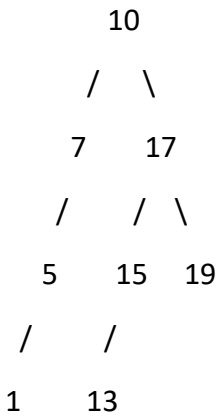
Balance an Unbalanced Binary Search Tree LeetCode

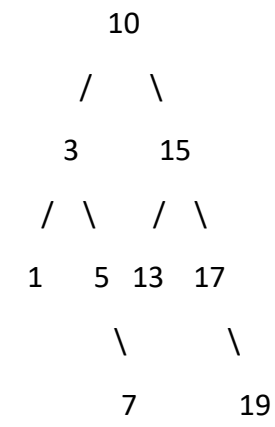
Given the root of a binary search tree, return *a **balanced** binary search tree with the same node values*. If there is more than one answer, return **any of them**.

A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.

Example:



Output:



Approach 1: Function to balance a binary search tree using Morris in-order traversal

- **Function Purpose**: Balance a binary search tree (BST) using Morris in-order traversal.
- **Explanation**:
    - Perform an in-order traversal of the BST using Morris Traversal to store values.
    - Create a balanced BST from the sorted values.
- **Time Complexity: O(N), where N is the number of nodes in the BST.**
- **Space Complexity: O(N) to store the in-order traversal result.**

Approach 2: Function to balance a binary search tree using Morris in-order traversal and store Nodes

- **Function Purpose**: Balance a binary search tree (BST) using Morris in-order traversal and store nodes.
- **Explanation**:
    - Perform an in-order traversal of the BST using Morris Traversal to store nodes.
    - Create a balanced BST from the sorted nodes.
- **Time Complexity: O(N), where N is the number of nodes in the BST.**
- **Space Complexity: O(N) to store the in-order traversal result.**

Conclusion:

- Both Approach 1 and Approach 2 provide a balanced BST from an unbalanced BST.
- Both have the same time complexity of O(N), but Approach 2 stores nodes in the in-order traversal instead of values, which can be useful in certain scenarios. The choice between the two approaches depends on whether you need to work with the values or nodes during the balancing process.