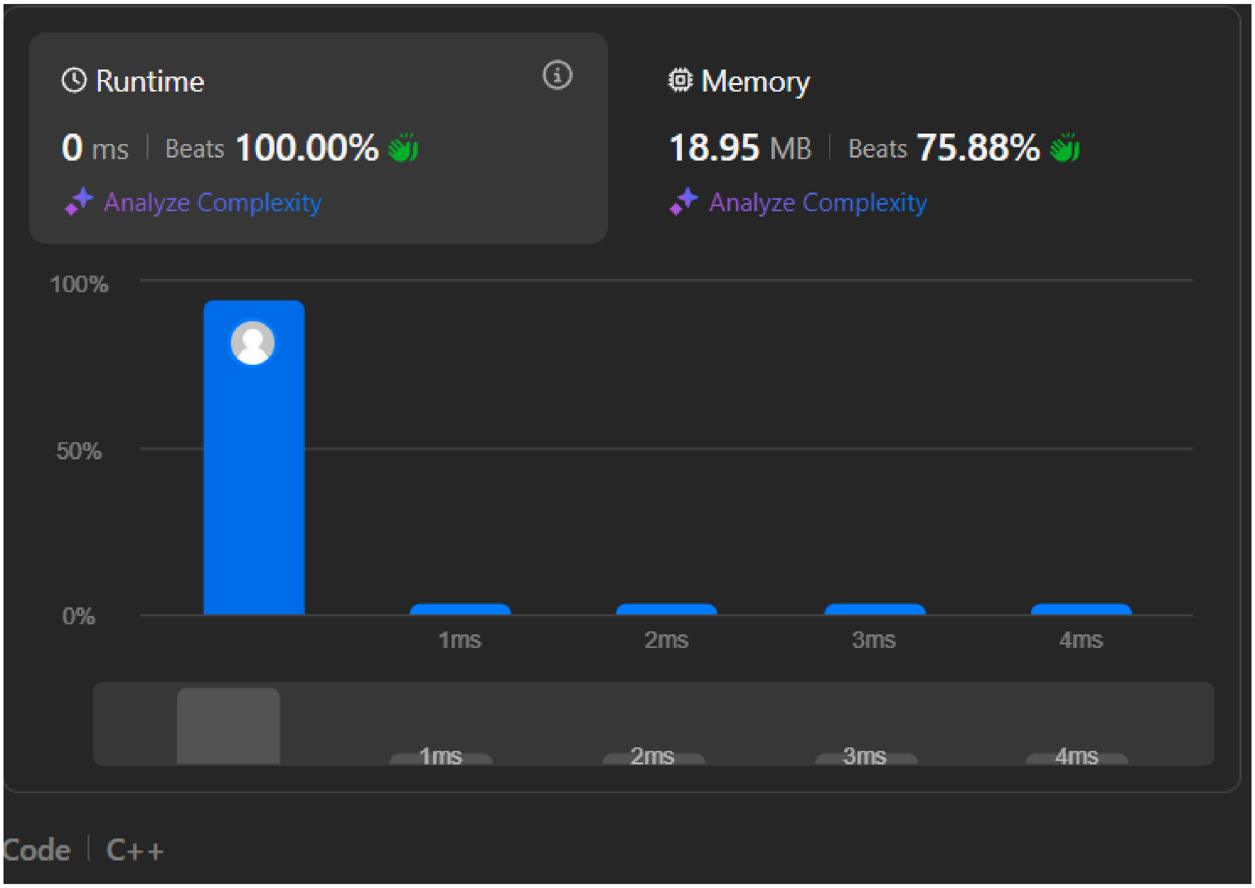
AP ASSIGNMENT - 5

Name: Vipul Vidya | UID: 22bcs16942 | Section: 612-“B”

maximum-depth-of-binary-tree

```
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;}}
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

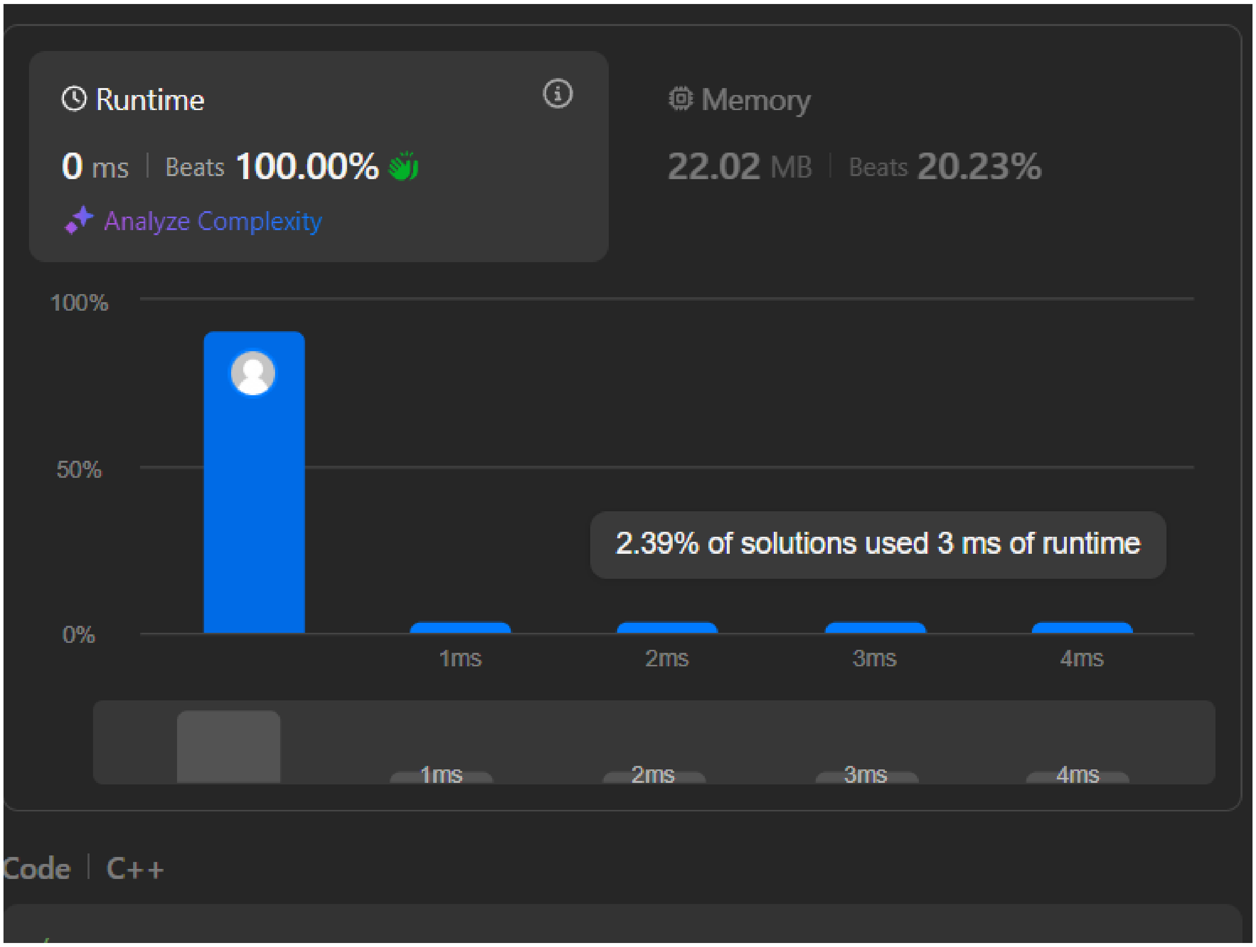
18.95 MB | Beats 75.88%

Analyze Complexity

Code | C++

Validate Binary Search Tree

```
class Solution {
public:
   bool isValidBST(TreeNode* root) {
        return isValidBSTHelper(root, LONG_MIN, LONG_MAX); }
private:
   bool isValidBSTHelper(TreeNode* root, long minVal, long maxVal) {
      if (!root) return true;
      if (root->val <= minVal || root->val >= maxVal) return false;
      return isValidBSTHelper(root->left, minVal, root->val) &&
           isValidBSTHelper(root->right, root->val, maxVal);}};
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

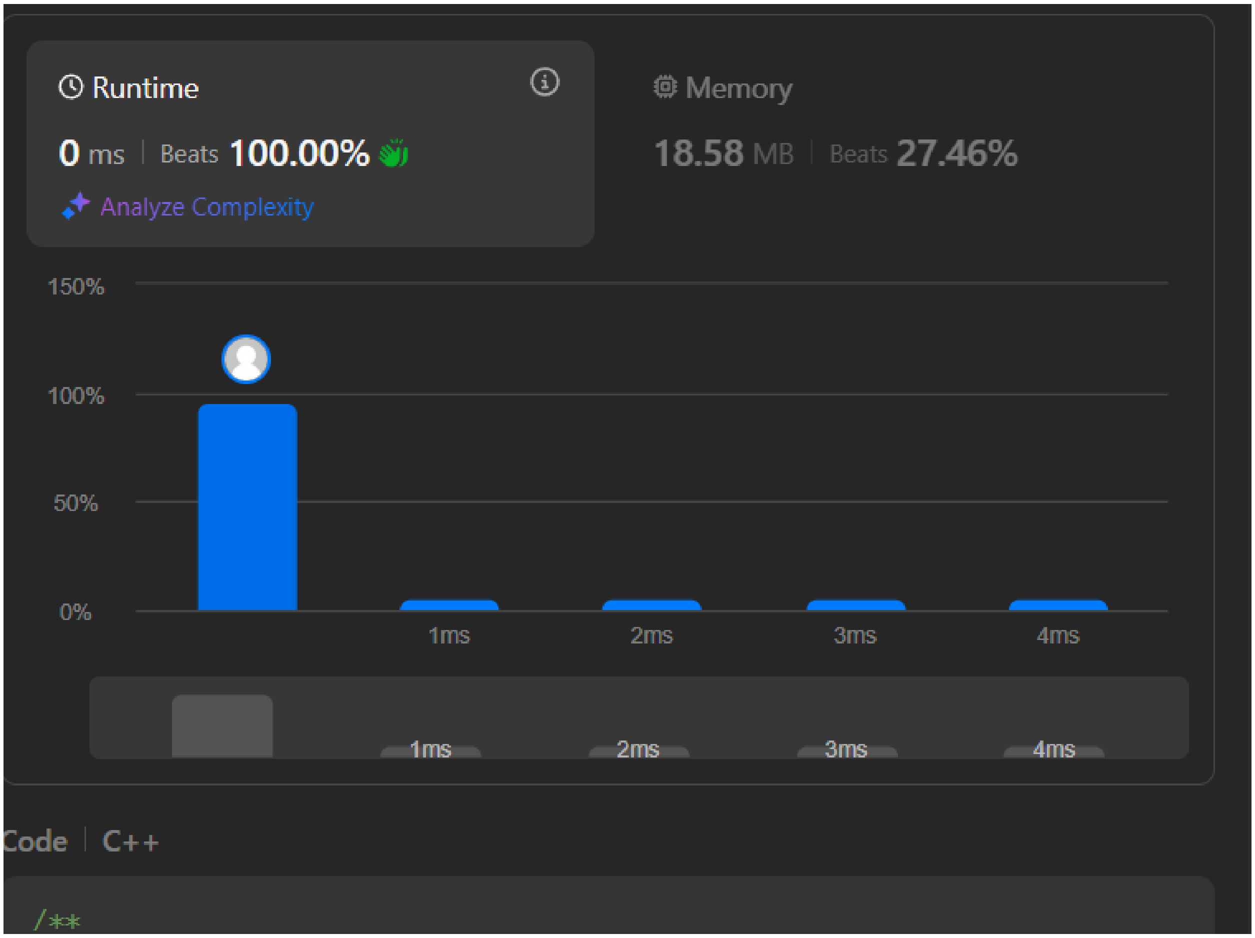
22.02 MB | Beats 20.23%

2.39% of solutions used 3 ms of runtime

Code | C++

Symmetric Tree

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return root == nullptr || isMirror(root->left, root->right);
    }

private:
    bool isMirror(TreeNode* a, TreeNode* b) {
        if (!a || !b) return a == b;
        return (a->val == b->val) && isMirror(a->left, b->right) && isMirror(a->right, b->left);
    }
};
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

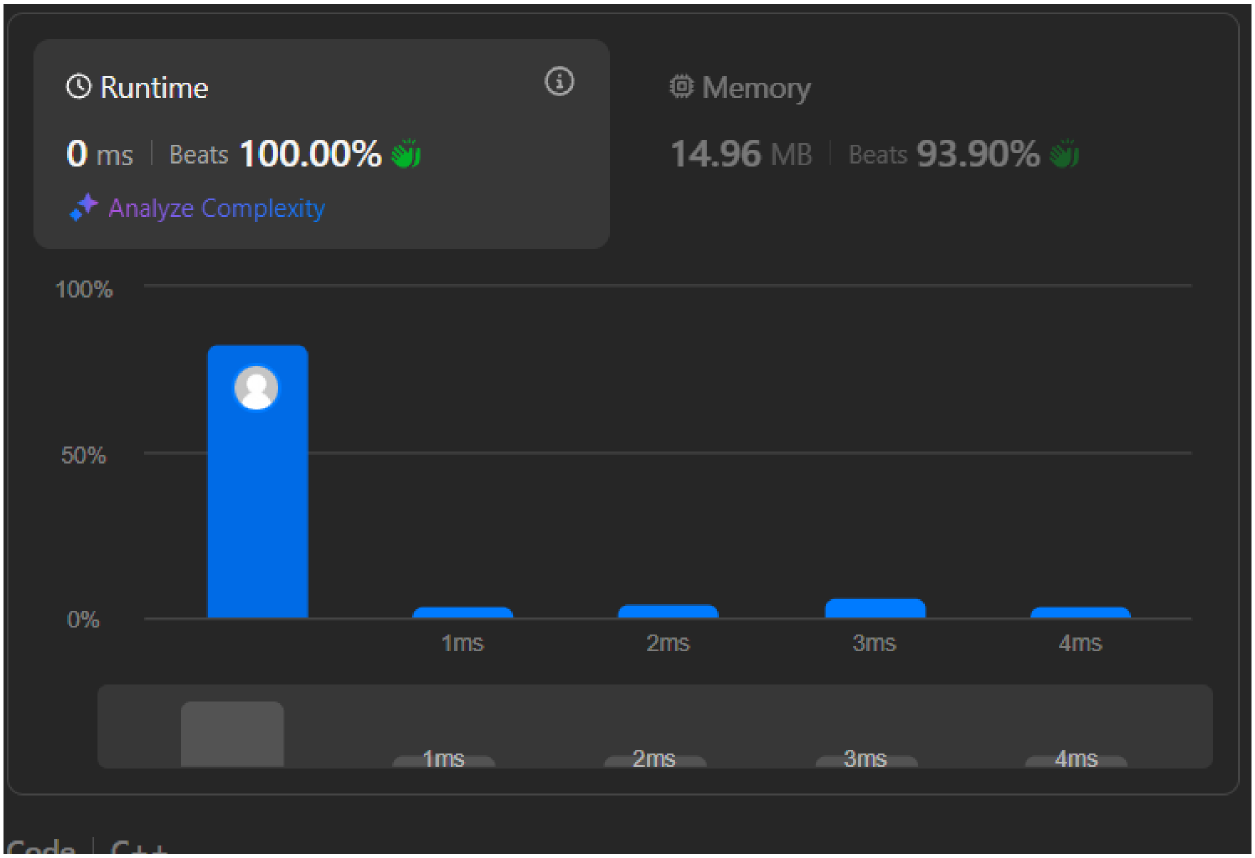
18.58 MB | Beats 27.46%

Code | C++

```
/**
```

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;
    }
};
```



Runtime

0 ms | Beats 100.00%

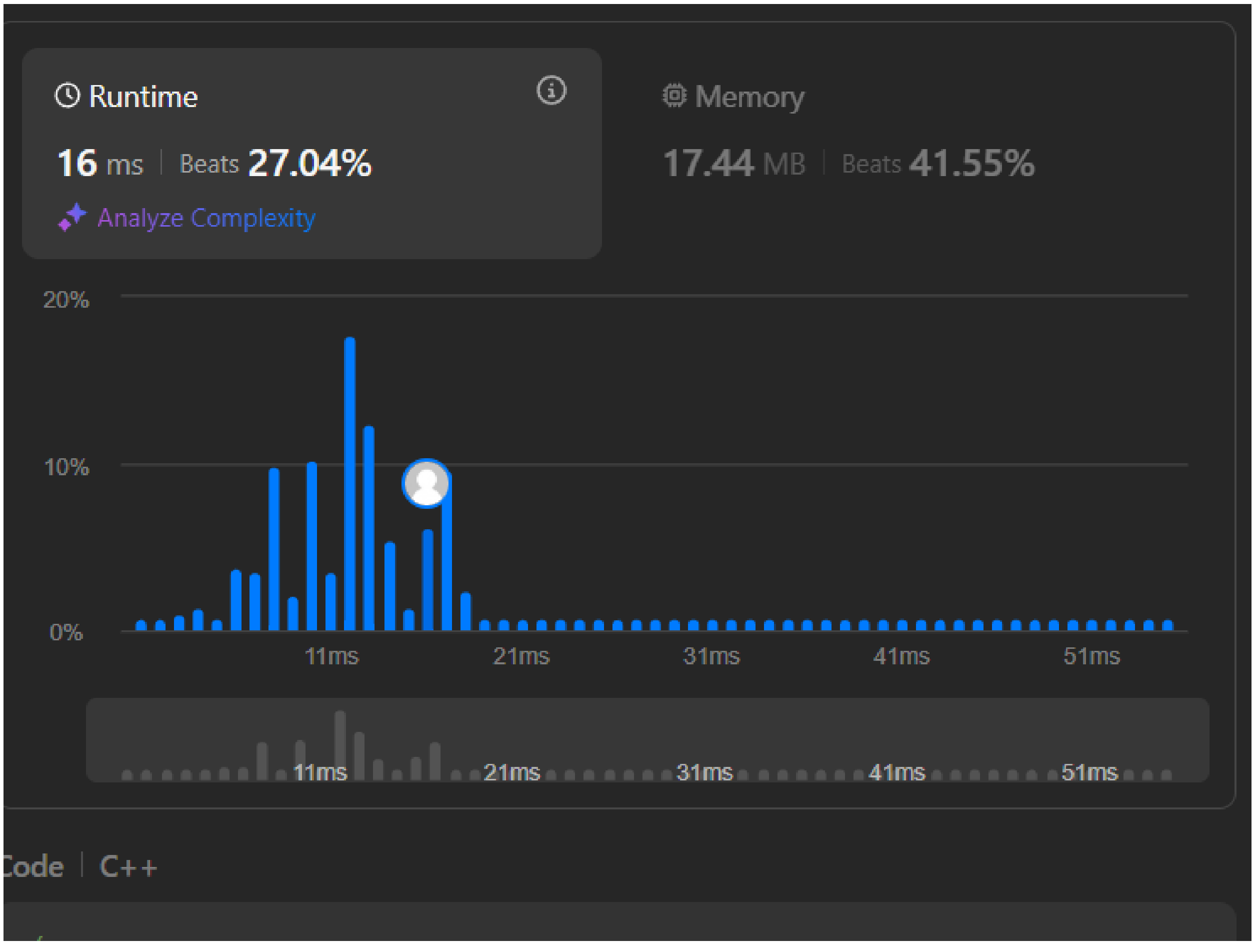
Analyze Complexity

Memory

14.96 MB | Beats 93.90%

<u>Lowest Common Ancestor of a Binary Tree</u>

```
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;
    }
};
```



Runtime

16 ms | Beats 27.04%

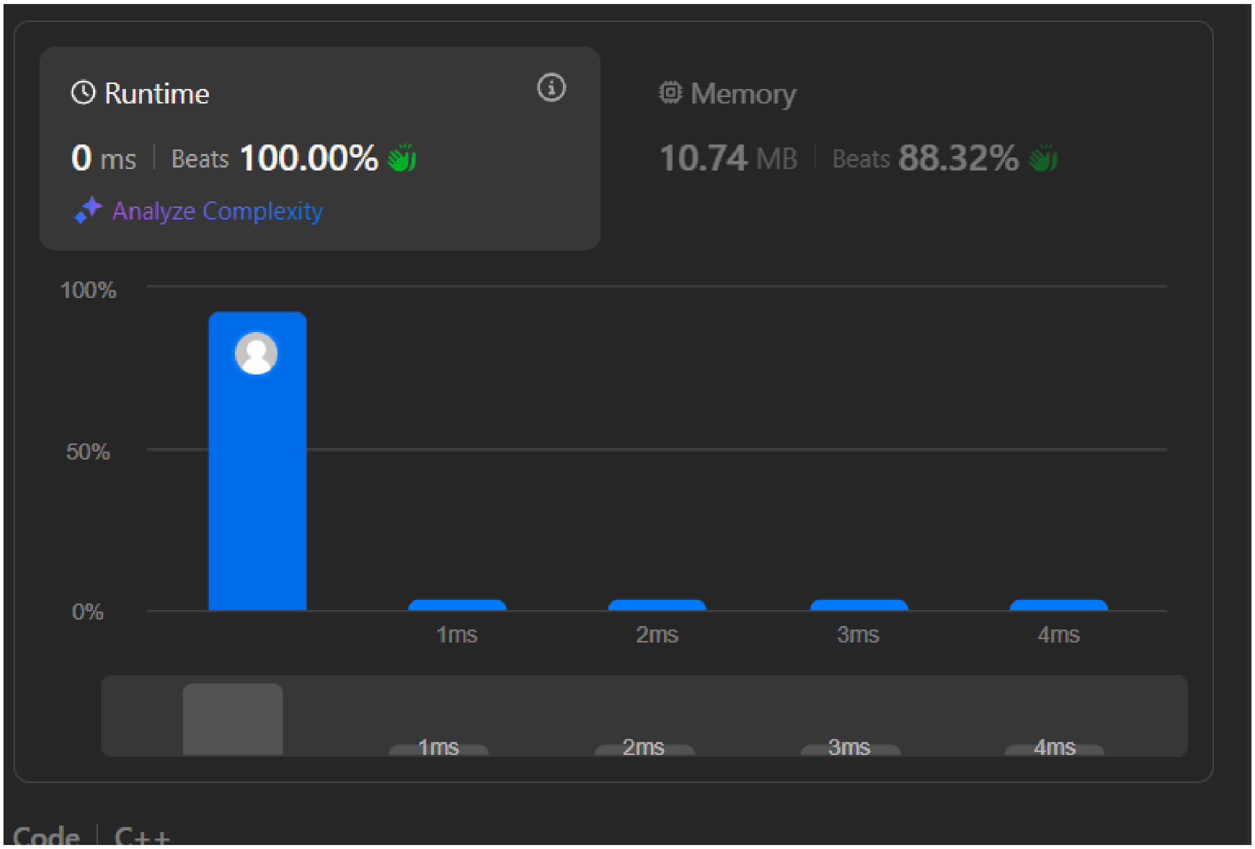
Analyze Complexity

Memory

17.44 MB | Beats 41.55%

Binary Tree Inorder Traversal

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



Runtime

0 ms | Beats 100.00%

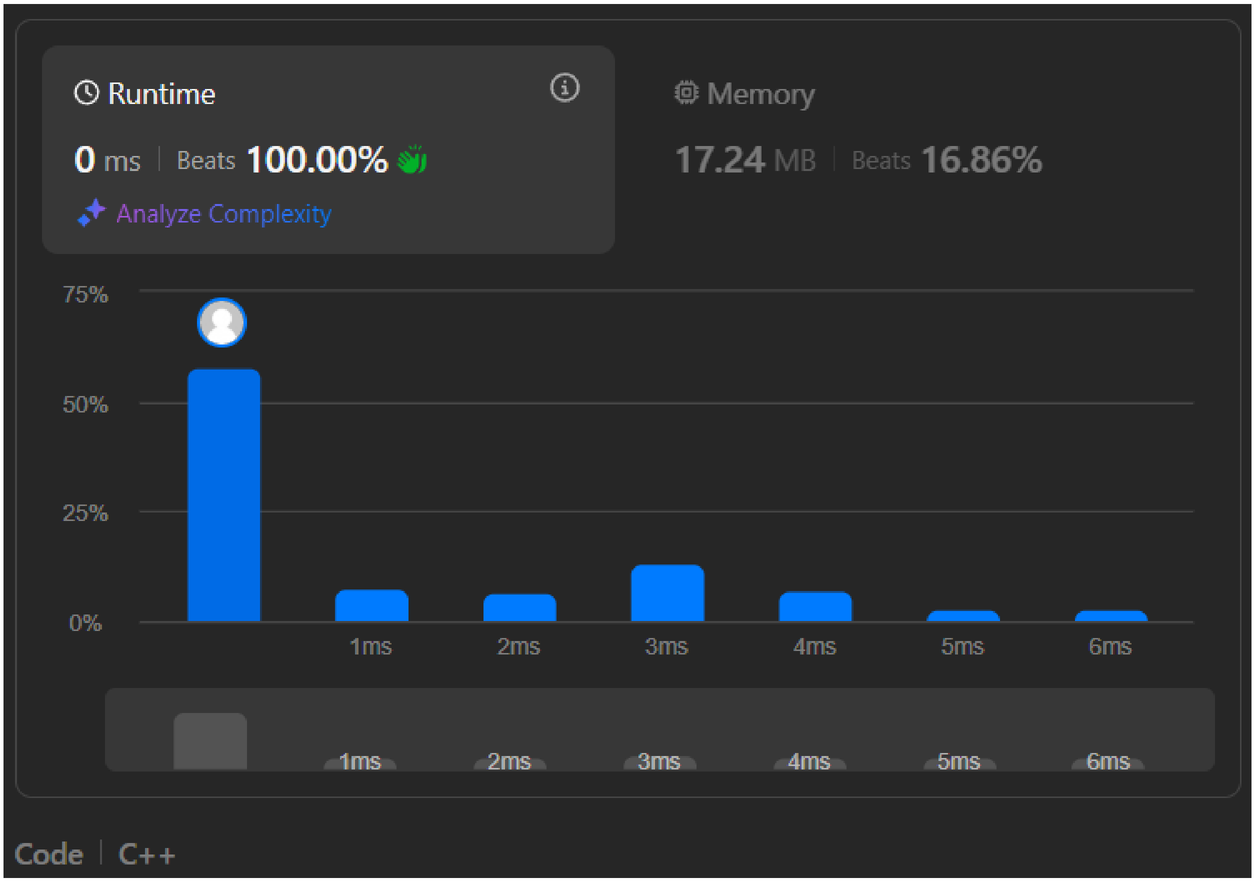
Analyze Complexity

Memory

10.74 MB | Beats 88.32%

<u>Binary Tree Level Order Traversal</u>

```
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 size = q.size();
            vector<int> level;

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

            result.push_back(level);
        }

        return result;
    }
};
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

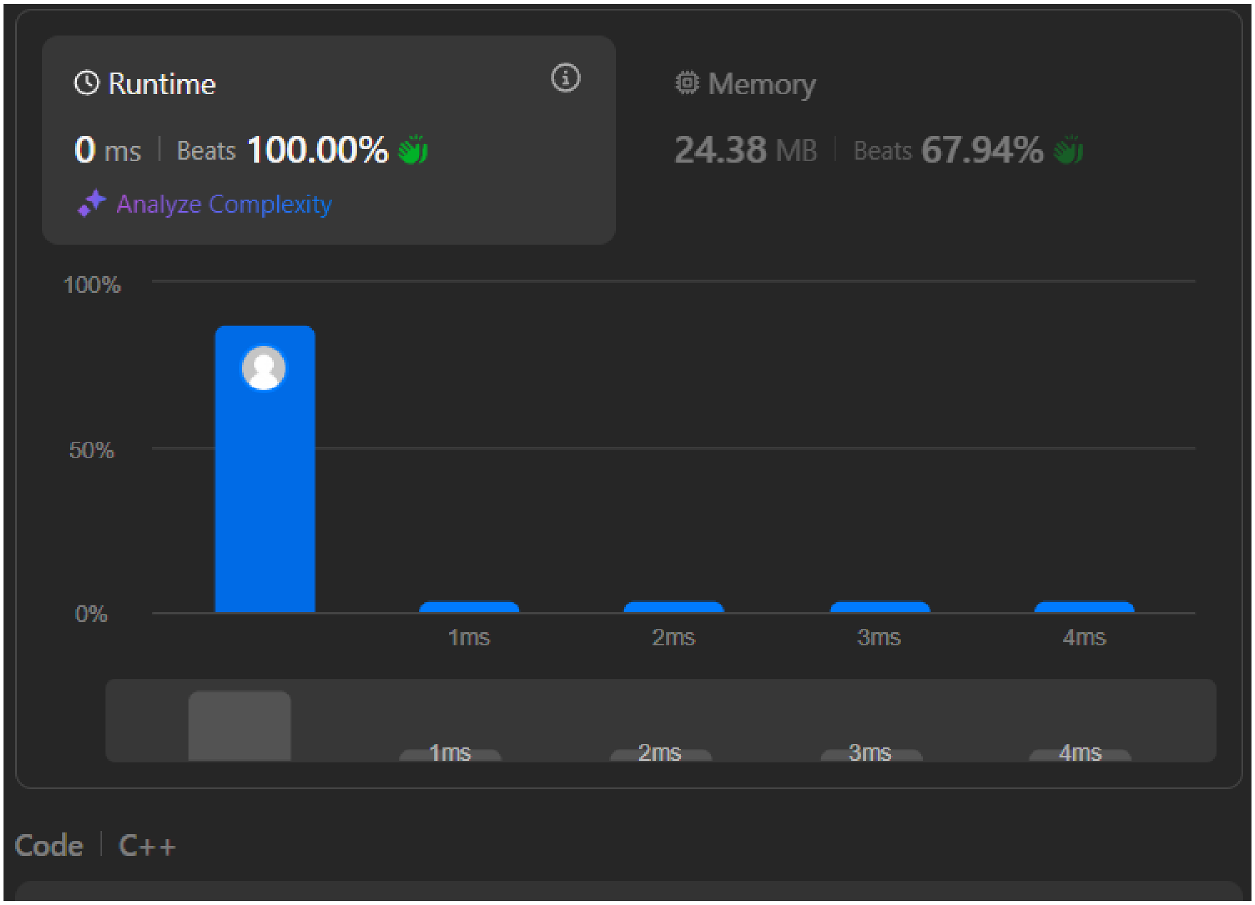
Memory

17.24 MB | Beats 16.86%

Code | C++

Kth Smallest Element in a BST

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0, result = 0;
        inorder(root, k, count, result);
        return result;
    }

private:
    void inorder(TreeNode* root, int k, int& count, int& result) {
        if (!root) return;

        inorder(root->left, k, count, result);
        count++;
        if (count == k) {
            result = root->val;
            return;
        }

        inorder(root->right, k, count, result);
    }
};
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

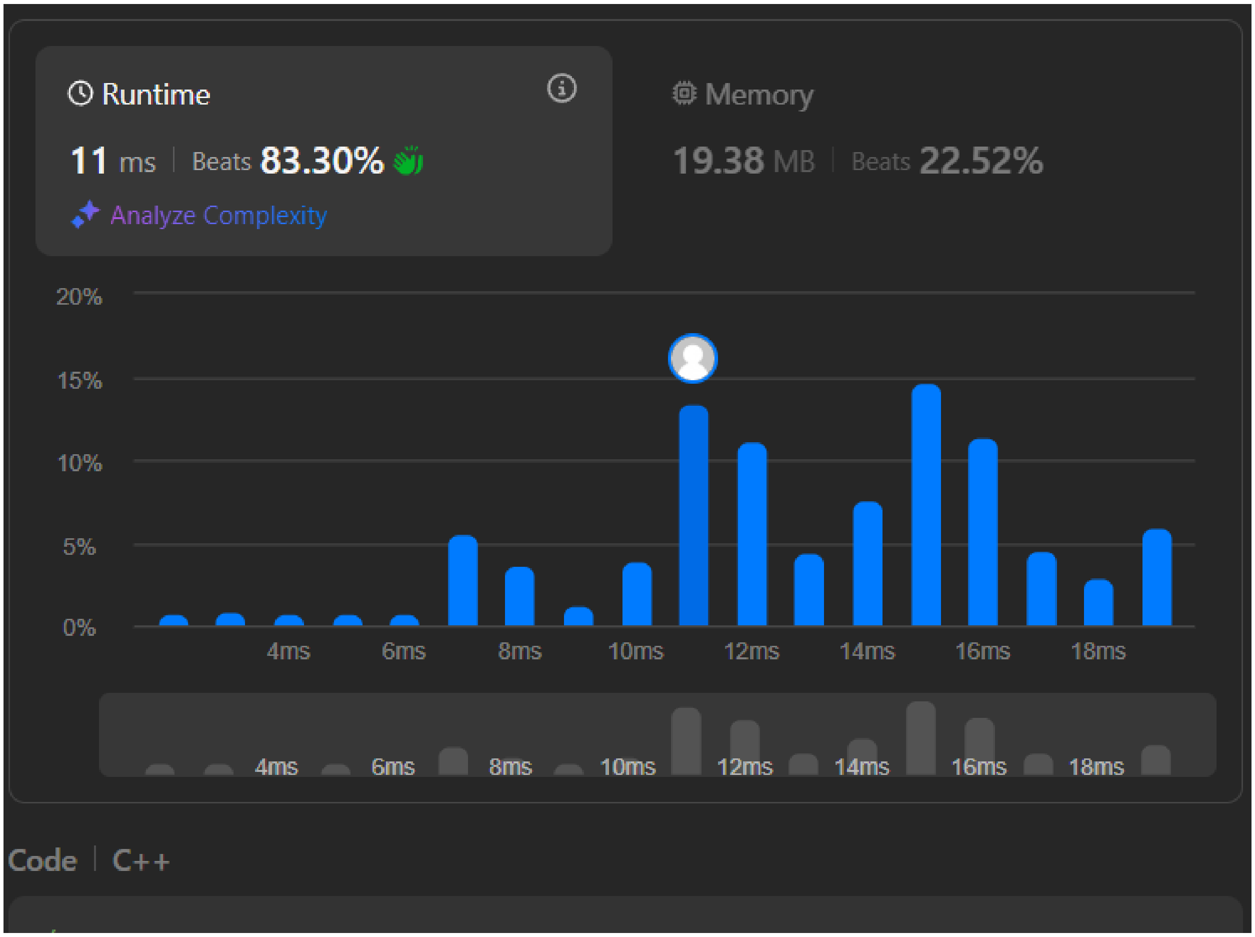
Memory

24.38 MB | Beats 67.94%

Populating Next Right Pointers 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();
            Node* prev = nullptr;

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

                if (prev) prev->next = node;
                prev = node;

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

        return root;
    }
};
```



Runtime

11 ms | Beats 83.30%

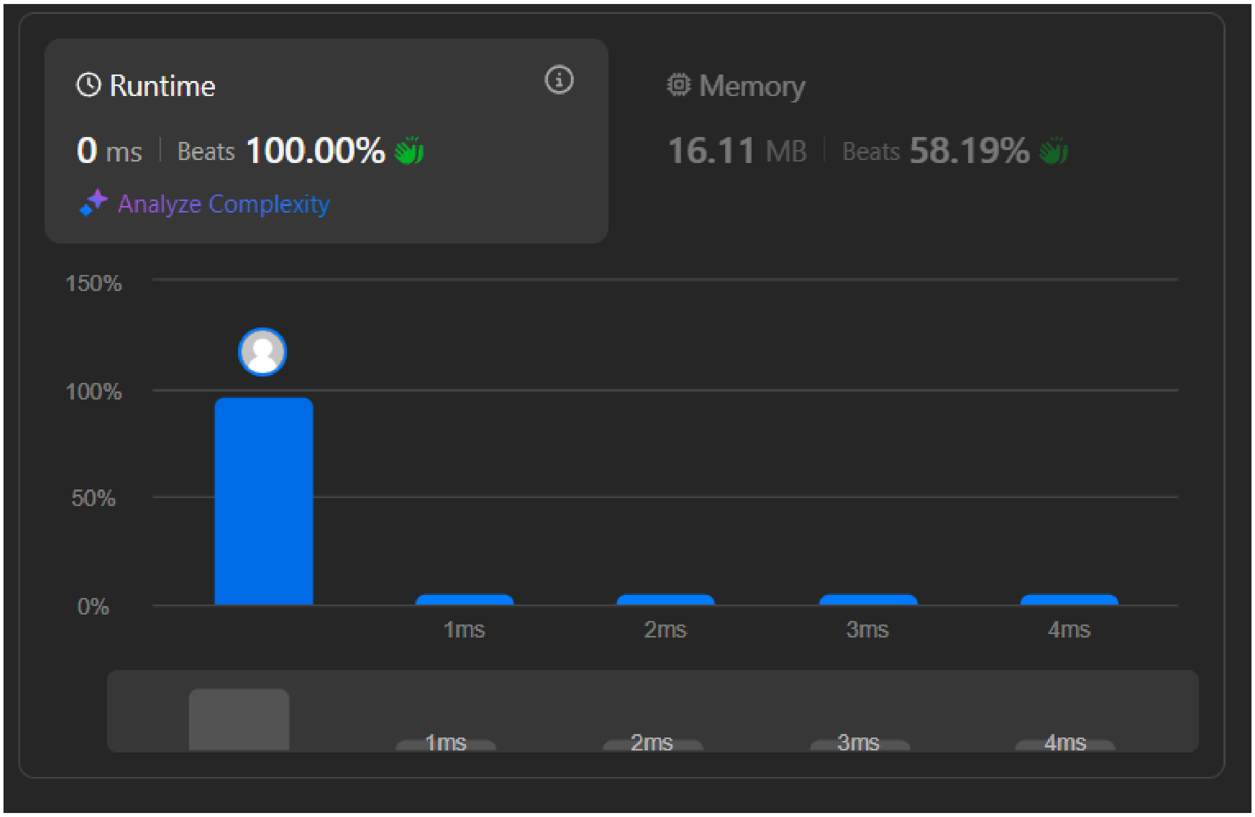
Memory

19.38 MB | Beats 22.52%

<u>**Sum of Left Leaves**</u>

```
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;  // If left child is a leaf, add its value
    }

    return sum + sumOfLeftLeaves(root->left) + sumOfLeftLeaves(root->right);
```

```
  }
};
```



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

16.11 MB | Beats 58.19%