14) COUNT PAIRS FROM 2 BST'S WHOSE SUM IS EQUAL TO GIVEN SUM.

METHOD 1:
In This method,first take the inOrder Traversal of the two BST's into the two different arrays.
So we get two sorted arrays.Then we do 2 pointer approach,in which the first pointer is taken at the zero index of the first vector and the second pointer is taken at the last index of the second vector.
Then we follow the same method which we did in the sorted array to find the pairs whose sum is equal to the given sum.
Space Complexity : O(n+m)
Time Complexity : O(n+m)

METHOD 2:
Using normal searching for every element in BST 1 in the BST 2 for the required sum.
Space Complexity : O(1)
Time Complexity : O(N log M)

METHOD 3:
Using unordered_map for the method 2 to avoid the searching time of logN.
Space Complexity : O(n+m)
Time Complexity : O(n+m)

LINK OF METHODS : Count pairs from 2 BST whose sum is equal to given value "X" | BST | …

CODE FOR METHOD 1:

```
class Solution
{
public:
   void function(Node *root,vector<int>&v){
      if(root!=NULL){
         function(root->left,v);
         v.push_back(root->data);
         function(root->right,v);
      }
   }
   int countPairs(Node* root1, Node* root2, int x)
   {
      int count=0;
      vector<int> a,b;
      function(root1,a);
      function(root2,b);
      int i=0,j=b.size()-1;
      while(i<a.size() && j>=0){
         if(a[i]+b[j]==x){
            count++;
            i++;
            j--;
         }
         else if(a[i]+b[j]<x){
```

```
            i++;
        }
        else{
            j--;
        }
    }
    return count;
  }
};
```

CODE OF METHOD 2:

```
class Solution
{
public:
  bool search(Node *root,int n){
    if(root->data==n){
        return true;
    }
    else if(root->data<n){
        if(root->right==NULL){
            return false;
        }
        else{
            return search(root->right,n);
        }
    }
    else{
        if(root->left==NULL){
            return false;
        }
        else{
            return search(root->left,n);
        }
    }
  }
  int function(Node *root1,Node *root2,int x){
    if(root1!=NULL){
        int left=function(root1->left,root2,x);
        int right=function(root1->right,root2,x);
        int t=x-root1->data;
        if(search(root2,t)==true){
            return 1+left+right;
        }
        else{
            return left+right;
        }
    }
    else{
        return 0;
```

```
        }
    }
    int countPairs(Node* root1, Node* root2, int x){
        int count=function(root1,root2,x);
        return count;
    }
};
```