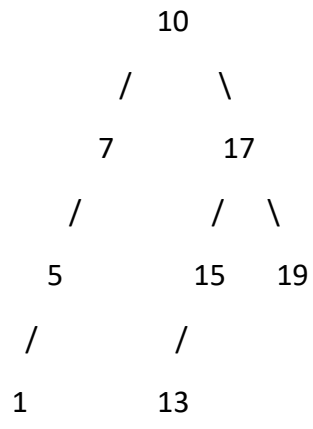
Merge Two Binary Search Trees into one Balanced Binary Tree CodeStudio

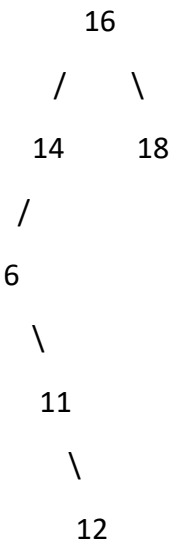
You are given Two Binary Search Tree of integers having M and N nodes merge the two trees and return the root of the newly created balanced binary tree.

Example:

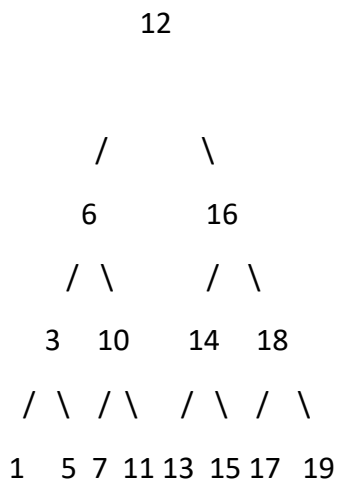
Tree 1:



Tree 2:



Output:



Inorder Traversal: 1 3 5 6 7 10 11 12 13 14 15 16 17 18 19

Approach 1: Merge two Binary Search Trees into a single Binary Search Tree using Inorder Traversal

- **Function Purpose:**
    - Merges two Binary Search Trees (BSTs) into a single balanced BST.
- **Explanation**:
    - Perform in-order traversal on both input BSTs and store the nodes in two separate arrays.
    - Merge the two sorted arrays into one.
    - Create a balanced Binary Search Tree from the merged array.
- **Time Complexity:**
    - **In-order traversal of each BST: O(N1) and O(N2), where N1 and N2 are the number of nodes in the respective BSTs.**
    - **Merging two sorted arrays: O(N1 + N2).**
    - **Creating a balanced BST: O(N1 + N2).**
    - **Total time complexity: O(N1 + N2).**
- **Space Complexity:**
    - **Space for storing in-order traversal arrays: O(N1 + N2).**
    - **Additional space for recursion and variables: O(H1 + H2), where H1 and H2 are the heights of the two BSTs.**
    - **Total space complexity: O(N1 + N2 + H1 + H2).**

Approach 2: Merge two BSTs into one optimized BST using Doubly Linked List

- **Function Purpose:**
    - Merges two Binary Search Trees (BSTs) into a single optimized balanced BST using doubly linked lists.
- **Explanation**:
    - Convert both input BSTs to sorted doubly linked lists.
    - Merge the two sorted doubly linked lists into one.
    - Create a balanced Binary Search Tree from the merged doubly linked list.
- **Time Complexity:**
    - **Converting BSTs to doubly linked lists: O(N1 + N2).**
    - **Merging sorted doubly linked lists: O(N1 + N2).**
    - **Creating a balanced BST from the merged doubly linked list: O(N1 + N2).**
- **Space Complexity:**
    - **Space for storing doubly linked lists: O(N1 + N2).**
    - **Additional space for recursion and variables: O(H1 + H2).**
    - **Total space complexity: O(N1 + N2 + H1 + H2).**

Conclusion:

- Both approaches have the same time and space complexities.
- The choice between Approach 1 and Approach 2 depends on the specific use case and preference.
- Approach 1 directly uses arrays, while Approach 2 uses doubly linked lists, which may be more memory-efficient in certain scenarios.
- **Approach 2 can be advantageous when memory efficiency is a concern, while Approach 1 may be preferable for simplicity.**