ASSIGNMENT 3

Name- Badal Yadav

UID-22BCS1092

SEC-601 'A'

94.Binary Tree Inorder Traversal

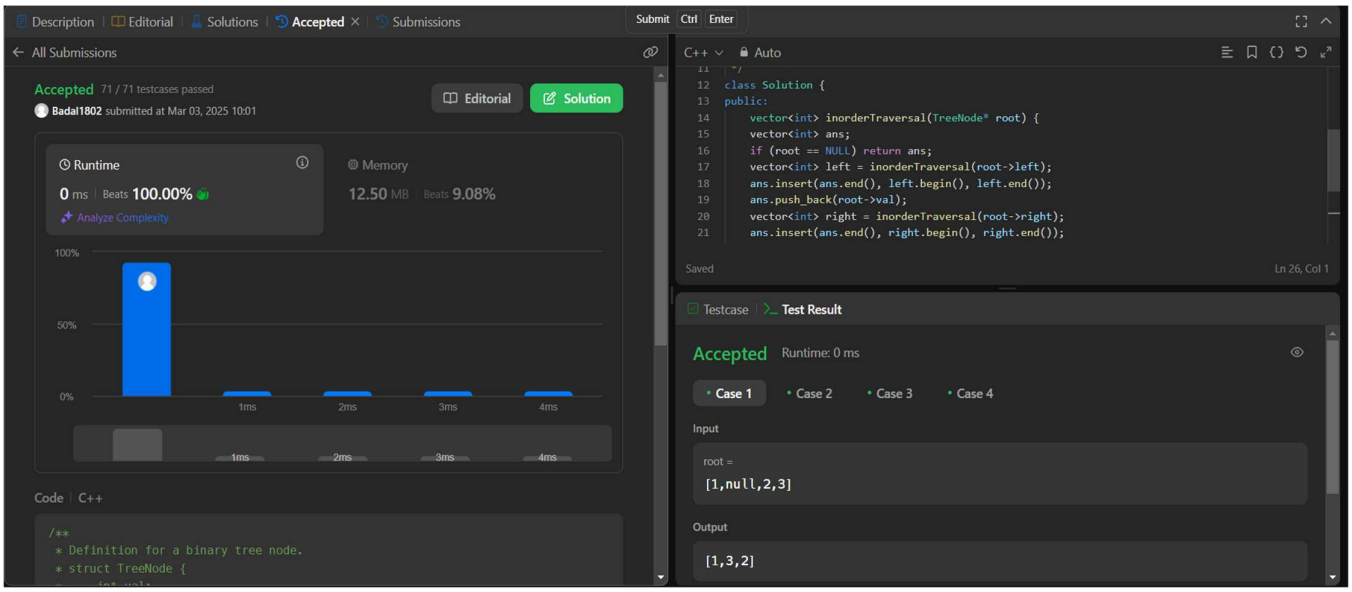
CODE:

```
/**
 * 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> 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;
```

}

};

OUTPUT:



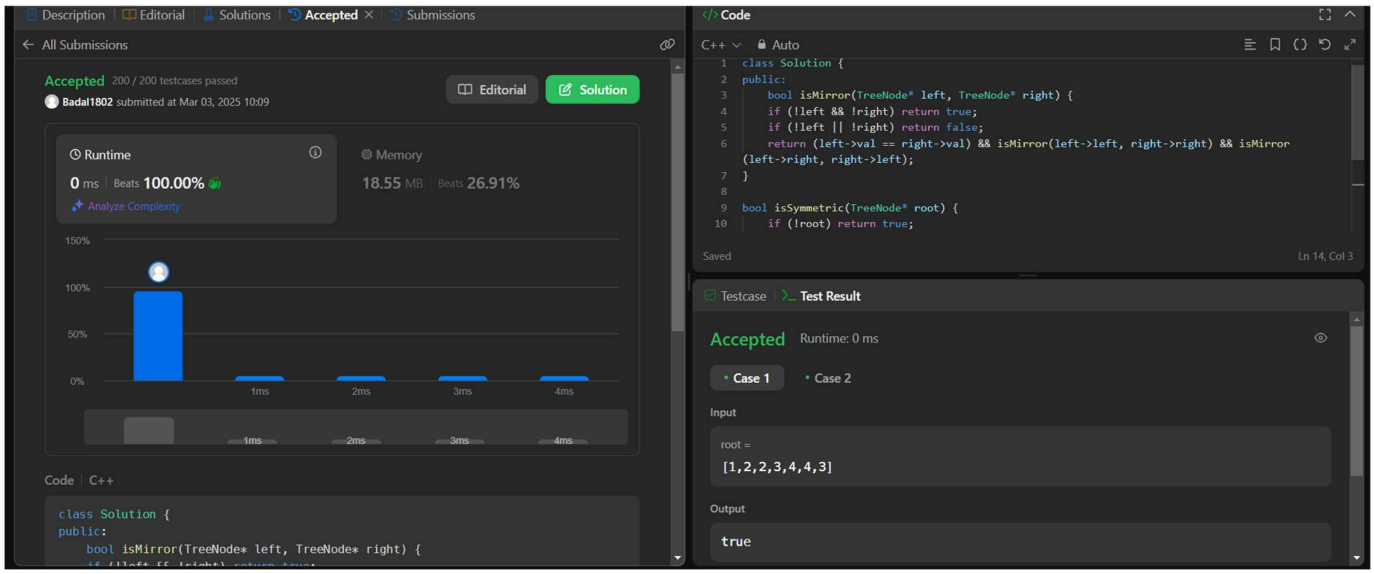
101.Symmetric Tree

CODE:

```
class Solution {

public:

    bool isMirror(TreeNode* left, TreeNode* right) {

    if (!left && !right) return true;

    if (!left || !right) return false;

    return (left->val == right->val) && 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:



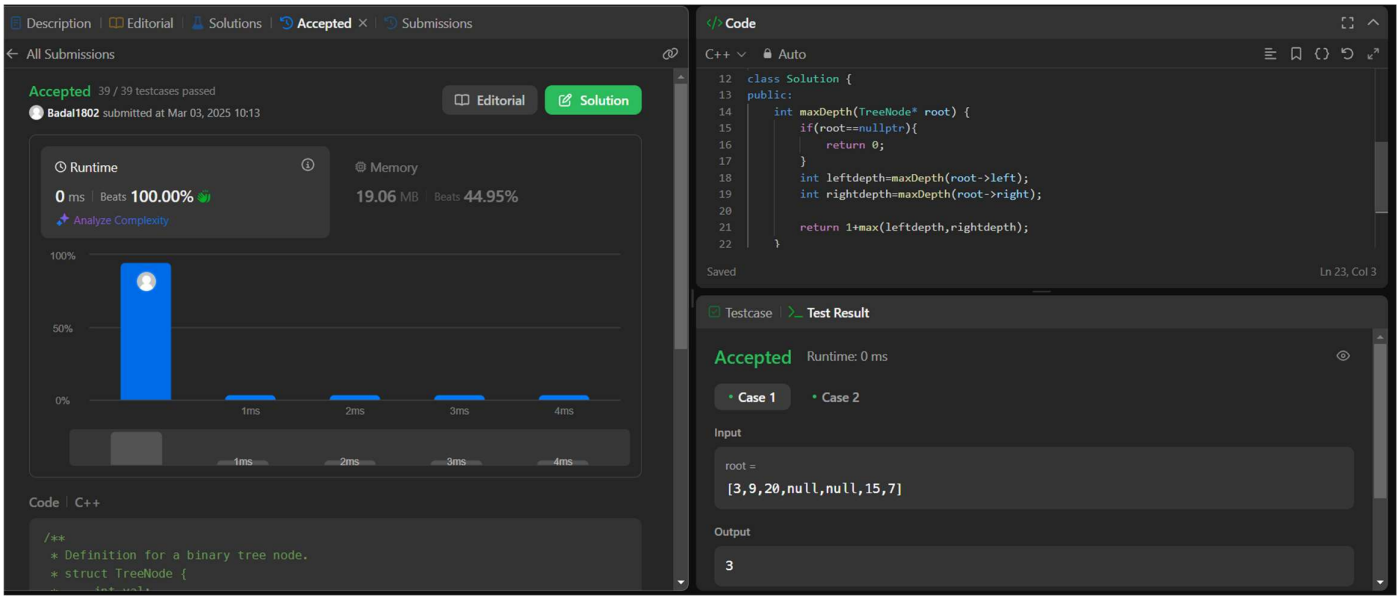
104.Maximum Depth of Binary Tree

CODE:

```
/**
 * 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 1+max(leftdepth,rightdepth);
    }
};
```

OUTPUT:

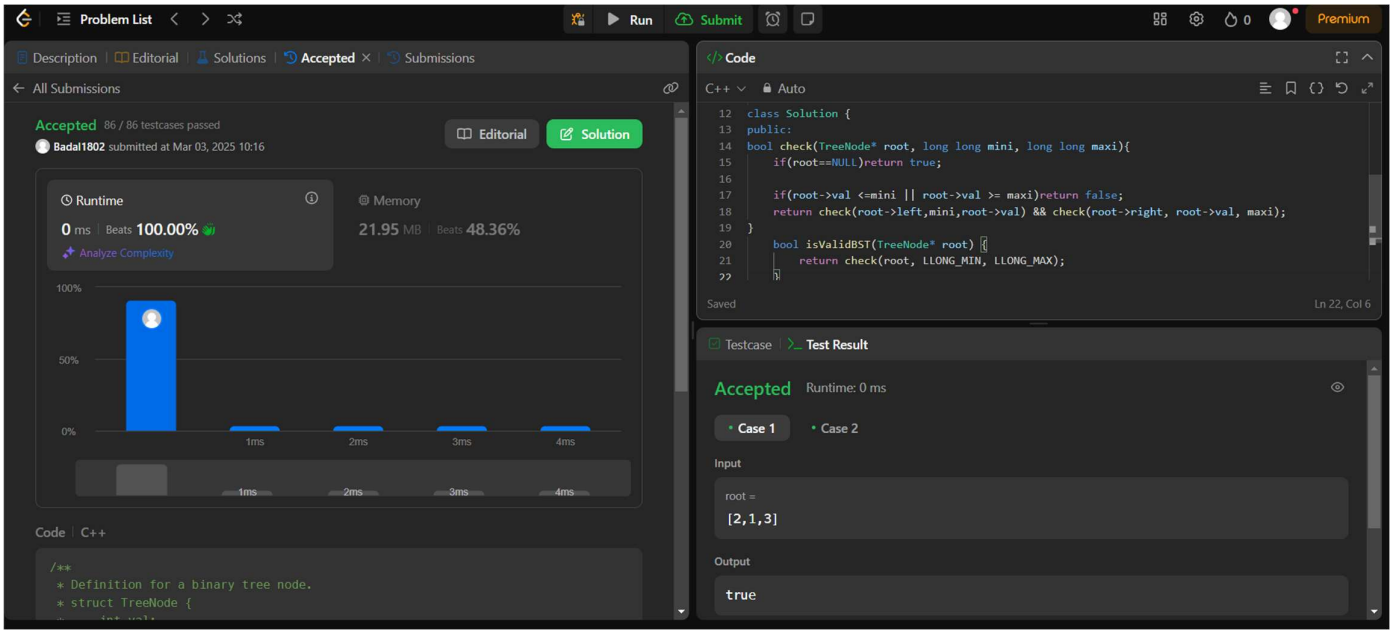


98.Validate Binary Search Tree

CODE:

```
class Solution {
public:
bool check(TreeNode* root, long long mini, long long maxi){
    if(root==NULL)return true;

    if(root->val <=mini || root->val >= maxi)return false;
    return check(root->left,mini,root->val) && check(root->right, root->val, maxi);
}
    bool isValidBST(TreeNode* root) {
        return check(root, LLONG_MIN, LLONG_MAX);
    }
};
```

OUTPUT:

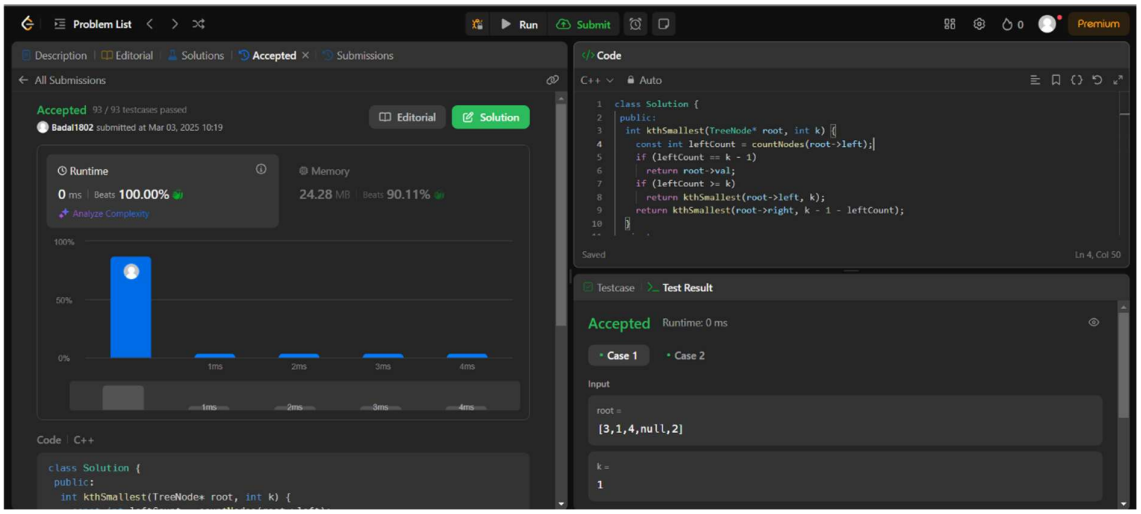


230.Kth Smallest Element in a BST

CODE:

```
class Solution {
 public:
  int kthSmallest(TreeNode* root, int k) {
    const int leftCount = countNodes(root->left);
    if (leftCount == k - 1)
      return root->val;
    if (leftCount >= k)
      return kthSmallest(root->left, k);
    return kthSmallest(root->right, k - 1 - leftCount);
  }
 private:
  int countNodes(TreeNode* root) {
    if (root == nullptr)
      return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
  }
};
```

OUTPUT:

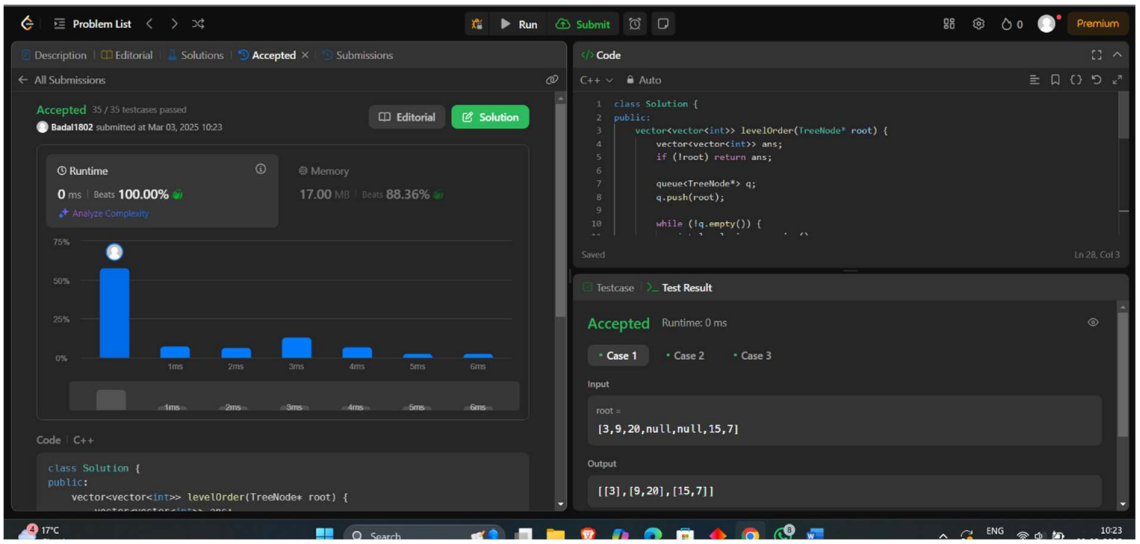


102. Binary Tree Level Order Traversal

CODE:

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

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

        while (!q.empty()) {
            int level_size = q.size();
            vector<int> level;

            for (int i = 0; i < level_size; ++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);
            }

            ans.push_back(level);
        }

        return ans;
    }
};
```

OUTPUT:



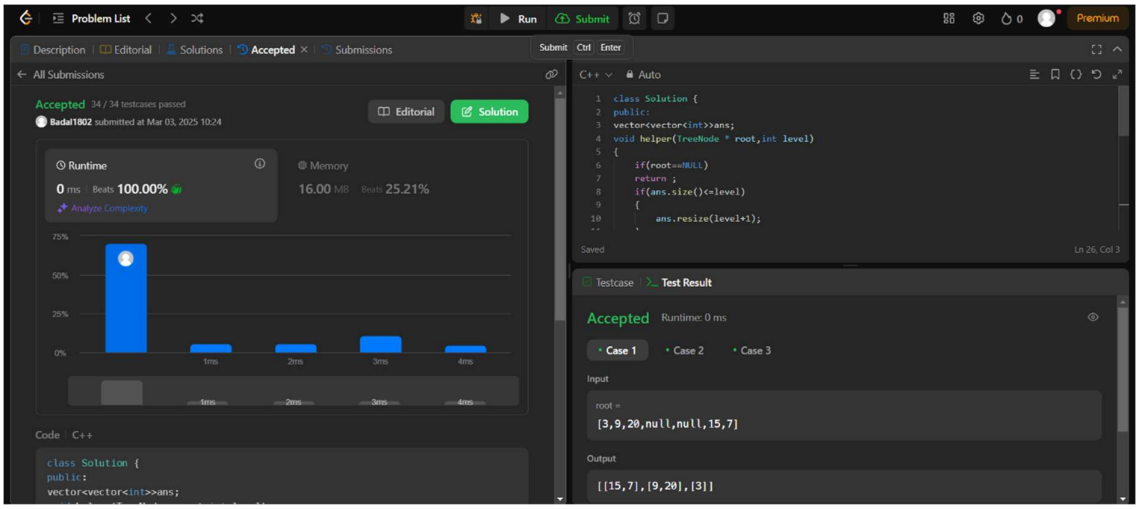
107.Binary Tree Level Order Traversal II

CODE:

```
class Solution {
public:
vector<vector<int>>ans;
void helper(TreeNode * root,int level)
{
   if(root==NULL)
   return ;
   if(ans.size()<=level)
   {
      ans.resize(level+1);
   }
   ans[level].push_back(root->val);
   if(root->left)
   helper(root->left,level+1);
```

```
    if(root->right)
    helper(root->right,level+1);
    return ;

}
    vector<vector<int>> levelOrderBottom(TreeNode* root) {

        helper(root,0);
        reverse(ans.begin(),ans.end());
        return ans;
    }
};
```

OUTPUT:



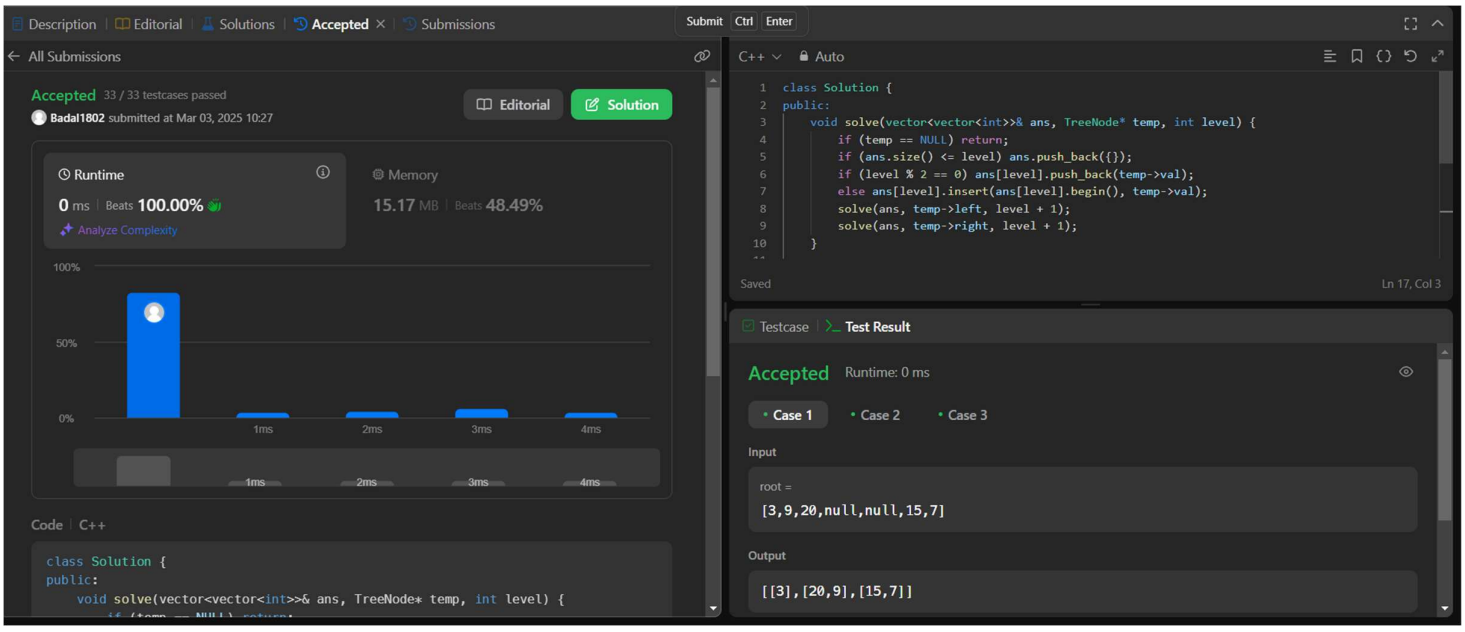
103.Binary Tree Zigzag Level Order Traversal

CODE:

```
class Solution {
```

```
public:
    void solve(vector<vector<int>>& ans, TreeNode* temp, int level) {
        if (temp == NULL) return;
        if (ans.size() <= level) ans.push_back({});
        if (level % 2 == 0) ans[level].push_back(temp->val);
        else ans[level].insert(ans[level].begin(), temp->val);
        solve(ans, temp->left, level + 1);
        solve(ans, temp->right, level + 1);
    }

    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        solve(ans, root, 0);
        return ans;
    }
};
```

OUTPUT:

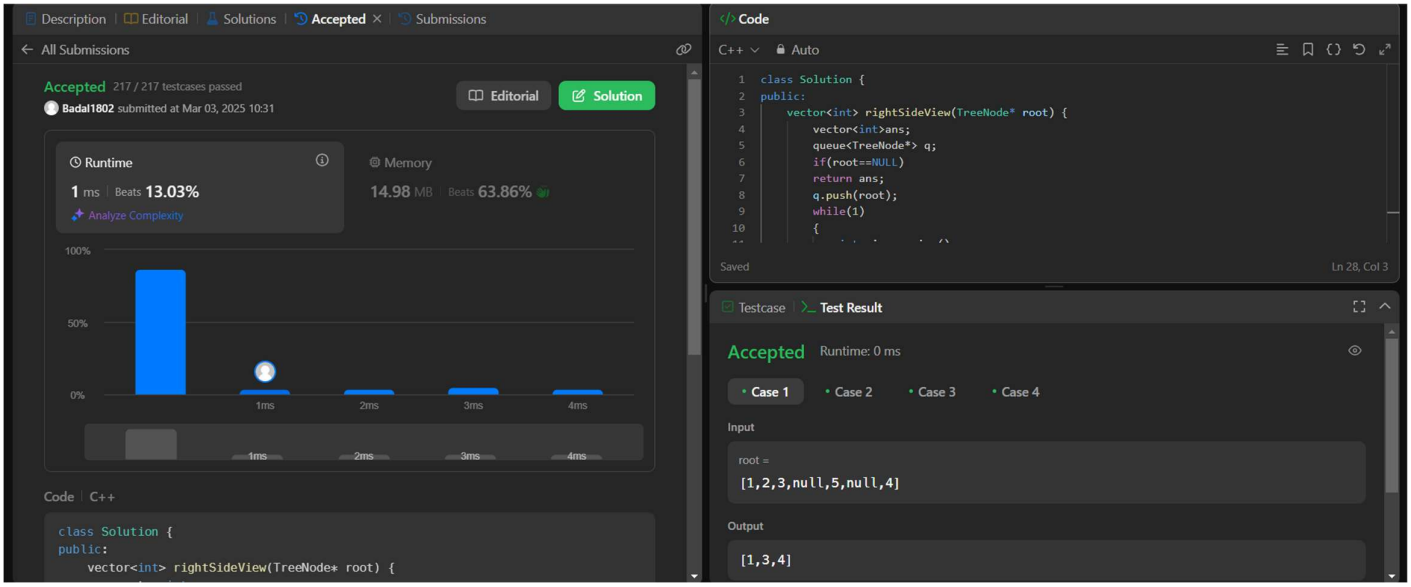


199.Binary Tree Right Side View

CODE:

```
class Solution {
public:
   vector<int> rightSideView(TreeNode* root) {
      vector<int>ans;
      queue<TreeNode*> q;
      if(root==NULL)
      return ans;
      q.push(root);
      while(1)
      {
         int size=q.size();
         if(size==0)
         return ans;
         vector<int> data;
         while(size--)
         {
            TreeNode* temp=q.front();
            q.pop();
            data.push_back(temp->val);
            if(temp->left!=NULL)
            q.push(temp->left);
            if(temp->right!=NULL)
            q.push(temp->right);
         }
         ans.push_back(data.back());
      }
   }
};
```

OUTPUT:



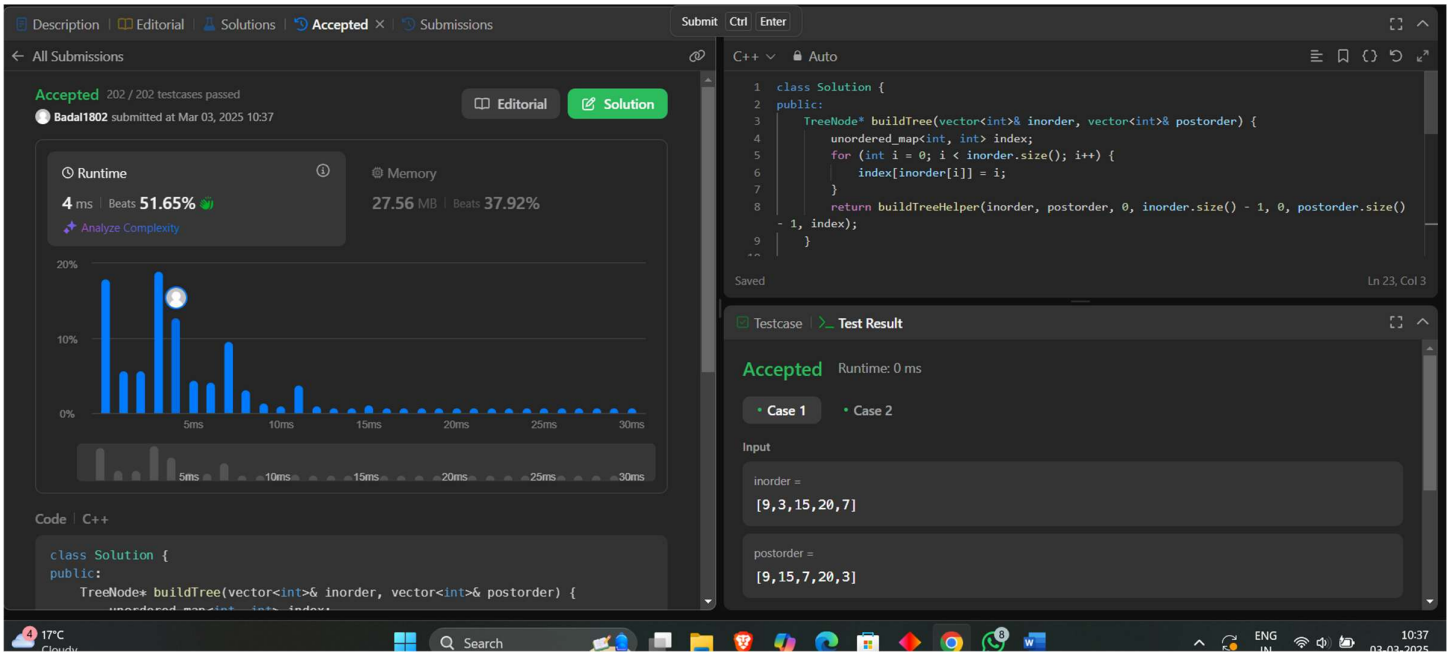
106.Construct Binary Tree from Inorder and Postorder Traversal

CODE:

```
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;
  }
};
```

OUTPUT:

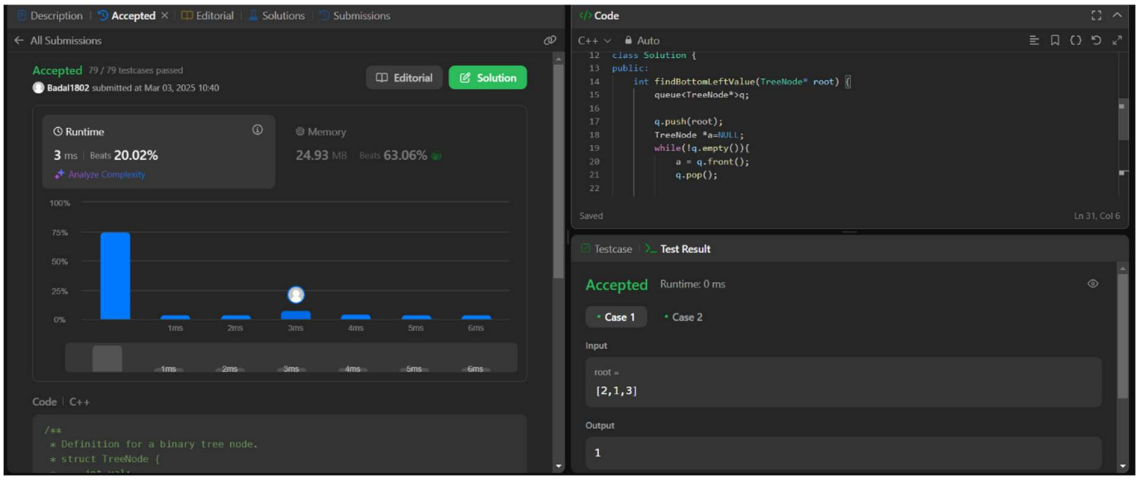


513.Find Bottom Left Tree Value

CODE:

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*>q;
        q.push(root);
        TreeNode *a=NULL;
        while(!q.empty()){
            a = q.front();
            q.pop();
            if(a->right){
                q.push(a->right);
            }
            if(a->left){
                q.push(a->left);
            }
        }
        return a->val;
    }
};
```

OUTPUT:

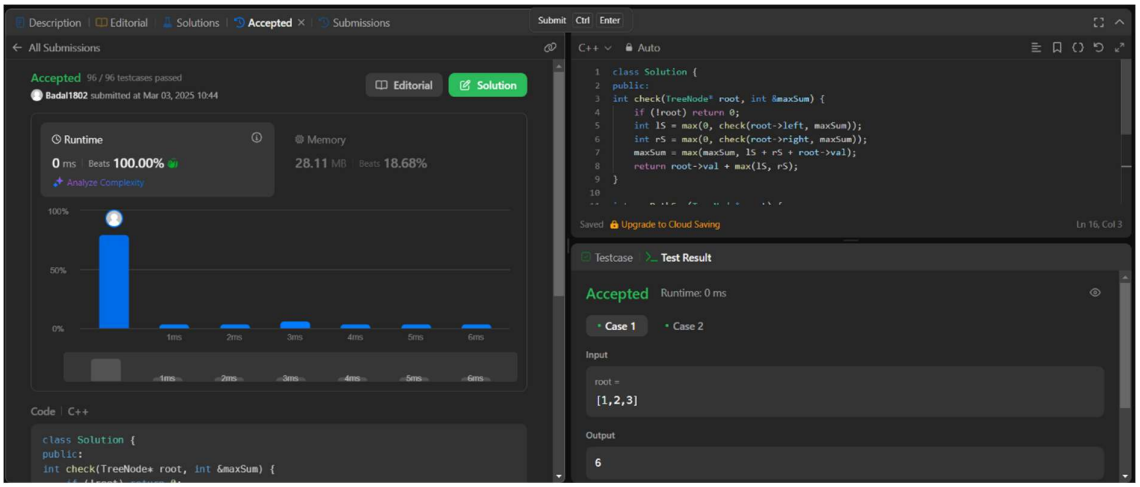


124. Binary Tree Maximum Path Sum

CODE:

```
class Solution {
public:
int check(TreeNode* root, int &maxSum) {
    if (!root) return 0;
    int lS = max(0, check(root->left, maxSum));
    int rS = max(0, check(root->right, maxSum));
    maxSum = max(maxSum, lS + rS + root->val);
    return root->val + max(lS, rS);
}

int maxPathSum(TreeNode* root) {
    int maxSum = INT_MIN;
    check(root, maxSum);
    return maxSum;
}
};
```

OUTPUT:



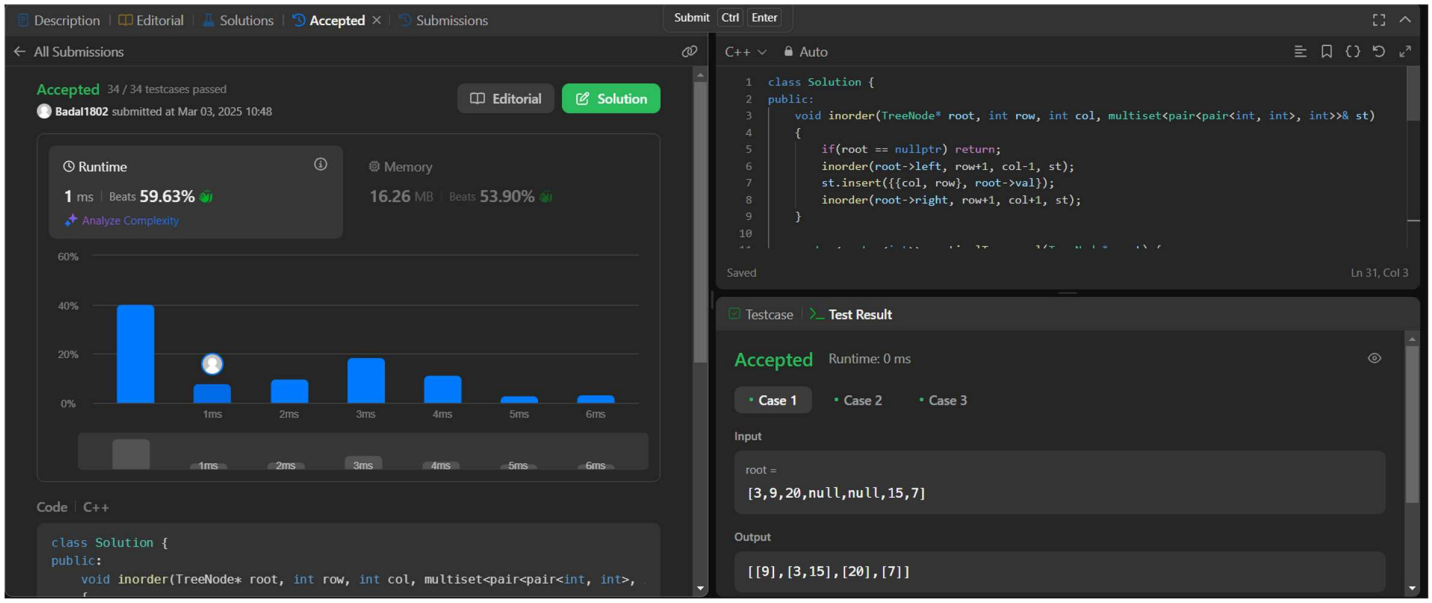
987.Vertical Order Traversal of a Binary Tree

CODE:

```
class Solution {
public:
    void inorder(TreeNode* root, int row, int col, multiset<pair<pair<int, int>, int>>& st)
    {
        if(root == nullptr) return;
        inorder(root->left, row+1, col-1, st);
        st.insert({{col, row}, root->val});
        inorder(root->right, row+1, col+1, st);
    }

    vector<vector<int>> verticalTraversal(TreeNode* root) {
        vector<vector<int>>ans;
        if(root == nullptr) return ans;
        multiset<pair<pair<int, int>, int>>st;
        inorder(root, 0, 0, st);

        auto it = st.begin();
        while(it != st.end())
        {
            vector<int>vec;
            int col = it->first.first;
            while(it != st.end() && it->first.first == col)
            {
                vec.push_back(it->second);
                it++;
            }
            ans.push_back(vec);
        }
        return ans;
```

```
    }
};
```

OUTPUT:


Description | Editorial | Solutions | Accepted | Submissions
Submit Ctrl Enter
All Submissions
Accepted 34 / 34 testcases passed
Badal1802 submitted at Mar 03, 2025 10:48
Editorial
Solution
Runtime
1 ms | Beats 59.63%
Analyze Complexity
Memory
16.26 MB | Beats 53.90%
60%
40%
20%
0%
1ms 2ms 3ms 4ms 5ms 6ms
Code | C++
class Solution {
public:
void inorder(TreeNode* root, int row, int col, multiset<pair<pair<int, int>,
C++ Auto
class Solution {
public:
void inorder(TreeNode* root, int row, int col, multiset<pair<pair<int, int>, int>>& st)
{
if(root == nullptr) return;
inorder(root->left, row+1, col-1, st);
st.insert({{col, row}, root->val});
inorder(root->right, row+1, col+1, st);
}
Saved
Ln 31, Col 3
Testcase | Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[3,9,20,null,null,15,7]
Output
[[9],[3,15],[20],[7]]