PROJECT TOPIC:KTH SMALLEST ELEMENT IN BST

SUBJECT:DATA STRUCTURE

GROUP MEMBERS NAME AND CU ID:

MIAN SHEHRYAR AHMAD CU:4128-2023

M.SAQLAIN CU 4122-2023

SYED JIBRAN CU 4151-2023

PROJECT NAME:KTH SMALLEST ELEMENT IN BST

. Project Overview

In this project, we will implement an algorithm to find the Kth smallest element in a Binary

Search Tree (BST) using C++. The BST follows the properties where:

The left subtree contains values less than the root.

The right subtree contains values greater than the root.

We will use in-order traversal (since it visits nodes in sorted order) and count the elements until we reach the Kth smallest element.

1. Introduction

A Binary Search Tree (BST) is a data structure that maintains a sorted order, allowing efficient searching, insertion, and deletion. In this project, we will find the Kth smallest element in a BST using various approaches.

2. Problem Statement

Given a Binary Search Tree (BST) and an integer K, find the Kth smallest element in the BST.

Example

Input

```
BST:          5
            /   \
           3     8
          / \   / \
         2   4 7   9
K = 3
```

Output

```
3rd smallest element: 4
```

3. Project Plan

Milestones

1. Implement the BST (Insert, Display)

2. Find the Kth smallest element using in-order traversal

3. Optimize using iterative in-order traversal with a stack

4. Test with various cases

5. Enhance with a balanced BST for efficiency

4. Implementation

4.1 BST Node Structure

Each node contains:

•An integer data

•A pointer to the left and right child nodes

4.2 Inserting Nodes into BST

A function to insert nodes while maintaining BST properties.

4.3Finding Kth Smallest Element

Approach 1: Recursive In-order Traversal

°Perform in-order traversal and maintain a counter.

Approach 2: Iterative In-order Traversal using Stack

°Uses a stack to avoid recursion overhead.

Approach 3: Optimized with Augmented BST

°Modify BST nodes to store subtree sizes for quick lookups.

5 Code Implementation

5.1C++ Code for BST and Finding Kth Smallest Element

```
#include <iostream>
using namespace std;

// Structure of a BST Node
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) {
        data = val;
        left = right = nullptr;
    }
};

// Function to insert a node into the BST
TreeNode* insert(TreeNode* root, int val) {
    if (!root) return new TreeNode(val);
    if (val < root->data)
```

```
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

// Recursive in-order traversal to find Kth smallest element
void inorder(TreeNode* root, int &k, int &result) {
    if (!root) return;

    inorder(root->left, k, result); // Left subtree

    k--;
    if (k == 0) {  // If this is the kth smallest element
        result = root->data;
        return;
    }

    inorder(root->right, k, result); // Right subtree
}

// Function to find Kth smallest element
int kthSmallest(TreeNode* root, int k) {
    int result = -1;
```

```
    inorder(root, k, result);
    return result;
}

// Function to print BST (in-order)
void inorderPrint(TreeNode* root) {
    if (!root) return;
    inorderPrint(root->left);
    cout << root->data << " ";
    inorderPrint(root->right);
}

// Driver code
int main() {
    TreeNode* root = nullptr;
    int nodes[] = {20, 8, 22, 4, 12, 10, 14}; // Sample tree elements
    for (int val : nodes)
        root = insert(root, val);

    cout << "BST In-order Traversal: ";
    inorderPrint(root);
    cout << endl;

    int k = 3;
```

```
    cout << k << "rd smallest element: " << kthSmallest(root, k) << endl;

    return 0;
}
```

6.Explanation of Code

1. BST Insertion (insert function)

•Recursively inserts elements while maintaining BST rules.

2. Finding Kth Smallest (kthSmallest function)

•Uses in-order traversal to visit elements in sorted order.

•Decrements k each time a node is visited.

•When k reaches 0, stores the result.

3. Printing the BST (inorderPrint function)

•Displays BST elements in sorted order for validation.

4. Driver Code (main function)

•Creates a BST and inserts elements.

•Finds the Kth smallest element and displays it.

6. Example Output

BST In-order Traversal: 4 8 10 12 14 20 22

3rd smallest element: 10

7. Optimized Approach using Iterative In-order Traversal

If recursion is not preferred, we can use a stack:

```
#include <stack>

int kthSmallestIterative(TreeNode* root, int k) {
    stack<TreeNode*> st;
```

7. Future Enhancements

•Use Self-Balancing BST (AVL Tree) for efficiency.

•Implement Persistent BST for historical Kth smallest queries.

8. Conclusion

This project efficiently finds the Kth smallest element in a BST using different approaches. The iterative method is often preferred due to better space efficiency, while augmented BSTs are optimal for frequent queries.