Convert Binary Search Tree (BST) to Greater Sum Tree (GST) [LeetCode](https://leetcode.com/problems/convert-bst-to-greater-tree/description/)

Given the root of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

* The left subtree of a node contains only nodes with keys **less than** the node's key.
* The right subtree of a node contains only nodes with keys **greater than** the node's key.
* Both the left and right subtrees must also be binary search trees.

Example:

4

/ \

1. 6

/ \ / \

0 2 5 7

\ \

1. 8

Output:

30

/ \

1. 21

/ \ / \

36 35 26 15

\ \

33 8

**Approach 1: Convert a binary search tree (BST) into a Greater Sum Tree (GST)**

* **Function Purpose**:
  + The **convertBST** function converts a BST into a Greater Sum Tree (GST).
* **Explanation**:
  + It uses two helper functions: **inorderTraversal** to calculate cumulative sums of nodes and **updateBST** to update the BST with cumulative sums.
  + Cumulative sums are stored in a vector, and values are updated in the BST by popping values from the vector.
* **Time Complexity**:
  + **O(N), where N is the number of nodes in the BST, as we perform an in-order traversal once.**
* **Space Complexity:**
  + **O(N), for the vector storing cumulative sums and the call stack space for recursion.**

**Approach 2: Convert a binary search tree (BST) into a Greater Sum Tree (GST) using an optimized approach**

* **Function Purpose**:
  + The **convertBSTOptimized** function converts a BST into a Greater Sum Tree (GST) using an optimized approach.
* **Explanation**:
  + It uses a helper function **convertBSTHelper** that performs the in-order traversal in reverse order, updating the nodes with the greater sum.
* **Time Complexity**:
  + **O(N), where N is the number of nodes in the BST, as we traverse all nodes once.**
* **Space Complexity:**
  + **O(H), where H is the height of the BST, for the call stack space.**

**Conclusion**:

* + **Approach 2 is more memory-efficient, and it has the same time complexity as Approach 1 (O(N)). It is the preferred approach.**