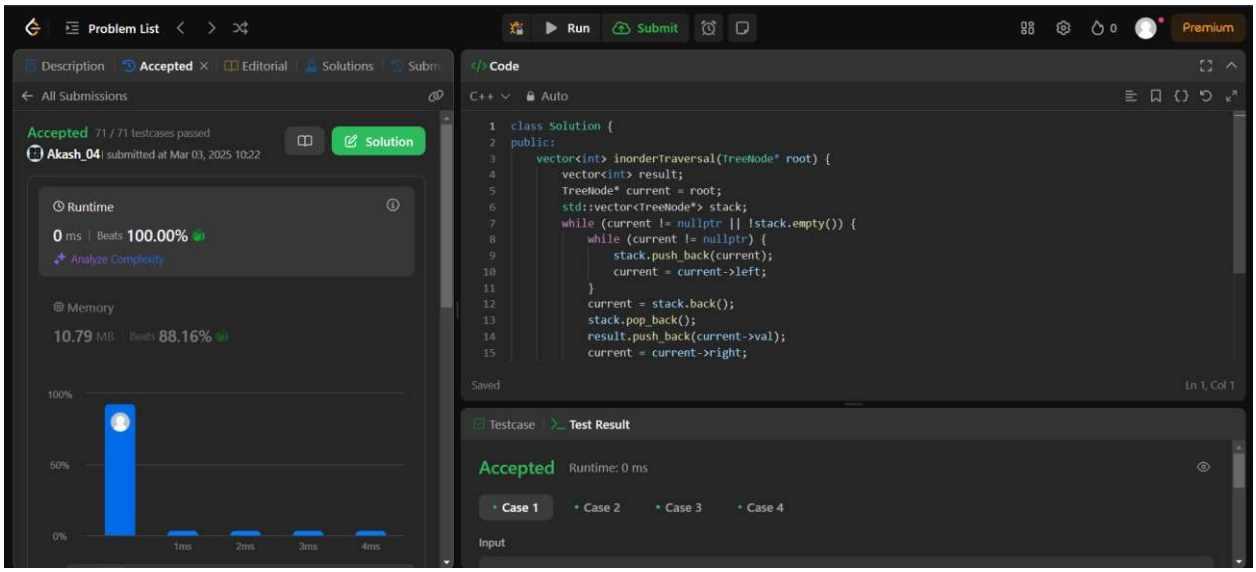
Name: Akash Pandey

UID: 22BCS11135

Batch: FL_IoT 601 /'A'

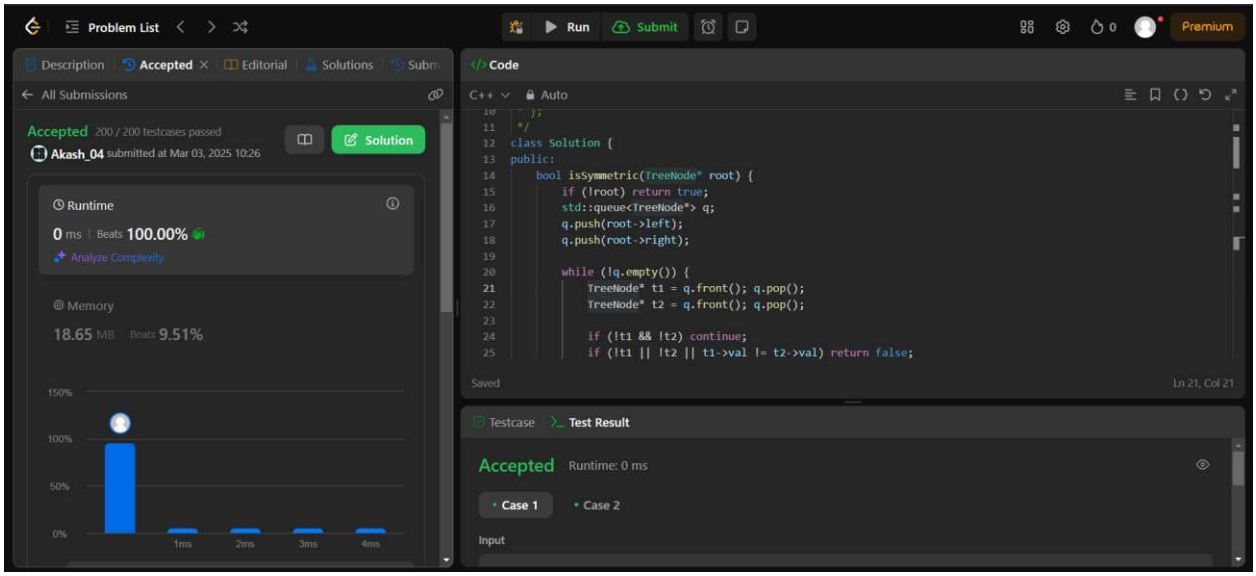
1. Binary Tree Inorder Traversal

```
class Solution {
public:
 vector<int> inorderTraversal(TreeNode* root) {
vector<int> ans;    stack<TreeNode*> stack;

   while (root != nullptr || !stack.empty()) {
    while (root != nullptr) {
stack.push(root);
      root = root->left;
    }
    root = stack.top(), stack.pop();
ans.push_back(root->val);      root
= root->right;
   }

   return ans;
 }
};
```



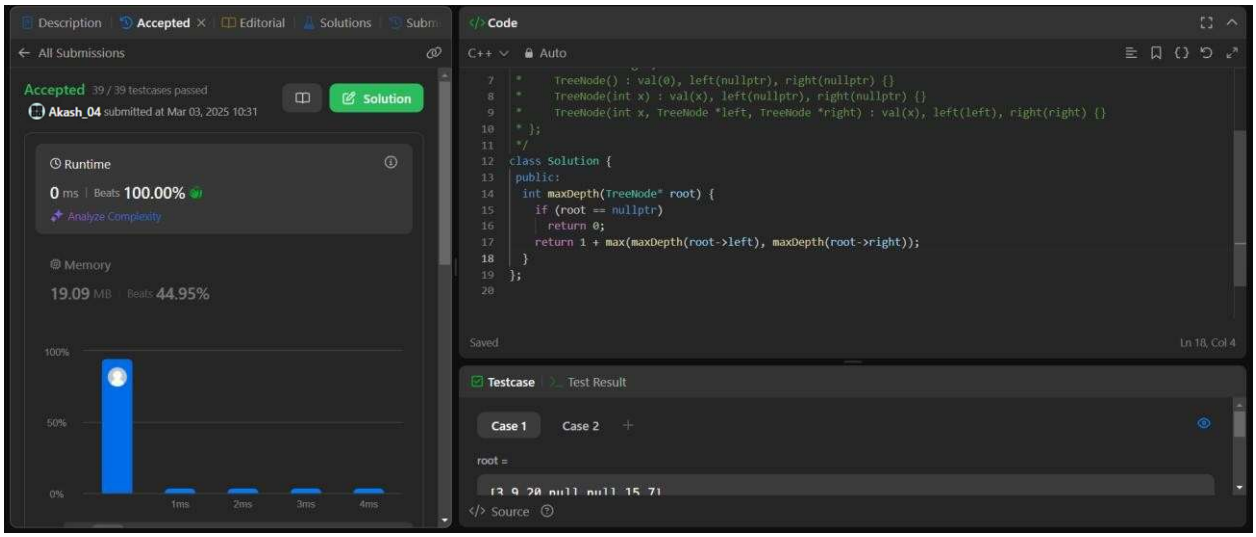
2. Symmetric Tree

```
class Solution {  public:   bool
isSymmetric(TreeNode* root) {
return isSymmetric(root, root);
  }   private:   bool isSymmetric(TreeNode* p,
TreeNode* q) {     if (!p || !q)       return
p == q;
     return p->val == q->val &&
//            isSymmetric(p->left, q->right) &&
//            isSymmetric(p->right, q->left);
  }
};
```



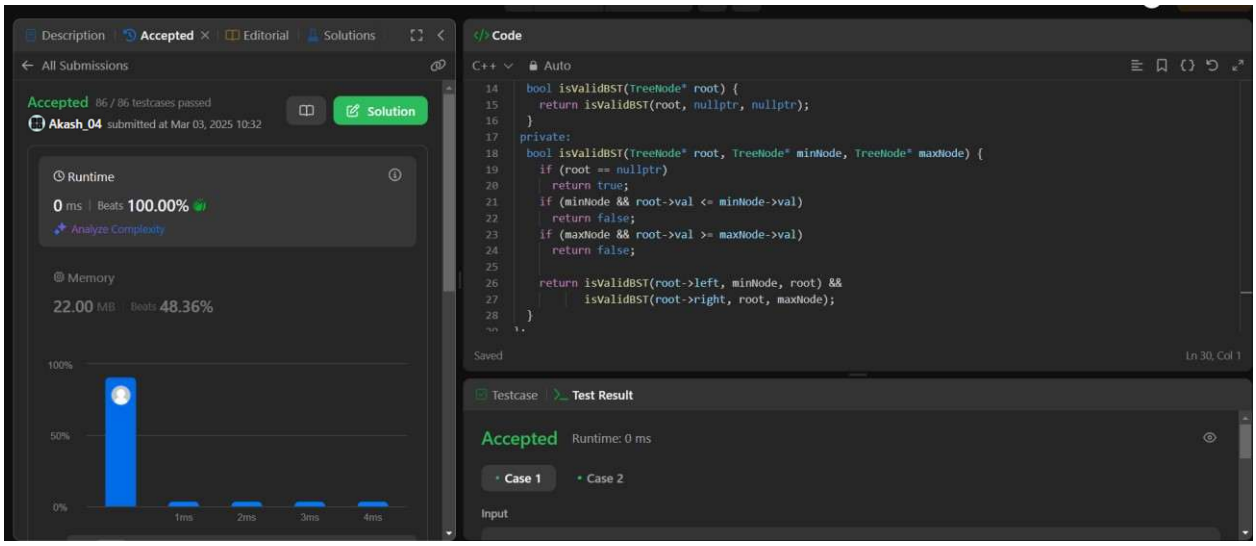
3 .Maximum Depth of Binary Tree

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



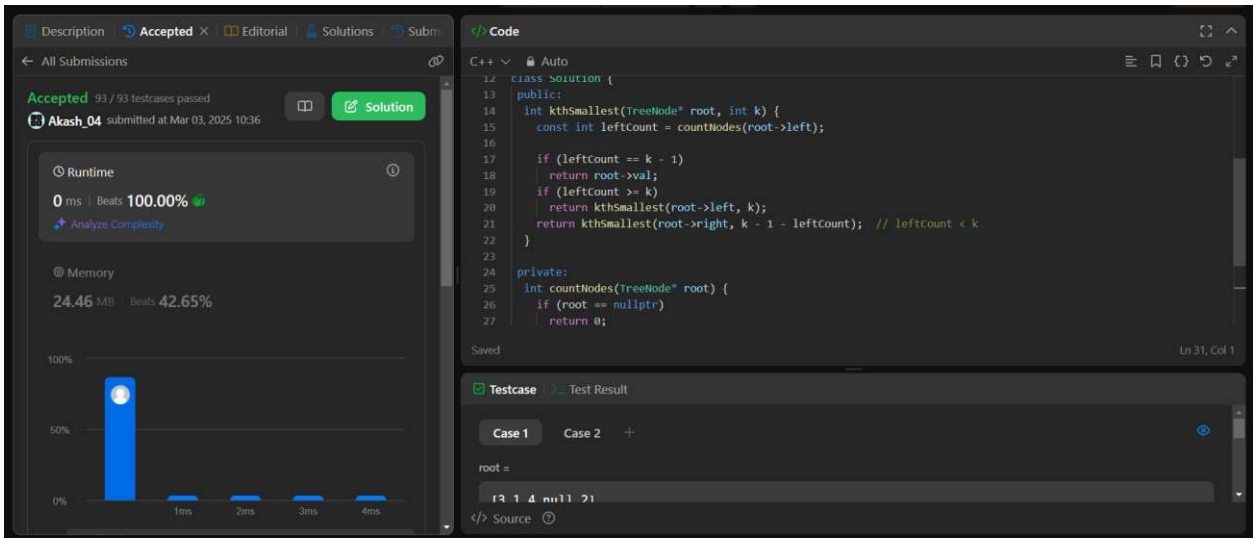
4. Validate Binary Search Tree

```
class Solution {
public:
  bool isValidBST(TreeNode* root) {
    return isValidBST(root, nullptr, nullptr);
  }
 private:
  bool isValidBST(TreeNode* root, TreeNode* minNode, TreeNode* maxNode) {
if (root == nullptr)
      return true;
    if (minNode && root->val <= minNode->val)
return false;
    if (maxNode && root->val >= maxNode->val)
return false;

    return isValidBST(root->left, minNode, root) &&          isValidBST(root-
>right, root, maxNode);
  }
};
```



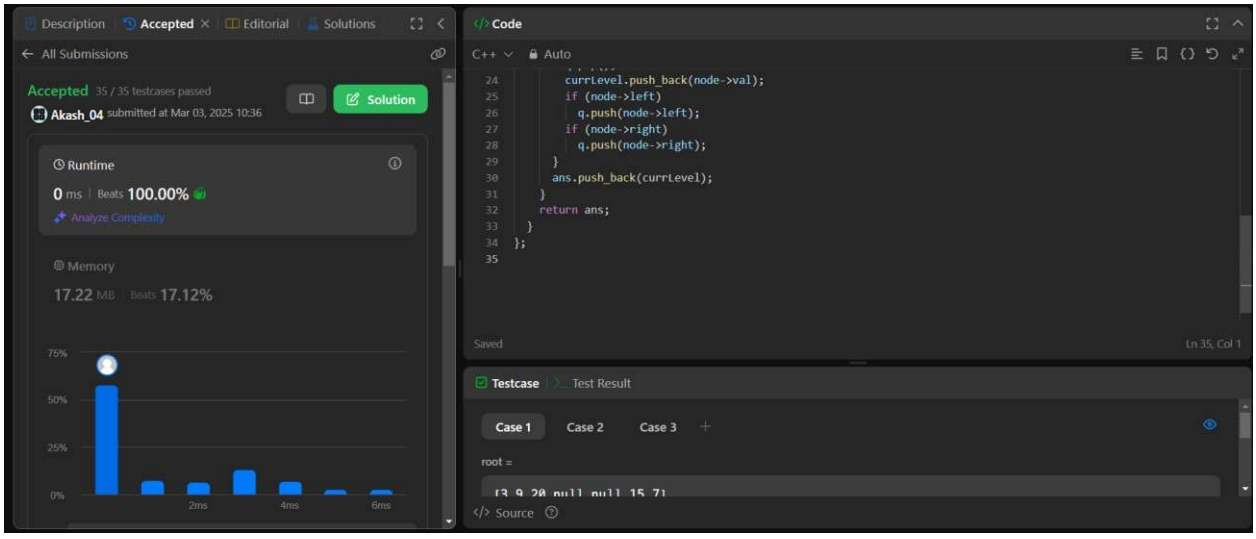
5.Kth Smallest Element in a BST

```
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);  // leftCount < k    }   private:
  int countNodes(TreeNode* root) {     if (root == nullptr)
return 0;     return 1 + countNodes(root->left) +
countNodes(root->right);
  }
};
```



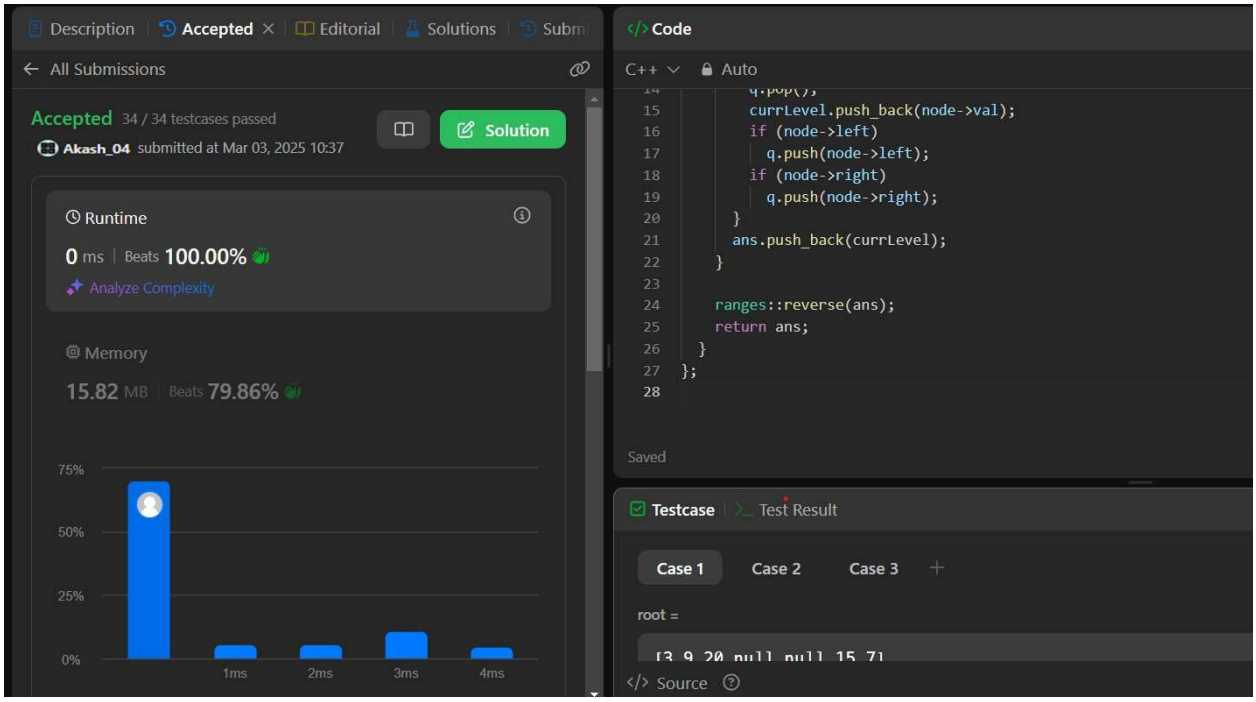
6. Binary Tree Level Order Traversal

```
class Solution {
public:
 vector<vector<int>> levelOrder(TreeNode* root) {
if (root == nullptr)
    return {};
   vector<vector<int>> ans;
queue<TreeNode*> q{{root}};
   while (!q.empty()) {
vector<int> currLevel;
    for (int sz = q.size(); sz > 0; --sz) {
TreeNode* node = q.front();       q.pop();
      currLevel.push_back(node->val);
if (node->left)
       q.push(node->left);
if (node->right)
       q.push(node->right);
     }
    ans.push_back(currLevel);
   }
   return ans;
 }
};
```



7. Binary Tree Level Order Traversal II

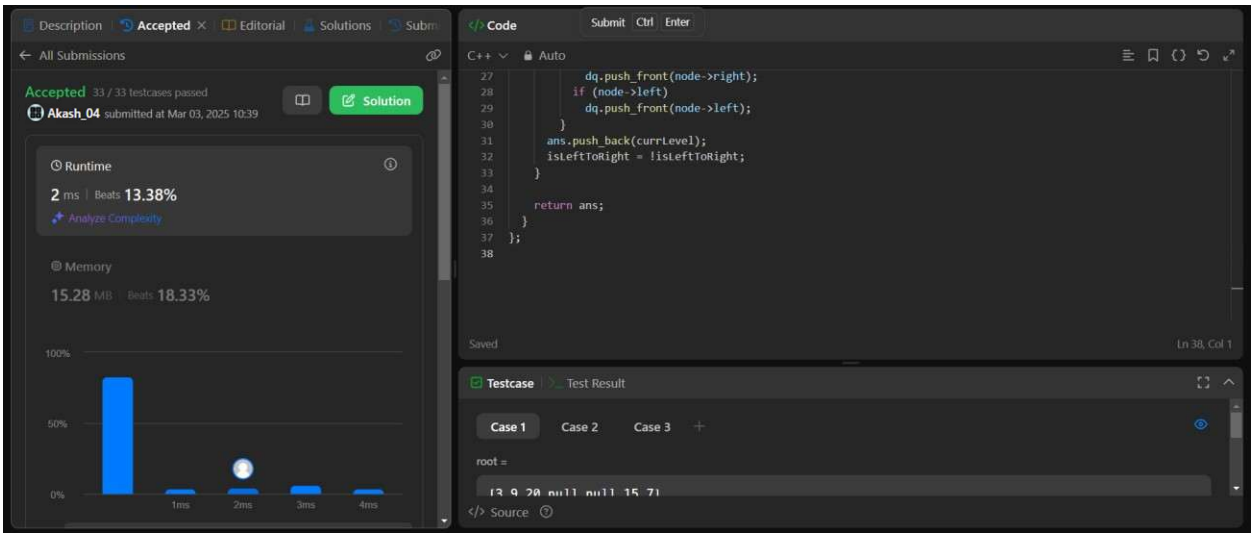
```
class Solution {  public:
  vector<vector<int>> levelOrderBottom(TreeNode* root) {
if (root == nullptr)
     return {};

   vector<vector<int>> ans;
   queue<TreeNode*> q{{root}};

   while (!q.empty()) {
vector<int> currLevel;
     for (int sz = q.size(); sz > 0; --sz) {
TreeNode* node = q.front();        q.pop();
       currLevel.push_back(node->val);
if (node->left)
         q.push(node->left);
if (node->right)
         q.push(node->right);
     }
     ans.push_back(currLevel);
   }

   ranges::reverse(ans);
return ans;
  }
};
```



8. .[Binary Tree Zigzag Level Order Traversal]

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

  vector<vector<int>> ans;
deque<TreeNode*> dq{{root}};
  bool isLeftToRight = true;

  while (!dq.empty()) {
vector<int> currLevel;      for (int sz
= dq.size(); sz > 0; --sz)       if
(isLeftToRight) {
        TreeNode* node = dq.front();
        dq.pop_front();
        currLevel.push_back(node->val);
        if (node->left)
          dq.push_back(node->left);
        if (node->right)
          dq.push_back(node->right);
       } else {
```

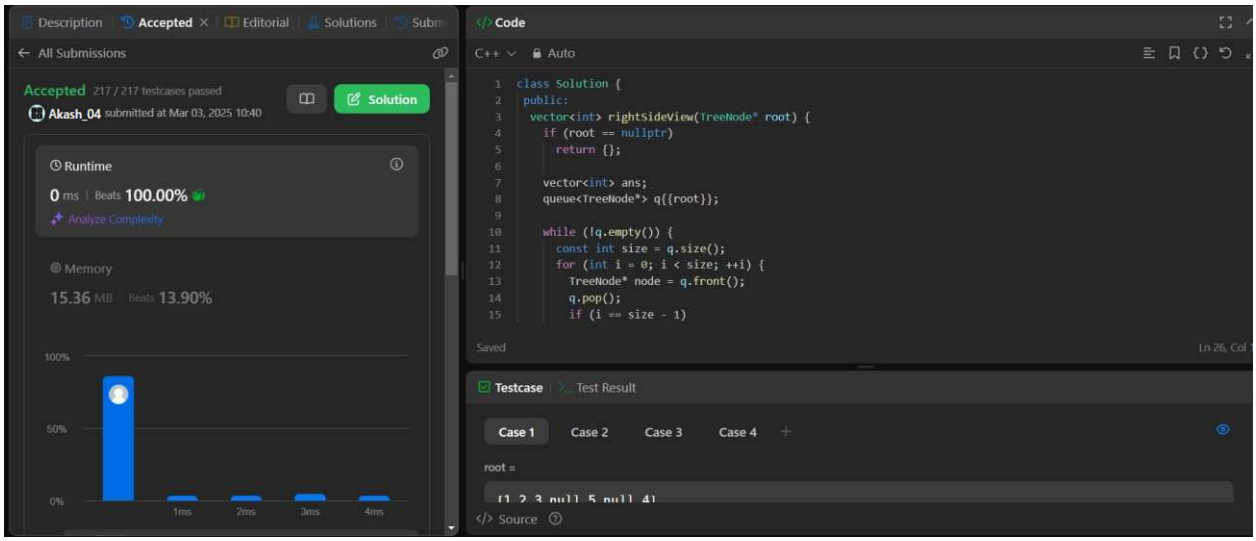
```
        TreeNode* node = dq.back();
        dq.pop_back();
        currLevel.push_back(node->val);
        if (node->right)
          dq.push_front(node->right);
        if (node->left)
          dq.push_front(node->left);
       }
     ans.push_back(currLevel);
     isLeftToRight = !isLeftToRight;
    }

    return ans;
  }
};
```



9. [Binary Tree Right Side View]

```
class Solution {
public:
  vector<int> rightSideView(TreeNode* root) {
if (root == nullptr)
     return {};

    vector<int> ans;
    queue<TreeNode*> q{{root}};

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

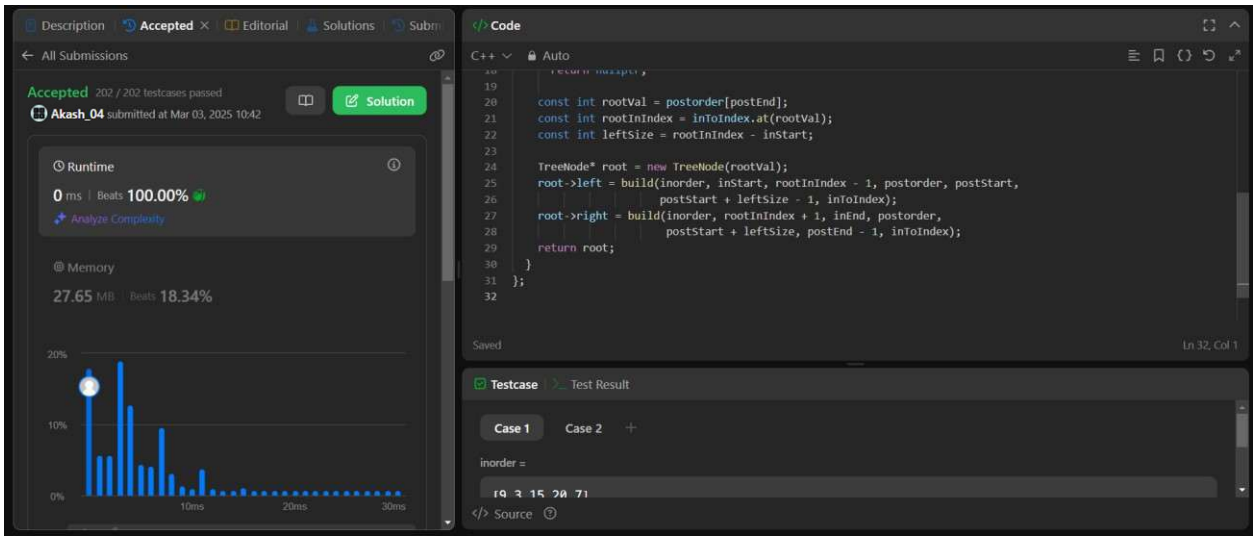
```
    for (int i = 0; i < size; ++i) {
      TreeNode* node = q.front();
q.pop();
      if (i == size - 1)
        ans.push_back(node->val);
if (node->left)
        q.push(node->left);
if (node->right)
        q.push(node->right);
    }
  }
  return ans;
 }
};
```



10. Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {  public:
  TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
unordered_map<int, int> inToIndex;

   for (int i = 0; i < inorder.size(); ++i)
     inToIndex[inorder[i]] = i;
```

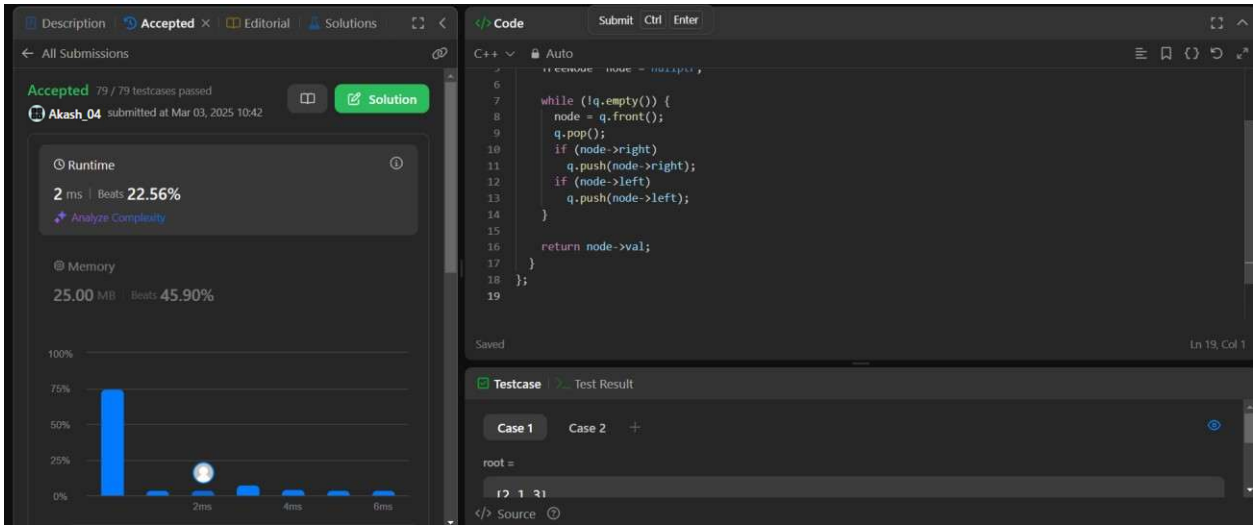
```
    return build(inorder, 0, inorder.size() - 1, postorder, 0,
                 postorder.size() - 1, inToIndex);
  }

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

    const int rootVal = postorder[postEnd];
const int rootInIndex = inToIndex.at(rootVal);
    const int leftSize = rootInIndex - inStart;

    TreeNode* root = new TreeNode(rootVal);
    root->left = build(inorder, inStart, rootInIndex - 1, postorder, postStart,
                       postStart + leftSize - 1, inToIndex);     root->right =
build(inorder, rootInIndex + 1, inEnd, postorder,
postStart + leftSize, postEnd - 1, inToIndex);     return root;
  }
};
```



11. Find Bottom Left Tree Value

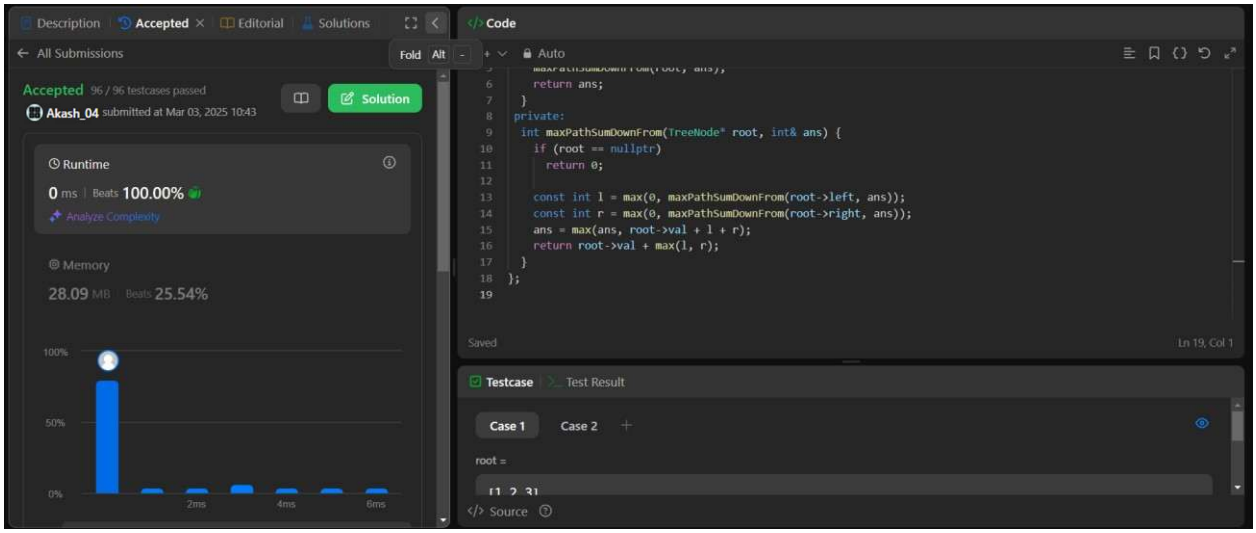
```
class Solution {
public:
  int findBottomLeftValue(TreeNode* root) {
    queue<TreeNode*> q{{root}};
    TreeNode* node = nullptr;
```

```
   while (!q.empty()) {
node = q.front();
q.pop();      if (node-
>right)
      q.push(node->right);
if (node->left)
      q.push(node->left);
   }

   return node->val;
 }
};
```



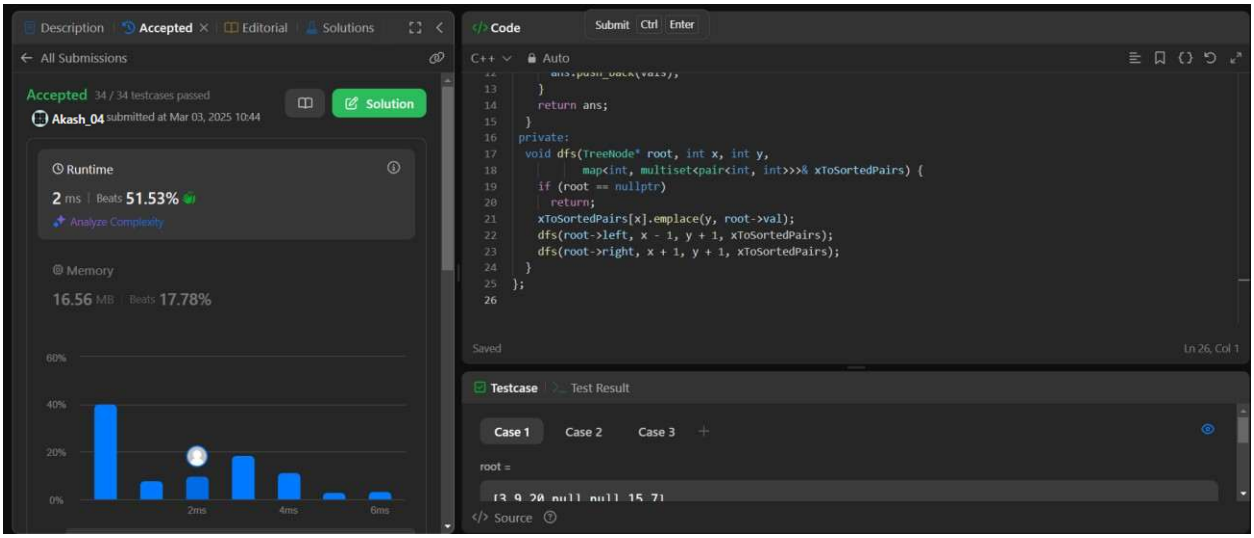
12. [Binary Tree Maximum Path Sum]

```
class Solution {
public:
  int maxPathSum(TreeNode* root) {
int ans = INT_MIN;
maxPathSumDownFrom(root, ans);
return ans;
  }
private:
  int maxPathSumDownFrom(TreeNode* root, int& ans) {
if (root == nullptr)       return 0;
     const int l = max(0, maxPathSumDownFrom(root->left, ans));
const int r = max(0, maxPathSumDownFrom(root->right, ans));
ans = max(ans, root->val + l + r);     return root->val +
max(l, r);
  } };
```



13. .Vertical Order Traversal of a Binary Tree

```
class Solution {
public:
  vector<vector<int>> verticalTraversal(TreeNode* root) {
vector<vector<int>> ans;     map<int, multiset<pair<int,
int>>> xToSortedPairs;
     dfs(root, 0, 0, xToSortedPairs);     for
(const auto& [_, pairs] : xToSortedPairs) {
vector<int> vals;       for (const pair<int, int>&
pair : pairs)         vals.push_back(pair.second);
ans.push_back(vals);
    }
return ans;
  }
private:
  void dfs(TreeNode* root, int x, int y,           map<int,
multiset<pair<int, int>>>& xToSortedPairs) {     if (root ==
nullptr)       return;
    xToSortedPairs[x].emplace(y, root->val);
dfs(root->left, x - 1, y + 1, xToSortedPairs);
dfs(root->right, x + 1, y + 1, xToSortedPairs);
  } };
```


Description
Accepted
Editorial
Solutions
All Submissions
Accepted 34 / 34 testcases passed
Akash_04 submitted at Mar 03, 2025 10:44
Solution
Runtime
2 ms | Beats 51.53%
Analyze Complexity
Memory
16.56 MB | Beats 17.78%
60%
40%
20%
0%
2ms
4ms
6ms
Code
Submit Ctrl Enter
C++
Auto
}
return ans;
}
private:
void dfs(TreeNode* root, int x, int y,
map<int, multiset<pair<int, int>>>& xToSortedPairs) {
if (root == nullptr)
return;
xToSortedPairs[x].emplace(y, root->val);
dfs(root->left, x - 1, y + 1, xToSortedPairs);
dfs(root->right, x + 1, y + 1, xToSortedPairs);
}
};
Saved
Ln 26, Col 1
Testcase
Test Result
Case 1
Case 2
Case 3
root =
[3,9,20,null,null,15,7]
Source