Kth Smallest Element Sum in BST - Pseudocode

Problem Statement

Given a Binary Search Tree and a number K, find the sum of all elements that are smaller than or equal to the Kth smallest element.

Approach 1: Simple Inorder Traversal (Easiest to understand)

```
ALGORITHM FindKthSmallestSum_Simple(root, K)
INPUT:
   - root: Root node of BST
   - K: Position of smallest element we want (1-indexed)
OUTPUT:
   - Sum of all elements <= Kth smallest element

BEGIN
   CREATE empty list called "elements"

   CALL InorderTraversal(root, elements, K)

   IF elements has at least K items THEN
      RETURN sum of first K elements in the list
   ELSE
      RETURN sum of all elements in the list
   END IF
END

ALGORITHM InorderTraversal(node, elements, K)
INPUT:
   - node: Current node being visited
   - elements: List to store elements in sorted order
   - K: Maximum elements we need

BEGIN
   IF node is NULL OR elements already has K items THEN
      RETURN
   END IF
```

```
    // Visit left subtree first (smaller elements)
    CALL InorderTraversal(node.left, elements, K)

    // Process current node if we haven't collected K elements yet
    IF elements has less than K items THEN
        ADD node.value to elements list
    END IF

    // Visit right subtree (larger elements)
    CALL InorderTraversal(node.right, elements, K)
END
```

Approach 2: Optimized with Augmented BST (O(height) time)

```
ALGORITHM FindKthSmallestSum_Optimized(root, K)
INPUT:
    - root: Root node of augmented BST
    - K: Position of smallest element (1-indexed)
OUTPUT:
    - Sum of elements <= Kth smallest

PREREQUISITE: Each node has additional fields:
    - leftCount: Number of nodes in left subtree
    - leftSum: Sum of all values in left subtree

BEGIN
    IF root is NULL THEN
        RETURN 0
    END IF

    SET leftCount = root.leftCount

    // Case 1: Kth smallest element is in left subtree
    IF K <= leftCount THEN
        RETURN FindKthSmallestSum_Optimized(root.left, K)

    // Case 2: Current root is exactly the Kth smallest element
```

```
    ELSE IF K equals (leftCount + 1) THEN
        // Sum all left subtree elements + current root
        RETURN root.leftSum + root.value

    // Case 3: Kth smallest element is in right subtree
    ELSE
        // We need the (K - leftCount - 1)th element from right subtree
        SET remainingK = K - leftCount - 1
        SET rightSum = FindKthSmallestSum_Optimized(root.right, remainingK)

        // Total sum = left subtree + root + partial right subtree
        RETURN root.leftSum + root.value + rightSum
    END IF
END
```

Preprocessing for Augmented BST

```
ALGORITHM CalculateAugmentedInfo(node)
INPUT: node - Current node to process
OUTPUT: (totalNodes, totalSum) - Count and sum of entire subtree

BEGIN
    IF node is NULL THEN
        RETURN (0, 0)
    END IF

    // Process left subtree
    SET (leftNodes, leftSum) = CalculateAugmentedInfo(node.left)

    // Process right subtree
    SET (rightNodes, rightSum) = CalculateAugmentedInfo(node.right)

    // Update current node's augmented information
    SET node.leftCount = leftNodes
    SET node.leftSum = leftSum

    // Calculate totals for this subtree
    SET totalNodes = leftNodes + 1 + rightNodes
```

```
    SET totalSum = leftSum + node.value + rightSum

    RETURN (totalNodes, totalSum)
END
```

Example Walkthrough

Given Tree:

```
    8
   /\
  7  10
 /  /\
2  9  13
```

For K = 3:

1. **Inorder traversal gives:** [2, 7, 8, 9, 10, 13]
2. **3rd smallest element is:** 8
3. **Elements ≤ 8 are:** [2, 7, 8]
4. **Sum:** 2 + 7 + 8 = 17

Optimized approach steps:

1. At root (8): leftCount = 2, K = 3
2. Since K > leftCount: K equals leftCount + 1, so root is Kth smallest
3. Return leftSum + root.value = 9 + 8 = 17

Time & Space Complexity

Simple Approach:

- Time: O(K) - We only traverse until we get K elements
- Space: O(K) for storing elements + O(height) for recursion

Optimized Approach:

- Time: O(height) - We traverse only one path down the tree
- Space: O(height) for recursion
- Preprocessing: O(N) time to calculate augmented info