Check for Identical BSTs without building the trees

Given two arrays which represent a sequence of keys. Imagine we make a Binary Search Tree (BST) from each array. We need to tell whether two BSTs will be identical or not without actually constructing the tree.

Examples

For example, the input arrays are {2, 4, 3, 1} and {2, 1, 4, 3} will construct the same tree

```
 Let the input arrays be a[] and b[]

Example 1:
a[] = {2, 4, 1, 3} will construct following tree.
   2
 /  \
1    4
    /
   3
b[] = {2, 4, 3, 1} will also also construct the same tree.
   2
 /  \
1    4
    /
   3
So the output is "True"

Example 2:
a[] = {8, 3, 6, 1, 4, 7, 10, 14, 13}
b[] = {8, 10, 14, 3, 6, 4, 1, 7, 13}

They both construct the same following BST, so output is "True"
            8
         /    \
       3       10
     /  \        \
    1     6       14
        /   \     /
       4     7   13
```

Solution:

According to BST property, elements of left subtree must be smaller and elements of right subtree must be greater than root.

Two arrays represent sane BST if for every element x, the elements in left and right subtrees of x appear after it in both arrays. And same is true for roots of left and right subtrees.

The idea is to check of if next smaller and greater elements are same in both arrays. Same properties are recursively checked for left and right subtrees. The idea looks simple, but implementation requires checking all conditions for all elements. Following is an interesting recursive implementation of the idea.

```
// A C program to check for Identical BSTs without building the trees
#include<stdio.h>
#include<limits.h>
#include<stdbool.h>

/* The main function that checks if two arrays a[] and b[] of size n construct
  same BST. The two values 'min' and 'max' decide whether the call is made for
  left subtree or right subtree of a parent element. The indexes i1 and i2 are
  the indexes in (a[] and b[]) after which we search the left or right child.
  Initially, the call is made for INT_MIN and INT_MAX as 'min' and 'max'
  respectively, because root has no parent.
  i1 and i2 are just after the indexes of the parent element in a[] and b[]. */
bool isSameBSTUtil(int a[], int b[], int n, int i1, int i2, int min, int max)
{
   int j, k;

   /* Search for a value satisfying the constraints of min and max in a[] and
      b[]. If the parent element is a leaf node then there must be some
      elements in a[] and b[] satisfying constraint. */
   for (j=i1; j<n; j++)
       if (a[j]>min && a[j]<max)
           break;
   for (k=i2; k<n; k++)
       if (b[k]>min && b[k]<max)
           break;

   /* If the parent element is leaf in both arrays */
   if (j==n && k==n)
       return true;

   /* Return false if any of the following is true
      a) If the parent element is leaf in one array, but non-leaf in other.
      b) The elements satisfying constraints are not same. We either search
         for left child or right child of the parent element (decinded by min
         and max values). The child found must be same in both arrays */
   if (((j==n)^(k==n)) || a[j]!=b[k])
       return false;

   /* Make the current child as parent and recursively check for left and right
      subtrees of it. Note that we can also pass a[k] in place of a[j] as they
      are both are same */
   return isSameBSTUtil(a, b, n, j+1, k+1, a[j], max) &&  // Right Subtree
          isSameBSTUtil(a, b, n, j+1, k+1, min, a[j]);    // Left Subtree
}

// A wrapper over isSameBSTUtil()
bool isSameBST(int a[], int b[], int n)
{
   return isSameBSTUtil(a, b, n, 0, 0, INT_MIN, INT_MAX);
}

// Driver program to test above functions
int main()
{
   int a[] = {8, 3, 6, 1, 4, 7, 10, 14, 13};
   int b[] = {8, 10, 14, 3, 6, 4, 1, 7, 13};
   int n=sizeof(a)/sizeof(a[0]);
   printf("%s\n", isSameBST(a, b, n)?
             "BSTs are same":"BSTs not same");
   return 0;
}
```

Output:

```
BSTs are same
```