Documentation for the "Convert BST to Greater Tree"

1. Problem Statement

Given the root of a Binary Search Tree (BST), convert it into a Greater Tree such that every node’s value is changed to the original value plus the sum of all node values greater than the current node's value in the BST.

Constraints:

- The number of nodes is in the range [0, 10^4]
- $-10^4 \leq$ Node.val $\leq 10^4$
- All node values are unique
- The input tree is a valid BST

2. Intuition

In a BST:

- The right child contains greater values.
- The left child contains smaller values.

To compute the required transformation, we must accumulate all greater values for each node. The natural way to traverse values from highest to lowest is reverse in-order traversal (right → root → left).

3. Key Observations

- In-order traversal (left → root → right) gives ascending order.
- Reverse in-order traversal gives descending order.
- Keeping a running sum during traversal allows us to update each node efficiently.

4. Approach

- Initialize a global variable self.sum = 0.
- Traverse the tree using reverse in-order traversal.
- At each node:
    - Visit the right subtree first.
    - Update the node's value using the running sum.
    - Visit the left subtree.
- Return the modified root.

5. Edge Cases

- Empty tree (None) → Should return None.
- Single node → Should return the node with the same value.
- Left-skewed or right-skewed trees → Should be handled correctly.
- Large trees with max constraints → Should remain efficient.

6. Complexity Analysis

Time Complexity

- O(n): Every node is visited exactly once.

Space Complexity

- O(h): Stack space for recursion where h is the height of the tree.
    - Best case (balanced tree): O(log n)
    - Worst case (skewed tree): O(n)

7. Alternative Approaches

1. Morris Traversal (Threaded Binary Tree)

- Reduces space complexity to O(1) (no recursion/stack).
- More complex and less readable.

2. Iterative with Stack

- Use explicit stack for reverse in-order traversal.
- Easier to debug than recursion.

8. Test Cases

Input	Output	Description
[4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]	[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]	Standard full BST
[0,null,1]	[1,null,1]	Right-skewed BST
[5]	[5]	Single node
[]	[]	Empty tree
[2,1]	[2,3]	Small BST

9. Final Thoughts

This problem is an excellent example of using reverse in-order traversal to transform a BST while maintaining its properties. The recursive solution is elegant and readable. For large trees or constrained environments, iterative or Morris traversal methods are also worth exploring.