:
vector<vector<int>> levelOrderBottom(TreeNode* root) {
vector<vector<int>> result;
if (!root) return result;

queue<TreeNode*> q;
q.push(root);

while (!q.empty()) {
int levelSize = q.size();
vector<int> currentLevel;
```

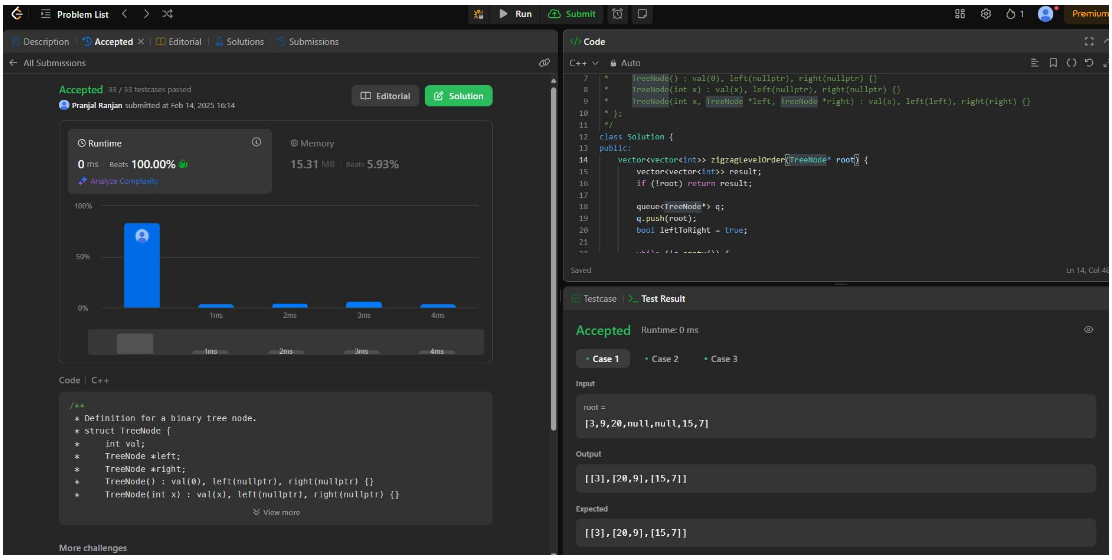
```
for (int i = 0; i < levelSize; ++i) {
TreeNode* node = q.front();
q.pop();
currentLevel.push_back(node->val);
if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}

result.push_back(currentLevel);
}

reverse(result.begin(), result.end());
return result;
}
};
```



103. BINARY TREE ZIGZAG LEVEL ORDER TRAVERSAL

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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:
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
vector<vector<int>> result;
if (!root) return result;

queue<TreeNode*> q;
q.push(root);
bool leftToRight = true;
```

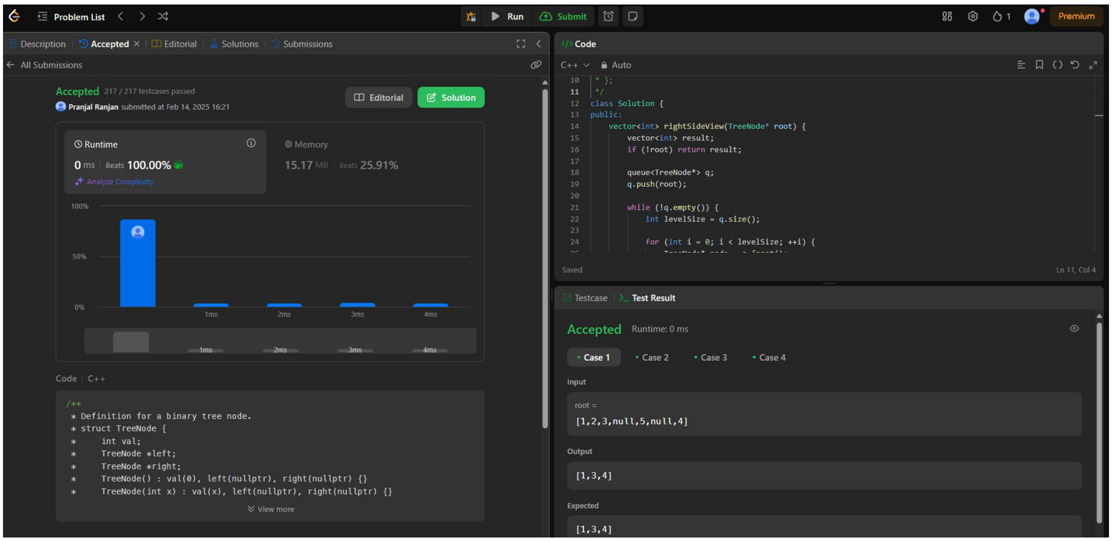
```
while (!q.empty()) {
int levelSize = q.size();
vector<int> currentLevel(levelSize);

for (int i = 0; i < levelSize; ++i) {
TreeNode* node = q.front();
q.pop();

int index = leftToRight ? i : (levelSize - 1 - i);
currentLevel[index] = node->val;

if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}

result.push_back(currentLevel);
leftToRight = !leftToRight;
}

return result;
}
};
```



199. <u>BINARY TREE RIGHT SSIDE VIEW</u>

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
* };
*/
class Solution {
```

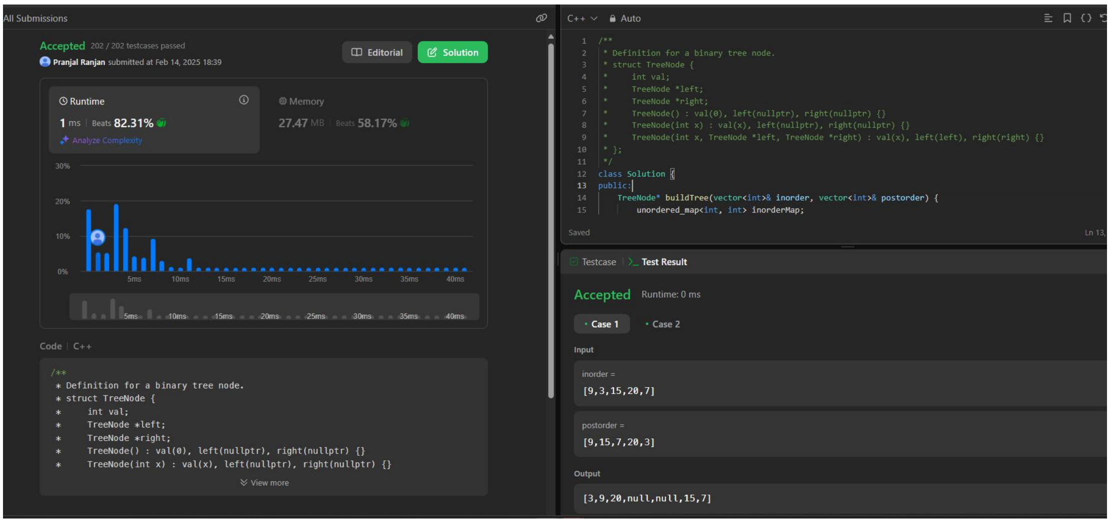
```
public:
vector<int> rightSideView(TreeNode* root) {
vector<int> result;
if (!root) return result;

queue<TreeNode*> q;
q.push(root);

while (!q.empty()) {
int levelSize = q.size();

for (int i = 0; i < levelSize; ++i) {
TreeNode* node = q.front();
q.pop();

// If it's the rightmost element of the level
if (i == levelSize - 1) {
result.push_back(node->val);
}

if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
}

return result;
}
};
```



105. <u>CONSTRUCT BINARY TREE FROM INORDER AND POSTORDER TRAVERSAL</u>

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
```

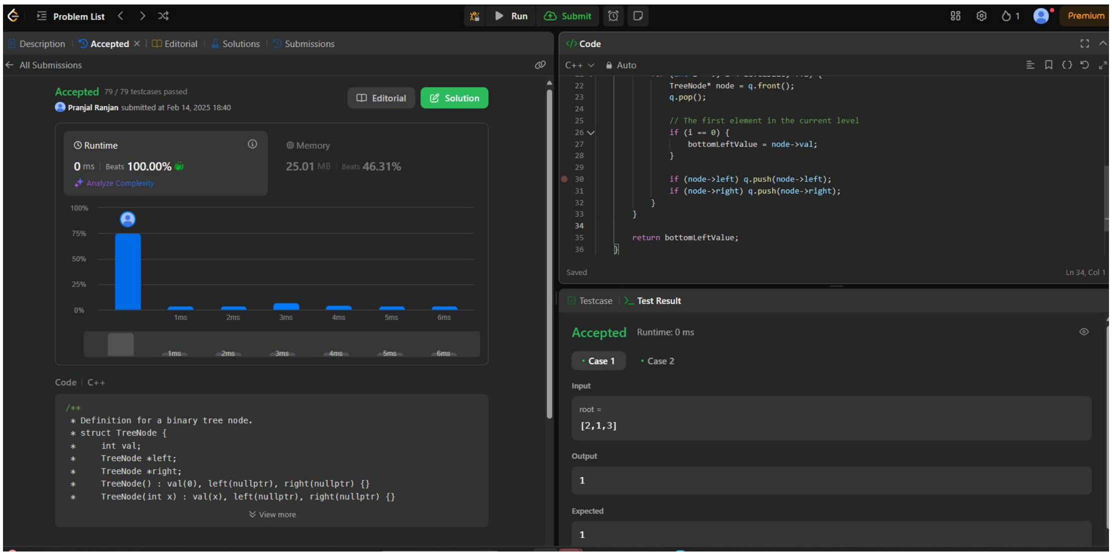
```
*    TreeNode *left;
*    TreeNode *right;
*    TreeNode() : val(0), left(nullptr), right(nullptr) {}
*    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
* };
*/
class Solution {
public:
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
unordered_map<int, int> inorderMap;
for (int i = 0; i < inorder.size(); ++i) {
inorderMap[inorder[i]] = i;
}
return build(inorder, postorder, 0, inorder.size() - 1, 0, postorder.size() - 1, inorderMap);
}

private:
TreeNode* build(vector<int>& inorder, vector<int>& postorder, int inStart, int inEnd, int postStart, int
postEnd, unordered_map<int, int>& inorderMap) {
if (inStart > inEnd || postStart > postEnd) {
return nullptr;
}

TreeNode* root = new TreeNode(postorder[postEnd]);
int inRoot = inorderMap[root->val];
int leftTreeSize = inRoot - inStart;

root->left = build(inorder, postorder, inStart, inRoot - 1, postStart, postStart + leftTreeSize - 1,
inorderMap);
root->right = build(inorder, postorder, inRoot + 1, inEnd, postStart + leftTreeSize, postEnd - 1,
inorderMap);

return root;
}
};
```



513. FIND BOTTOM LEFT TREE VALUE

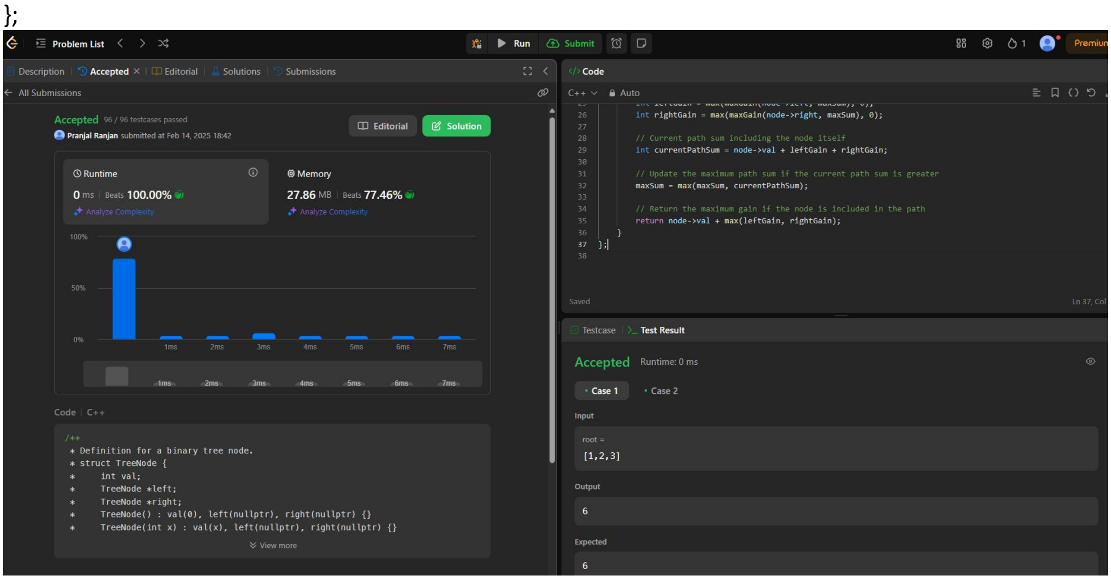
```
/**
```

```
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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 findBottomLeftValue(TreeNode* root) {
int bottomLeftValue = root->val;
queue<TreeNode*> q;
q.push(root);

while (!q.empty()) {
int levelSize = q.size();
for (int i = 0; i < levelSize; ++i) {
TreeNode* node = q.front();
q.pop();

// The first element in the current level
if (i == 0) {
bottomLeftValue = node->val;
}

if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
}

return bottomLeftValue;
}
};
```



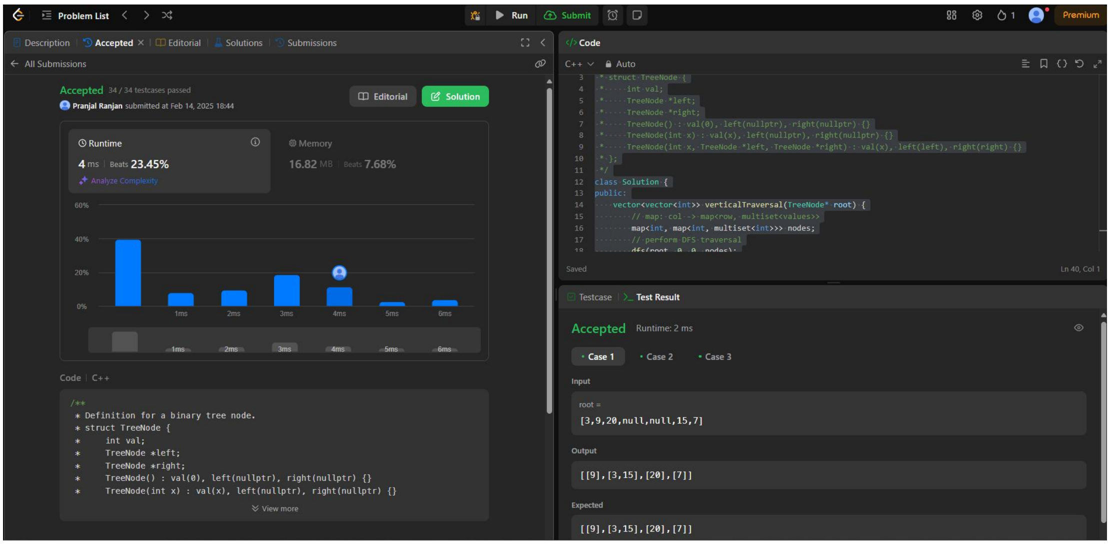
124. <u>BINARY TREE MAXIMUM PATH SUM</u>

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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 maxPathSum(TreeNode* root) {
int maxSum = INT_MIN;
maxGain(root, maxSum);
return maxSum;
}

private:
int maxGain(TreeNode* node, int& maxSum) {
if (!node) return 0;

// Recursively call maxGain for the left and right children
int leftGain = max(maxGain(node->left, maxSum), 0);
int rightGain = max(maxGain(node->right, maxSum), 0);

// Current path sum including the node itself
int currentPathSum = node->val + leftGain + rightGain;

// Update the maximum path sum if the current path sum is greater
maxSum = max(maxSum, currentPathSum);

// Return the maximum gain if the node is included in the path
return node->val + max(leftGain, rightGain);
}
};
```



125. VERTICAL ORDER TRAVERSAL OF A BINARY TREE

```
/**
* Definition for a binary tree node.
* struct TreeNode {
*     int val;
*     TreeNode *left;
*     TreeNode *right;
*     TreeNode() : val(0), left(nullptr), right(nullptr) {}
*     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:
vector<vector<int>> verticalTraversal(TreeNode* root) {
// map: col -> map<row, multiset<values>>
map<int, map<int, multiset<int>>> nodes;
// perform DFS traversal
dfs(root, 0, 0, nodes);

vector<vector<int>> result;
for (auto& [col, m] : nodes) {
vector<int> colVals;
for (auto& [row, s] : m) {
colVals.insert(colVals.end(), s.begin(), s.end());
}
result.push_back(colVals);
}
return result;
}

private:
void dfs(TreeNode* node, int row, int col, map<int, map<int, multiset<int>>>& nodes) {
if (!node) return;

nodes[col][row].insert(node->val);
dfs(node->left, row + 1, col - 1, nodes);
dfs(node->right, row + 1, col + 1, nodes);
}
};
```



Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 34 / 34 testcases passed
Pranjal Ranjan submitted at Feb 14, 2025 18:44
Editorial
Solution
Runtime
4 ms | Beats 23.45%
Analyze Complexity
Memory
16.82 MB | Beats 7.68%
Code | C++
/**
* Definition for a binary tree node.
* struct TreeNode {
*   int val;
*   TreeNode *left;
*   TreeNode *right;
*   TreeNode() : val(0), left(nullptr), right(nullptr) {}
*   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
View more
Code
C++
Auto
* struct TreeNode {
*   int val;
*   TreeNode *left;
*   TreeNode *right;
*   TreeNode() : val(0), left(nullptr), right(nullptr) {}
*   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:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        // map: col -> map<row, multiset<values>>
        map<int, map<int, multiset<int>>> nodes;
        // perform DFS traversal
Saved
Ln 40, Col 1
Testcase
Test Result
Accepted Runtime: 2 ms
Case 1
Case 2
Case 3
Input
root =
[3,9,20,null,null,15,7]
Output
[[9],[3,15],[20],[7]]
Expected
[[9],[3,15],[20],[7]]