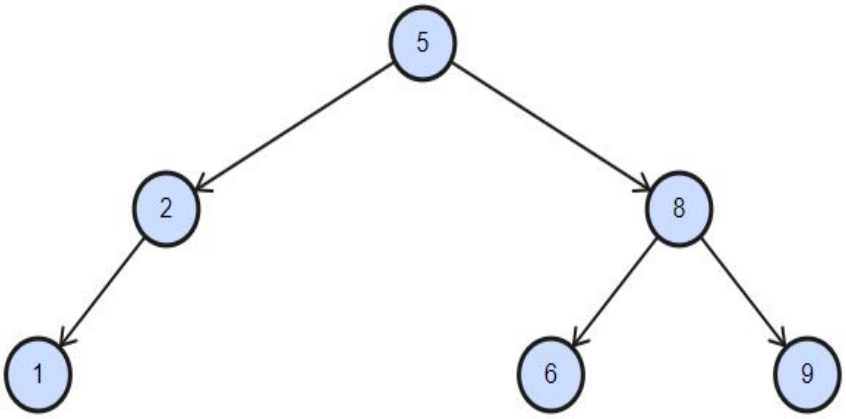
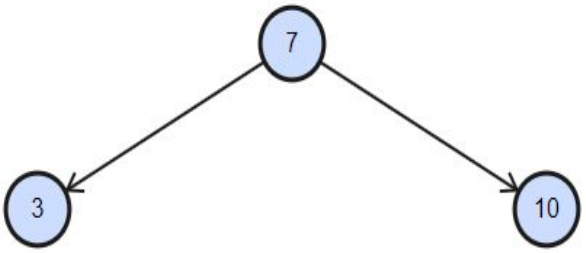
Merge two BSTs

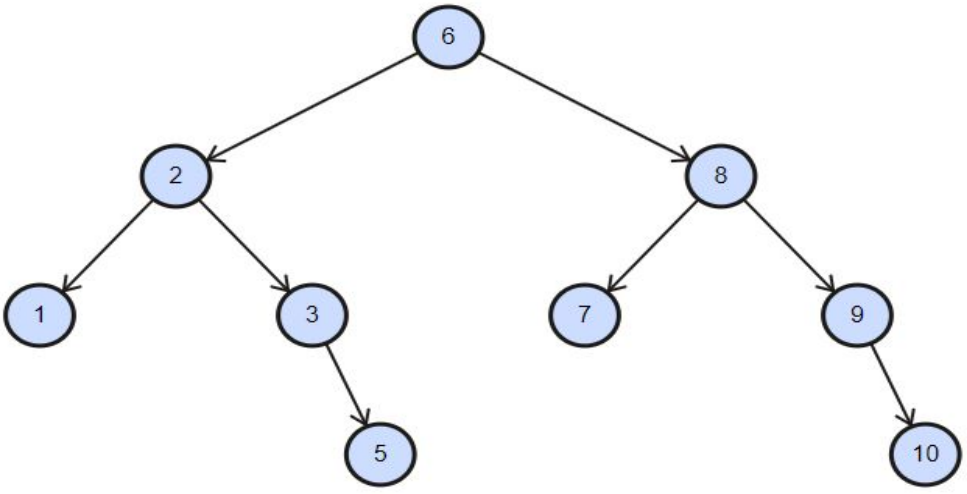
Problem Description: Given are two BSTs and our task is to merge the two BSTs and return the root of the new BST.



And



Are merged together to result in a BST:



(This is one possible BST by merging the above two BSTs.)

Solution Description: We can have multiple methods to solve this problem.

- Method 1 (Insert elements of the first tree to second)
  Take all elements of first BST one by one, and insert them into the second BST. Inserting an element to a self-balancing BST takes Logn time, where n is the size of the BST. So time complexity of this method is Log(n) + Log(n+1) … Log(m+n-1). The value of this expression will be between mLogn and mLog(m+n-1). For optimization, we can pick the smaller tree as the first tree.

- Method 2 (Merge Inorder Traversals)
  1) Do inorder traversal of the first tree and store the traversal in one temp array arr1[]. This step takes O(m) time.
  2) Do inorder traversal of the second tree and store the traversal in another temp array arr2[]. This step takes O(n) time.
  3) The arrays created in steps 1 and 2 are sorted arrays. Merge the two sorted arrays into one sorted array of size m + n. This step takes O(m+n) time.
  4) Construct a balanced tree from the merged array. This step takes O(m+n) time.

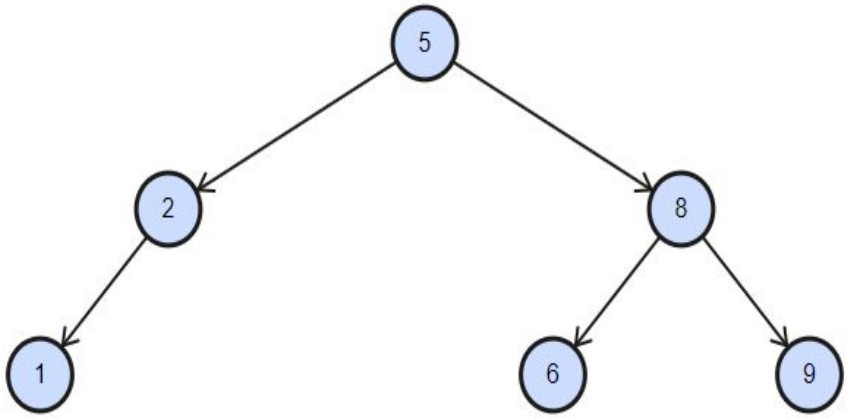
  The time complexity of this method is O(m+n) which is better than method 1. This method takes O(m+n) time even if the input trees are not balanced.

Now let's see the working of method 2:
Step 1: The first step is to do the inorder traversal of the BST and store it in one array.

Inorder Traversal of BST:
The inorder traversal gives us the nodes of BST in ascending order.
Algorithm:

1) Traverse the left subtree, i.e., call Inorder(left-subtree)
2) Visit the root.
3) Traverse the right subtree, i.e., call Inorder(right-subtree)

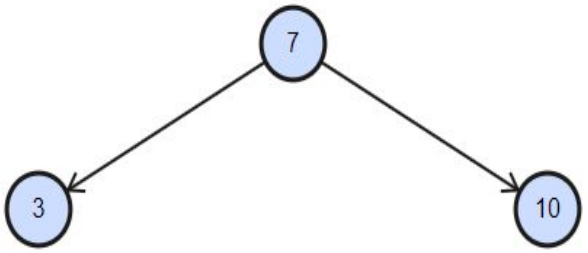
The inorder traversal of the following BST would be:



Inorder Traversal:

1	2	5	6	8	9

Similarly, the inorder traversal of second BST would be:



Inorder traversal:

3	7	10

Step 2: Now merge the two arrays storing the inorder traversal of both the BSTs.

Merge Two Sorted Arrays:
We can merge the two sorted arrays in O(m+n), where m and n are the size of the arrays.
Algorithm:

1. Create an array arr3[] of size m + n.
2. Simultaneously traverse arr1[] and arr2[].
    a. Pick smaller of current elements in arr1[] and arr2[], copy this smaller element to next position in arr3[] and move ahead in arr3[] and the array whose element is picked.
3. If there are remaining elements in arr1[] or arr2[], copy them also in arr3[].

For the arrays, we created in the previous step, merging them into a single sorted array would give us:

Step 3: Covert the merged array into a BST.

Sorted array to BST:
Here we use the sorted property of the array to ensure the construction of balanced BST where we divide the array into two equal parts and assign the mid-value as a root node. To be in alignment with the definition of a Binary Search Tree, the elements in the array to the left of the mid-value would contribute to the left subtree while the elements in the array to the right of the mid-value, would contribute to the right subtree.

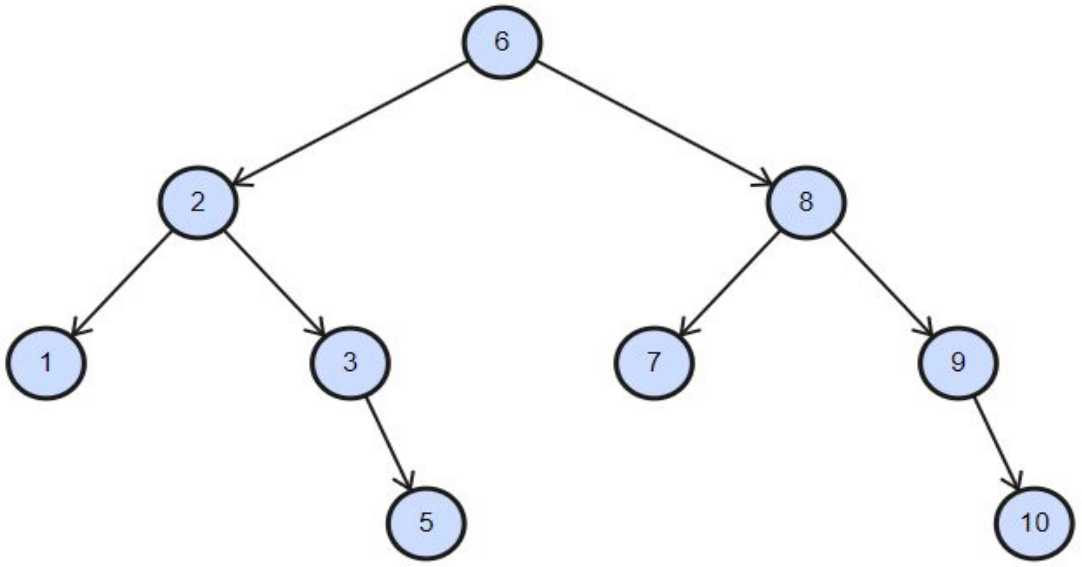
Algorithm:

1. Get the Middle of the array and make it root.
2. Recursively do the steps for the left half and right half.
    - Get the middle of the left half and make it the left child of the root.
    - Get the middle of the right half and make it the right child of the.

Pseudo-code:
Method: toBST
Input: A sorted array arr and two integer variables start and end
// Start and end denote the list elements to be converted to BST.

```
// Base case
if(start>end)
   return null

mid=(start+end)/2

// Create new node of arr[mid]
new_node=Node(arr[mid])

// Recursively find the left subtree of the node and link it to left of the node
new_node.left=toBST(arr,start,mid-1)

// Recursively find the right subtree of the node and link it to right of the node
new_node.right=toBST(arr,mid+1,end)

return new_node
```

For the sorted array generated in the previous step the BST would look like this:



Now, all we need to do is to combine all these steps to get the root of the merged BST:

Pseudo-code:

Method: mergeBST

Input: Two BSTs roots root1 and root2

// Store the inorder traversal of the first BST and store it in arr1.
arr1=inorder(root1)
// Store the inorder traversal of the second BST and store it in arr2.
arr2=inorder(root2)
// Merge arr1 and arr2 to form a sorted array arr3.
arr3=merge(arr1,arr2)
//Generate a BST from arr3.
new_root= toBST(arr3)
return new_root