

LCA of BST

You are given a binary search tree of integers with N nodes. You are also given references to two nodes 'P' and 'Q' from this BST.

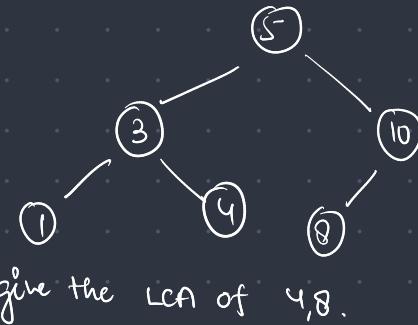
Your task is to find the lowest common ancestor(LCA) of these two given nodes.

The lowest common ancestor for two nodes P and Q is defined as the lowest node that has both P and Q as descendants (where we allow a node to be a descendant of itself)

A binary search tree (BST) is a binary tree data structure which has the following properties.

- The left subtree of a node contains only nodes with data less than the node's data.
- The right subtree of a node contains only nodes with data greater than the node's data.
- Both the left and right subtrees must also be binary search trees.

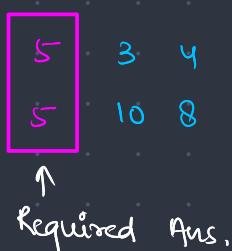
→ Approach :-



↳ Brute force: - We will store the paths of both the nodes
- Traverse both arrays & return the last common node.



5 3 X 4



5 3 X X 10 8

```

void getPath(TreeNode *root, vector<TreeNode *> &p, int x) {
    if (root == NULL)
        return;
    p.push_back(root);
    if (root->data == x)
        return;

    if (p.back()->data != x)
        getPath(root->left, p, x);
    if (p.back()->data != x)
        getPath(root->right, p, x);
    if (p.back()->data != x)
        p.pop_back();
}

TreeNode *LCAinaBST(TreeNode *root, TreeNode *P, TreeNode *Q) {
    if (root == NULL)
        return root;

    vector<TreeNode *> p;
    vector<TreeNode *> q;
    getPath(root, p, P->data);
    getPath(root, q, Q->data);

    int p_size = p.size(), q_size = q.size();

    int i = 0;
    while (i < p_size && i < q_size && p[i]->data == q[i]->data) {
        i++;
    }

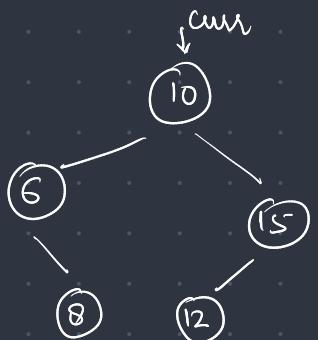
    return p[i - 1];
}

```

Time Complexity : $O(n)$

Space complexity : $O(n)$

→ Optimised Approach :- Using the property of BST.



3 cases are possible



→ We can easily find which case is it by using BST properties.

```

#include <bits/stdc++.h>
TreeNode* LCAinaBST(TreeNode* root, TreeNode* P, TreeNode* Q)
{
    TreeNode* curr = root;

    while(curr != NULL){
        if(curr->data > P->data && curr->data > Q->data) curr = curr->left;
        else if(curr->data < P->data && curr->data < Q->data) curr = curr->right;
        else return curr;
    }

    return NULL;
}

```

Time complexity : $O(\log n)$

Space complexity : $O(1)$