EXPERIMENT 5.1

Student Name: Sweta Sharma **UID:** 22BCS11536
Branch: BE-CSE **Section/Group:** 22BCS_IOT_637-'B'
Semester: 6th **Date of Performance:** 20-09-25
Subject Name: AP LAB **Subject Code:** 22CSP-351

1) Aim:

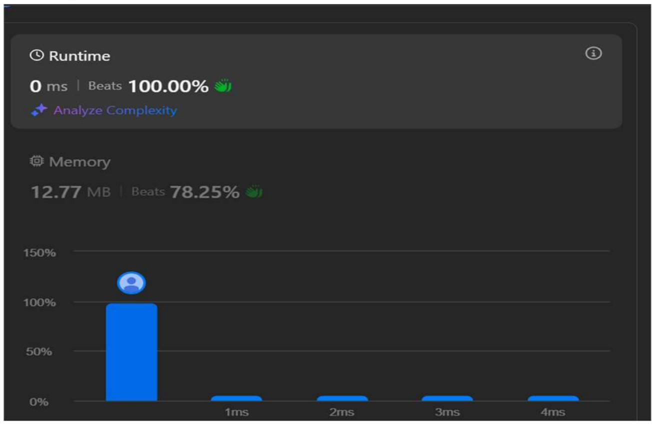
Same Tree

2) Implementation/Code: class Solution {

```
  public:
      bool isSameTree(TreeNode* p, TreeNode* q) {
          if (!p && !q) return true;  // Both trees are empty
          if (!p || !q) return false; // One tree is empty, the other is not
          if (p->val != q->val) return false; // Values are different

          // Recursively check left and right subtrees
          return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
      }
  };
```

3) Output:



Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
12.77 MB | Beats 78.25%

4) Complexity:

- **Time Complexity**: O(n2)
- **Space Complexity**: O(1)

PROBLEM 2

1. Aim:

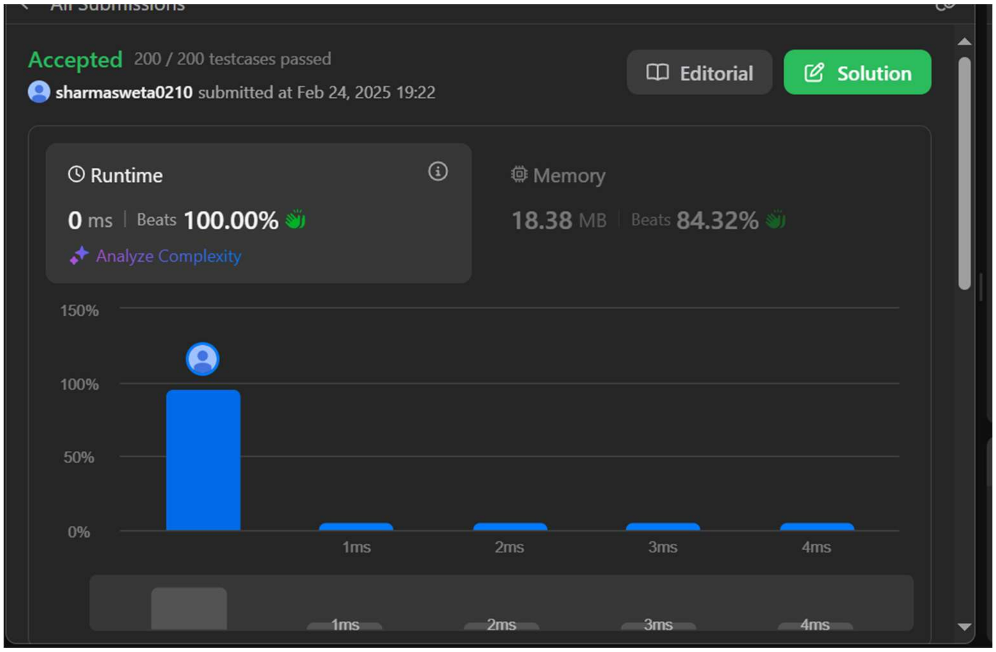
Symmetric Tree

2. Implementation/Code:

```
class Solution {
public:
   bool areMirrImg(TreeNode* root1, TreeNode* root2){
      if(!root1 && !root2){
         return true;
      }
      if(!root1 || !root2){
         return false;
      }
      return (root1->val == root2->val) && (areMirrImg(root1->left,root2->right)) &&
(areMirrImg(root1->right,root2->left));
   }
   bool isSymmetric(TreeNode* root) {
      if(!root){
         return true;
      }
```

```
        return areMirrImg(root->left,root->right);
      }
    };
```

3. Output:



All Submissions

Accepted 200 / 200 testcases passed

sharmasweta0210 submitted at Feb 24, 2025 19:22

Runtime

0 ms | Beats 100.00%

Memory

18.38 MB | Beats 84.32%

5. Complexity:

- **Time Complexity:** O(n) •
Space Complexity: O(1)

PROBLEM 3

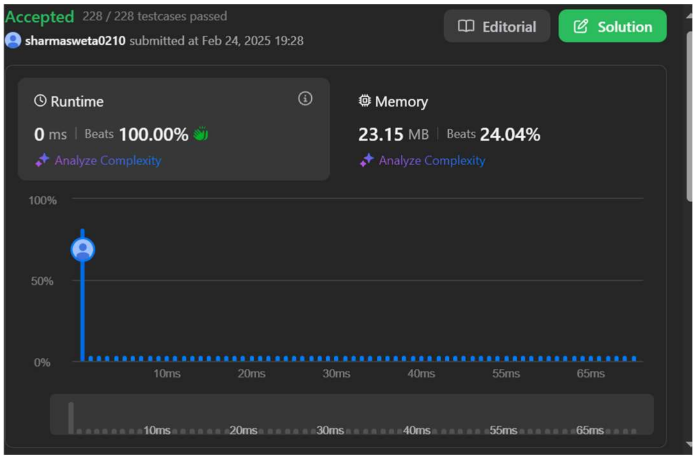
1) Aim:
Balanced Binary Tree

2) Implementation/Code:

```
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        return checkHeight(root) != -1;
    }

    int checkHeight(TreeNode* node) {
        if (!node) return 0;
        int leftHeight = checkHeight(node->left);
        if (leftHeight == -1) return -1;
        int rightHeight = checkHeight(node->right);
        if (rightHeight == -1) return -1;
        if (abs(leftHeight - rightHeight) > 1) return -1;
        return max(leftHeight, rightHeight) + 1;
    }
};
```

4) Output:



Accepted 228 / 228 testcases passed
sharmasweta0210 submitted at Feb 24, 2025 19:28

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
23.15 MB | Beats 24.04%
Analyze Complexity

5) Complexity:

- **Time Complexity:** O(n)

- **Space Complexity:** O(n)

PROBLEM 4

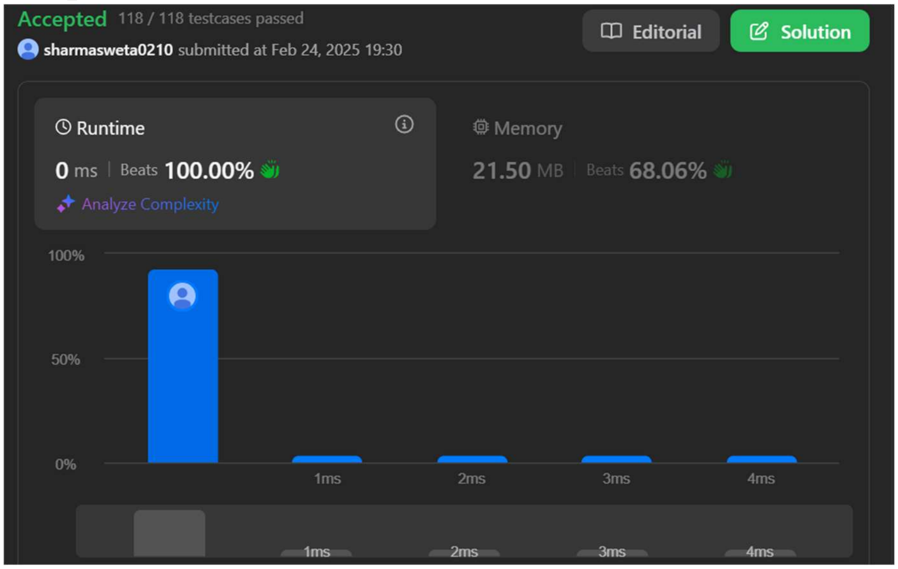
1) Aim:

Path Sum

2) Implementation/Code:

```
class Solution {
 public:
  bool hasPathSum(TreeNode* root, int sum) {
    if (root == nullptr)
      return false;
    if (root->val == sum && root->left == nullptr && root->right == nullptr)
      return true;
    return hasPathSum(root->left, sum - root->val) ||
           hasPathSum(root->right, sum - root->val);
  }
};
```

4) Output:



Accepted 118 / 118 testcases passed

sharmasweta0210 submitted at Feb 24, 2025 19:30

Runtime
0 ms | Beats 100.00%

Memory
21.50 MB | Beats 68.06%

5) Complexity:

- **Time Complexity:** O(1) •

Space Complexity: O(n)

PROBLEM 5

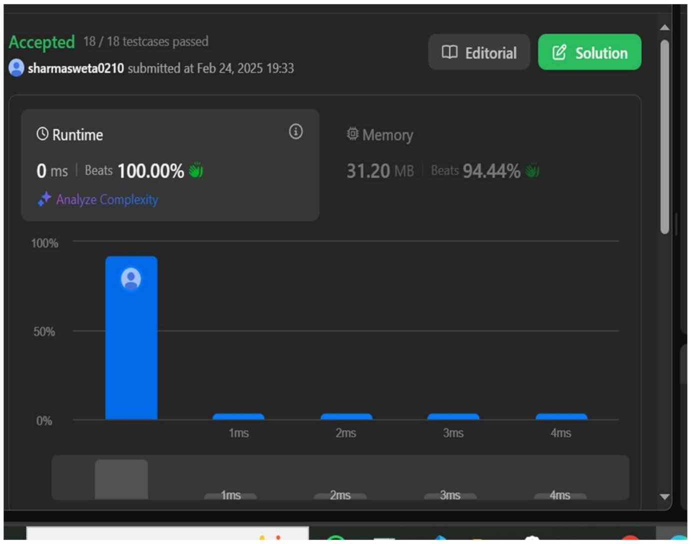
3) Aim:

Count Complete Tree Nodes.

4) Implementation/Code:

```
class Solution {
 public:
  int countNodes(TreeNode* root) {
    if (root == nullptr)
      return 0;
    TreeNode* left = root;
    TreeNode* right = root;
    int heightL = 0;
    int heightR = 0;
    while (left != nullptr) {
      ++heightL;
      left = left->left;
    }
    while (right != nullptr) {
      ++heightR;
      right = right->right;
    }
    if (heightL == heightR)
      return pow(2, heightL) - 1;
    return 1 + countNodes(root->left) + countNodes(root->right);
  }
};
```

5) Output:



Accepted 18 / 18 testcases passed

sharmasweta0210 submitted at Feb 24, 2025 19:33

Runtime

0 ms | Beats 100.00%

Memory

31.20 MB | Beats 94.44%

6) Complexity:

- **Time Complexity:** O(1) •

Space Complexity: O(n)

PROBLEM 6

1) Aim:

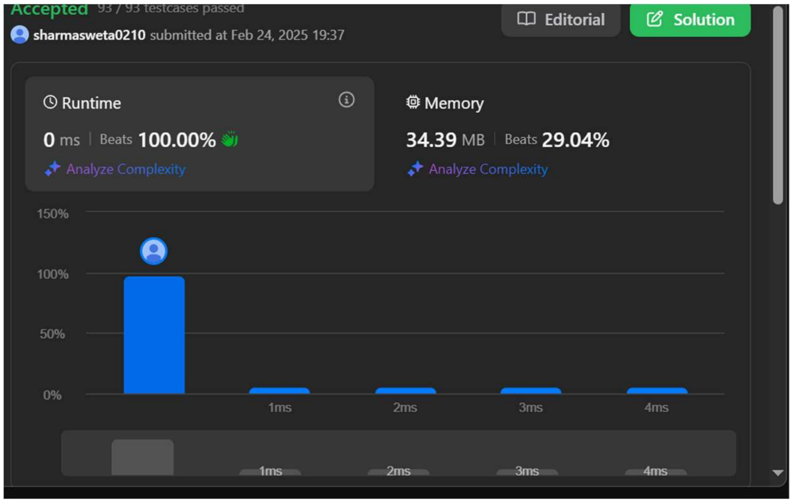
Delete Node in a BST

2) Implementation/Code:

```
class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        TreeNode* iter = root, *par = nullptr;
                    // search for key node & keep a pointer to current node's parent
        while(iter && iter -> val != key) {
            par = iter;
            if(iter -> val < key) iter = iter -> right;
            else iter = iter -> left;
        }
        if(!iter) return root;                              // node not found  => Case:1
        // iter is the node to be deleted

        // node found with less than two children  => Case-2/3/4 combined
        if(!iter -> left or !iter -> right) {
            auto child = iter -> left ? iter -> left : iter -> right;    // find child node of iter if it exists
            if(!par) root = child;                              // iter is root node. Update root as child of iter
            else if(par -> left == iter) par -> left = child;           // iter is left child. Update its parent's left pointer
as iter's child
            else par -> right = child;                              // Else update parent's right pointer as iter's child
        }
        // node found with both children => Case-5
        else {
            auto cur = iter;                                    // cur maintains a reference to the node to be deleted
            par = iter, iter = iter -> right;                          // go to right subtree
            while(iter -> left) par = iter, iter = iter -> left;          // and find smallest node in that right subtree
            cur -> val = iter -> val;                              // delete by replacing with smallest node found
                              // smallest node replaced from right subtree may have a right child.
                              // So update that node's parent to hold the right child
            if(par -> left == iter) par -> left = iter -> right;
            else par -> right = iter -> right;
        }
                      // dont show the interviewer that you are a leaker :)
        delete iter;        // free the memory
        return root;
    }
};
```

3) Output:



Accepted 93 / 93 testcases passed

sharmasweta0210 submitted at Feb 24, 2025 19:37

Runtime

0 ms | Beats 100.00%

Memory

34.39 MB | Beats 29.04%

4) Complexity:

- **Time Complexity:** O(1)
- **Space Complexity:** O(n)

PROBLEM 7

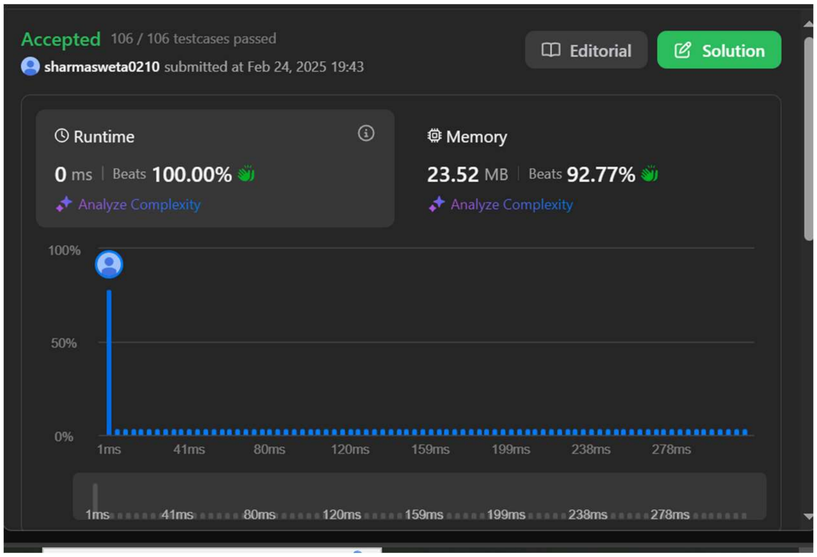
1) Aim:

Diameter of Bianry Tree

2) Implementation/Code:

```
class Solution {
public:
    pair<int, int> diameterOfBinaryTreeFast(TreeNode* root){
        if(!root){
            pair<int, int> p = make_pair(0, 0);
            return p;
        }
        pair<int, int> ans;
        pair<int, int> left =  diameterOfBinaryTreeFast(root->left);
        pair<int, int> right =  diameterOfBinaryTreeFast(root->right);
        ans.first = max(left.first, max(right.first, left.second+right.second+1));
        ans.second = max(left.second, right.second) + 1;
        return ans;
    }
    int diameterOfBinaryTree(TreeNode* root) {
        return diameterOfBinaryTreeFast(root).first - 1;
    }
};
```

3) Output:



Accepted 106 / 106 testcases passed

sharmasweta0210 submitted at Feb 24, 2025 19:43

Runtime

0 ms | Beats 100.00%

Memory

23.52 MB | Beats 92.77%

4) Complexity:

- **Time Complexity:** O(1)

- **Space Complexity:** O(n)