**Find Common Nodes in two BSTs: -**

**Easy** Accuracy: **51.7%** Submissions: **43K+** Points: **2**

Given two Binary Search Trees. Find the nodes that are common in both of them, ie- find the intersection of the two BSTs.

**Note**: Return the common nodes in **sorted**order.

**Example 1:**

**Input:**

**BST1:**

                 5

              /     \

            1       10

          /   \     /

         0     4    7

                    \

                      9

**BST2:**

               10

             /    \

            7     20

          /   \

         4     9

**Output:** 4 7 9 10

**Example 2:**

**Input:**

**BST1:**

  10

  / \

  2 11

  / \

 1 3

**BST2:**

  2

  / \

  1 3

**Output:** 1 2 3

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**findCommon()** that takes roots of the two BSTs as input parameters and returns a list of integers containing the common nodes in sorted order.

**Expected Time Complexity:**O(N1 + N2) where N1 and N2 are the sizes of the 2 BSTs.  
**Expected Auxiliary Space:**O(H1 + H2) where H1 and H2 are the heights of the 2 BSTs.

**Constraints:**  
1 <= Number of Nodes <= 105  
1 <= Node data <= 109

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// Tree Node

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// Function to Build Tree

Node\* buildTree(string str)

{

// Corner Case

if(str.length() == 0 || str[0] == 'N')

return NULL;

// Creating vector of strings from input

// string after spliting by space

vector<string> ip;

istringstream iss(str);

for(string str; iss >> str; )

ip.push\_back(str);

// Create the root of the tree

Node\* root = new Node(stoi(ip[0]));

// Push the root to the queue

queue<Node\*> queue;

queue.push(root);

// Starting from the second element

int i = 1;

while(!queue.empty() && i < ip.size()) {

// Get and remove the front of the queue

Node\* currNode = queue.front();

queue.pop();

// Get the current node's value from the string

string currVal = ip[i];

// If the left child is not null

if(currVal != "N") {

// Create the left child for the current node

currNode->left = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->left);

}

// For the right child

i++;

if(i >= ip.size())

break;

currVal = ip[i];

// If the right child is not null

if(currVal != "N") {

// Create the right child for the current node

currNode->right = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

// } Driver Code Ends

class Solution

{

public:

void fill(Node \*root, unordered\_map<int,int> &mp){

if(!root) return;

fill(root->left, mp);

++mp[root->data];

fill(root->right, mp);

}

void helper(Node \*root, unordered\_map<int,int> &mp, vector<int> &ans){

if(!root) return;

helper(root->left, mp, ans);

if(mp[root->data] > 0){

ans.push\_back(root->data);

--mp[root->data];

}

helper(root->right, mp, ans);

}

//Function to find the nodes that are common in both BST.

vector <int> findCommon(Node \*root1, Node \*root2)

{ unordered\_map<int,int> mp;

fill(root1, mp);

vector<int> ans;

helper(root2, mp, ans);

return ans;

}

};

//{ Driver Code Starts.

int main()

{

int t;

cin>>t;

getchar();

while(t--)

{

string s;

getline(cin,s);

Node\* root1 = buildTree(s);

getline(cin,s);

Node\* root2 = buildTree(s);

Solution ob;

vector <int> res = ob.findCommon(root1, root2);

for (int i : res)

cout << i << " ";

cout<< endl;

}

return 1;

}

// } Driver Code Ends

**T.C: - O(max(M,N))**

**S.C: - O(M)**

**M, N = count of nodes in BST1, BST2**