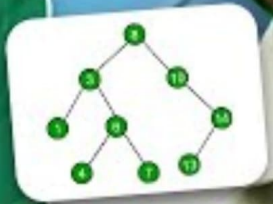
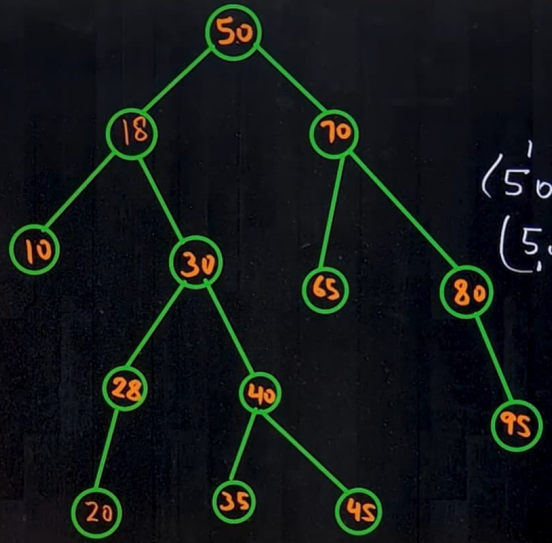
BINARY SEARCH TREE

- Lowest Common Ancestor in a BST
- Print BST Element in Given Range
- Check whether BST contains Dead End
- Common Node in two BST



Lowest Common Ancestor in a BST

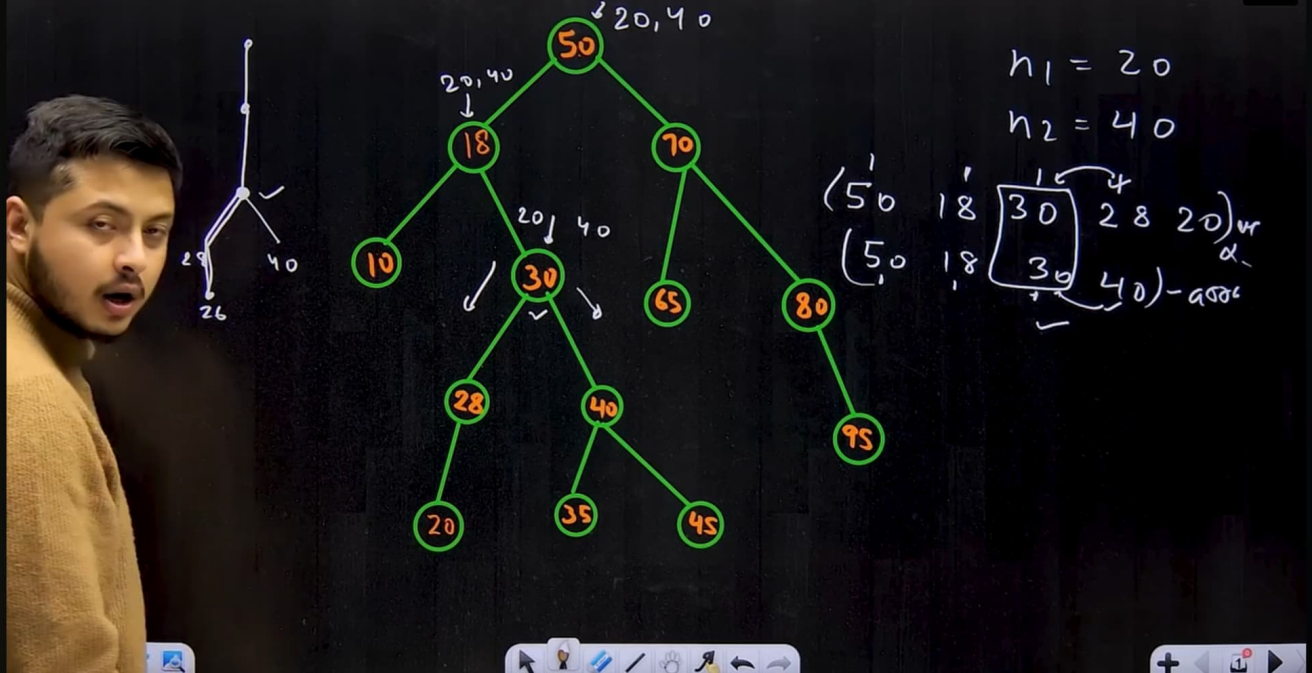


$n_1 = 20$

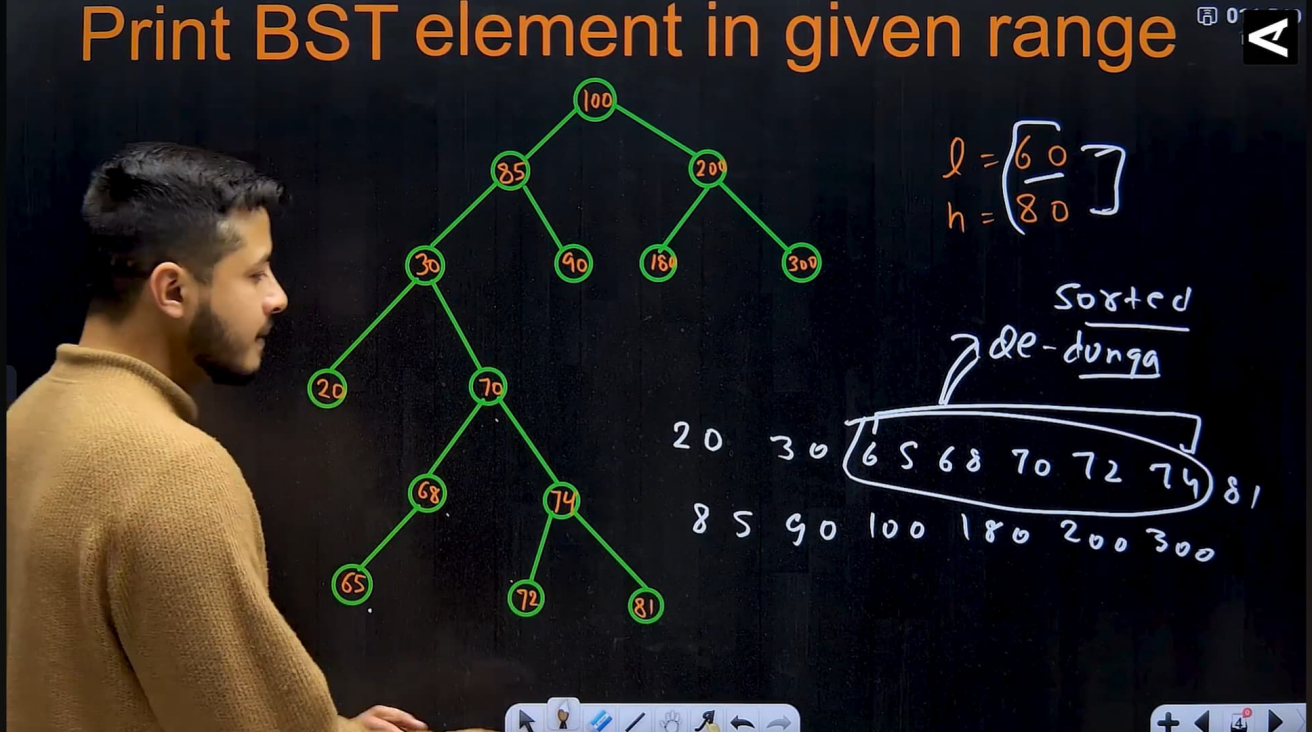
$n_2 = 40$

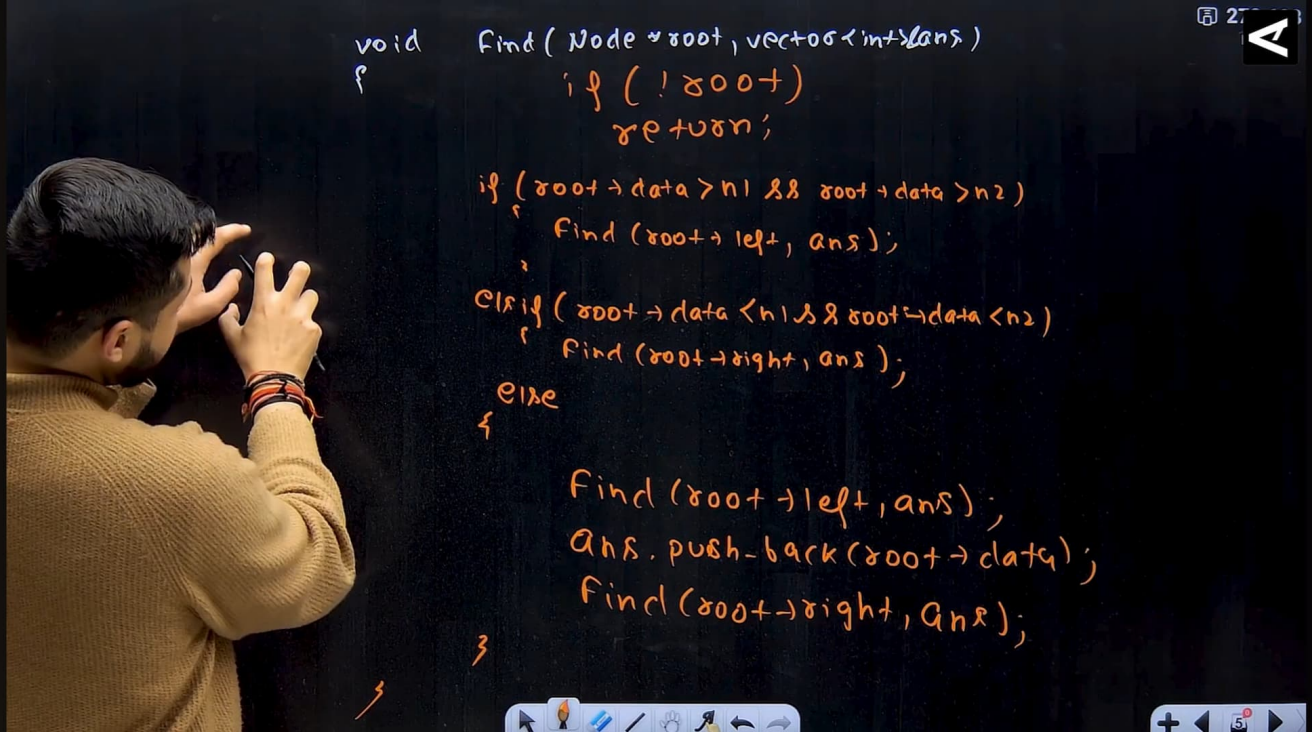
(50 18 30 28 20) ur
(50 18 30 40) - ans

Lowest Common Ancestor in a BST



Print BST element in given range




void Find(Node *root, vector<int>&ans)
{
if(!root)
return;
if(root->data > n1 && root->data > n2)
Find(root->left, ans);
else if(root->data < n1 && root->data < n2)
Find(root->right, ans);
else
{
Find(root->left, ans);
ans.push_back(root->data);
Find(root->right, ans);
}

LeetCode

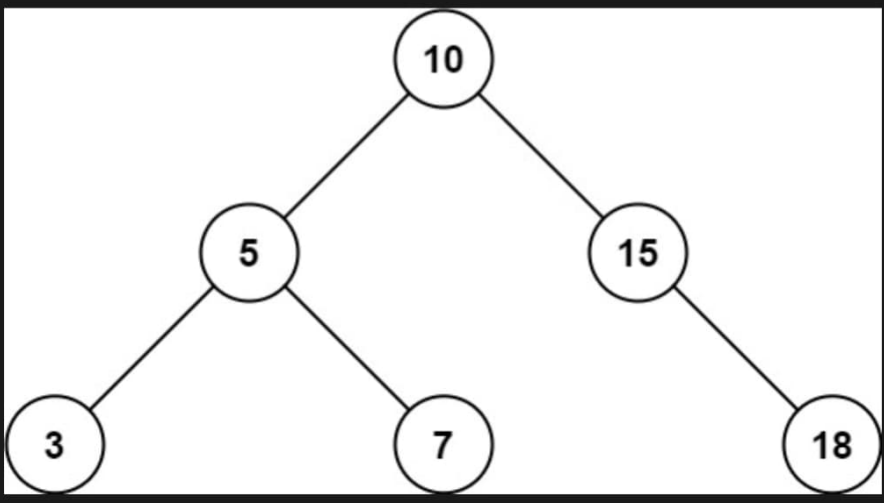
Description | Solutions | Submissions

938. Range Sum of BST

Easy | Topics | Companies

Given the `root` node of a binary search tree and two integers `low` and `high`, return *the sum of values of all nodes with a value in the* ***inclusive*** *range* `[low, high]`.

Example 1:




```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.
```

Example 2:

BST with Dead End

Difficulty: **Medium** Accuracy: **35.99%**
Submissions: **98K+** Points: **4**

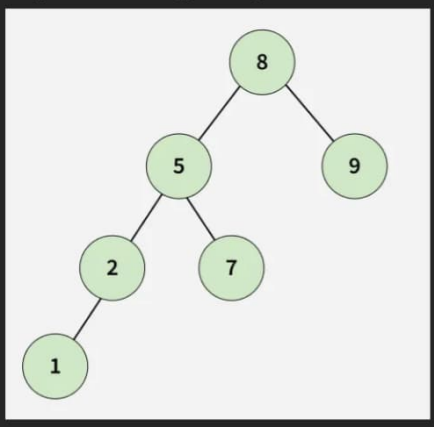
You are given a Binary Search Tree (BST) containing unique positive integers greater than 0.

Your task is to determine whether the BST contains a **dead end**.

Note: A **dead end** is a **leaf node** in the BST such that no new node can be inserted in the BST at or below this node while maintaining the BST property and the constraint that all node values must be > 0.

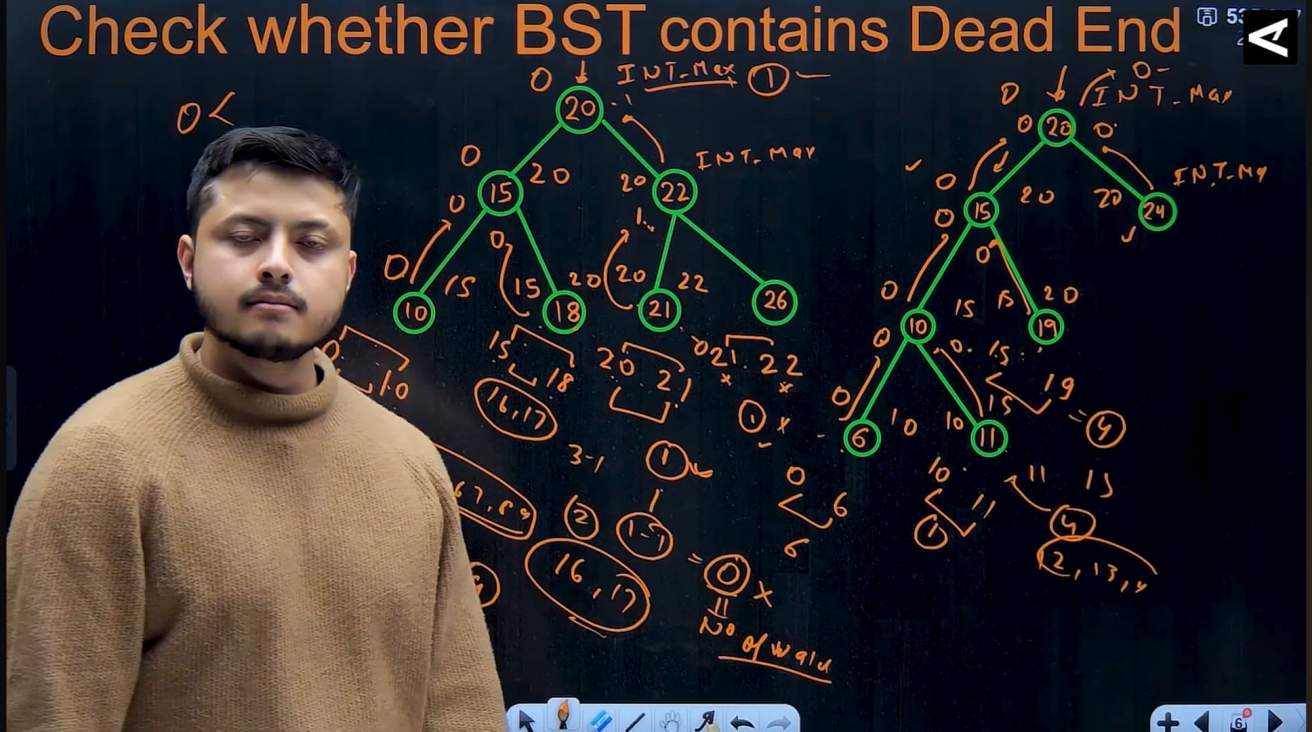
Examples:

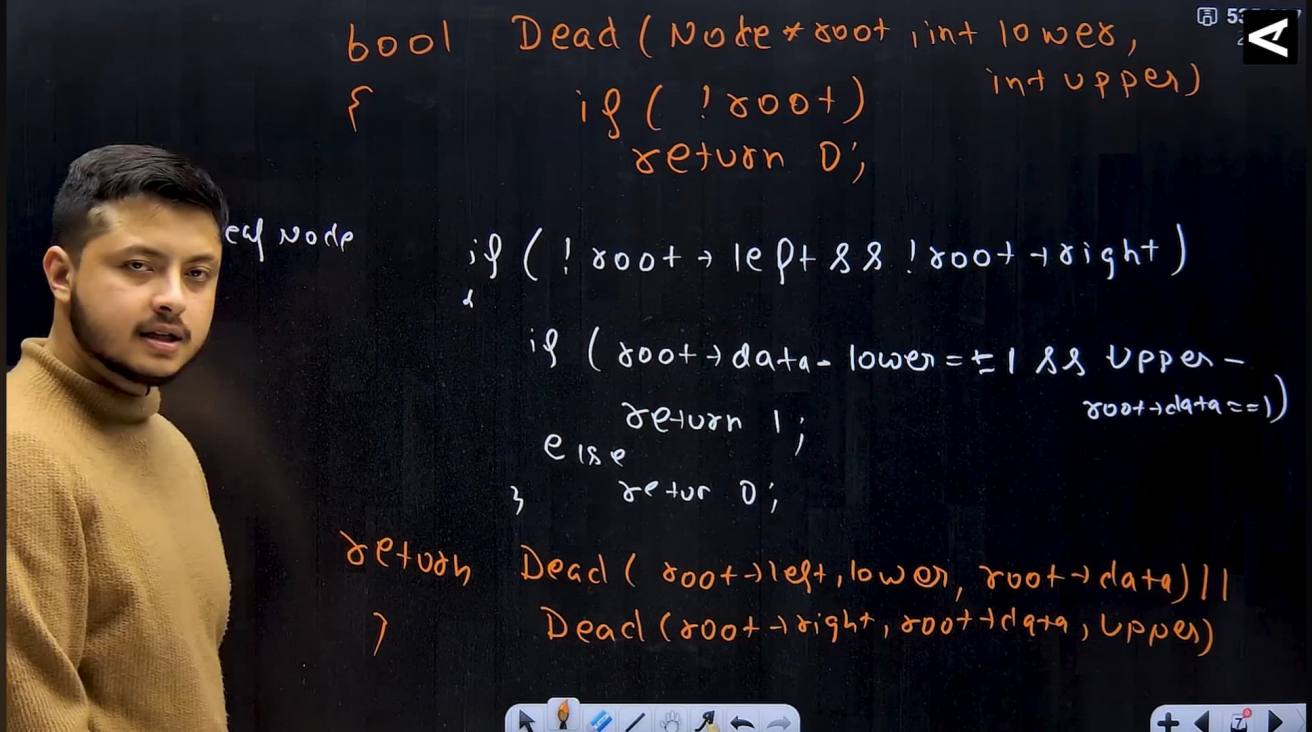
Input: root[] = [8, 5, 9, 2, 7, N, N, 1]



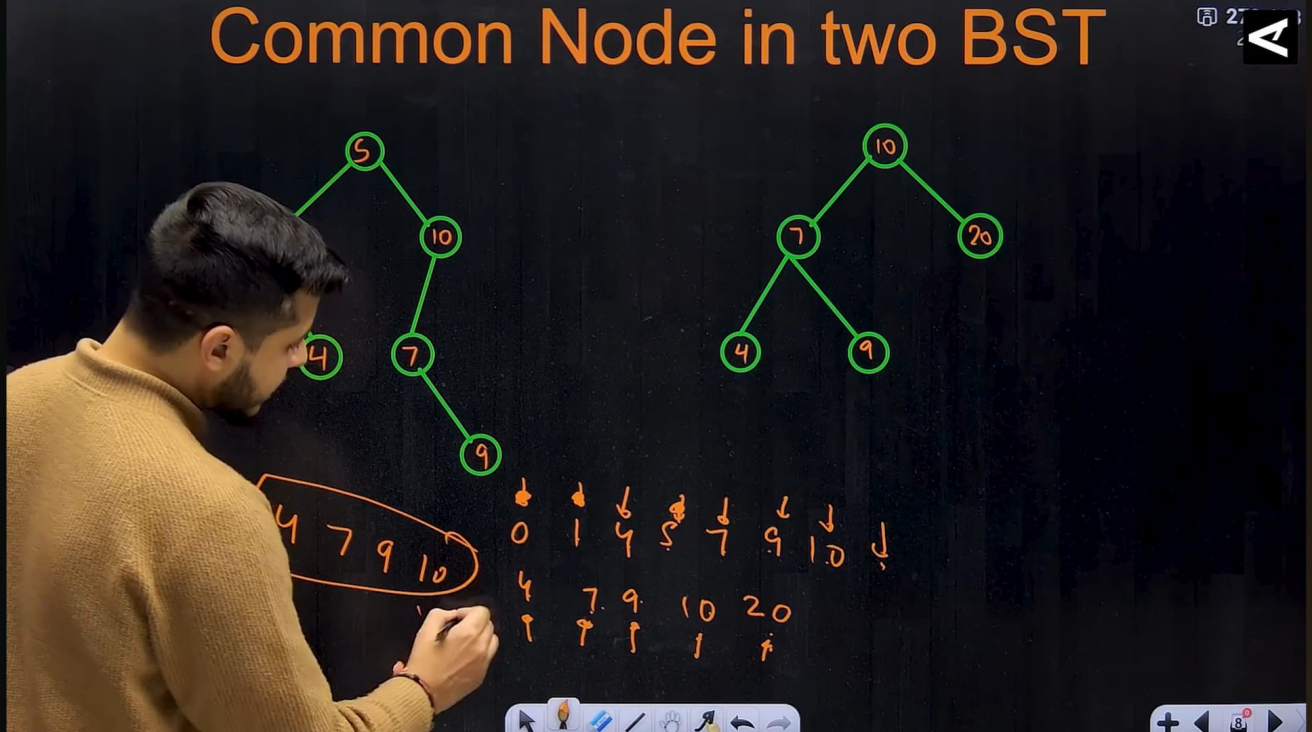
Output: true
Explanation: Node 1 is a Dead End in the given BST.

Check whether BST contains Dead End

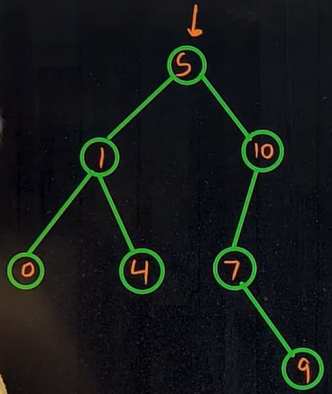


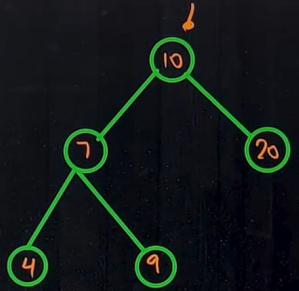

bool Dead(Node *root, int lower,
int upper)
{
if(!root)
return 0;
if(!root->left && !root->right)
if(root->data-lower==1 && upper-
root->data==1)
return 1;
else
return 0;
return Dead(root->left, lower, root->data) ||
Dead(root->right, root->data, upper)
}

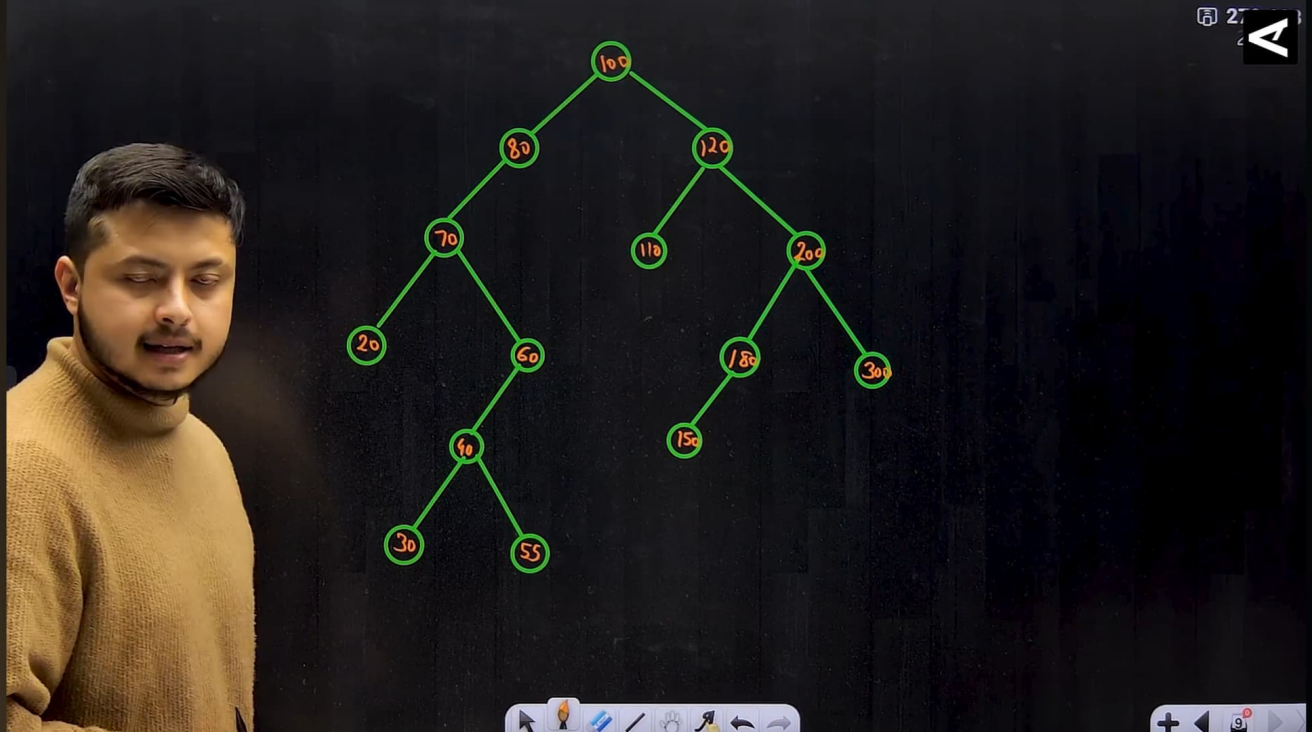
Common Node in two BST

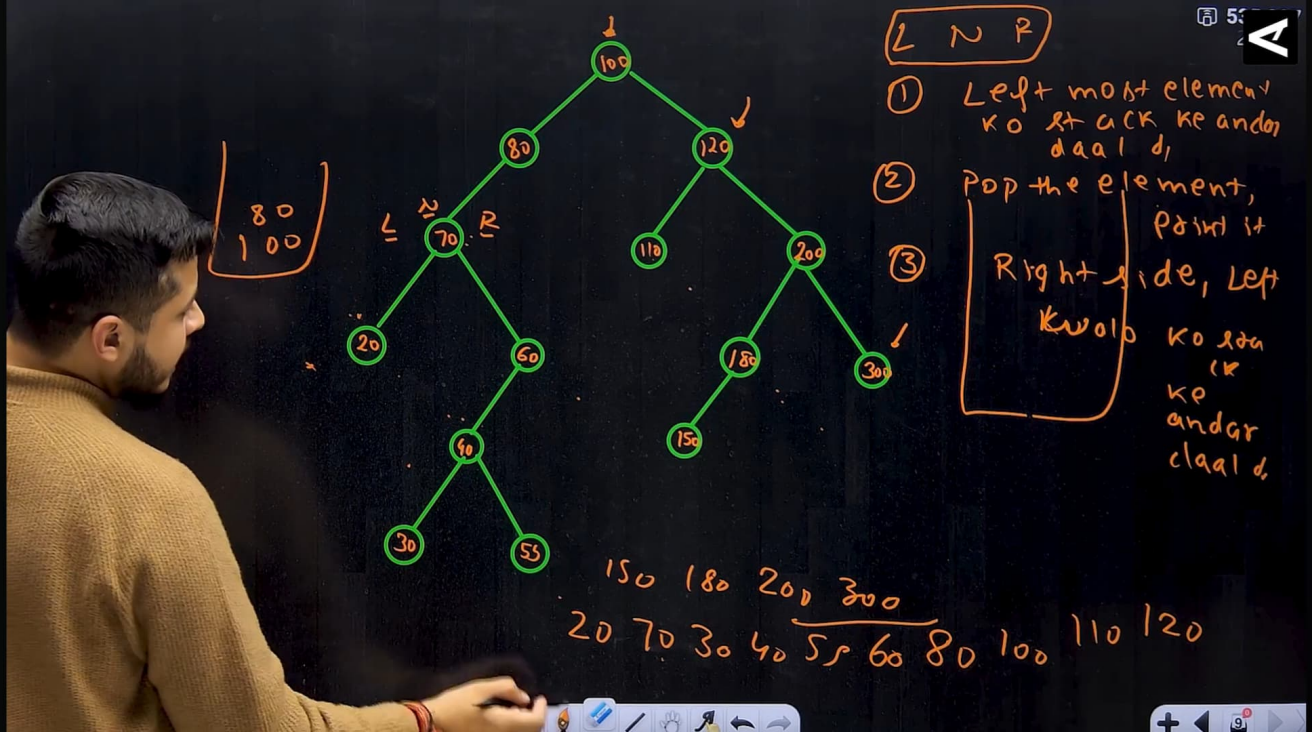


Common Node in two BST

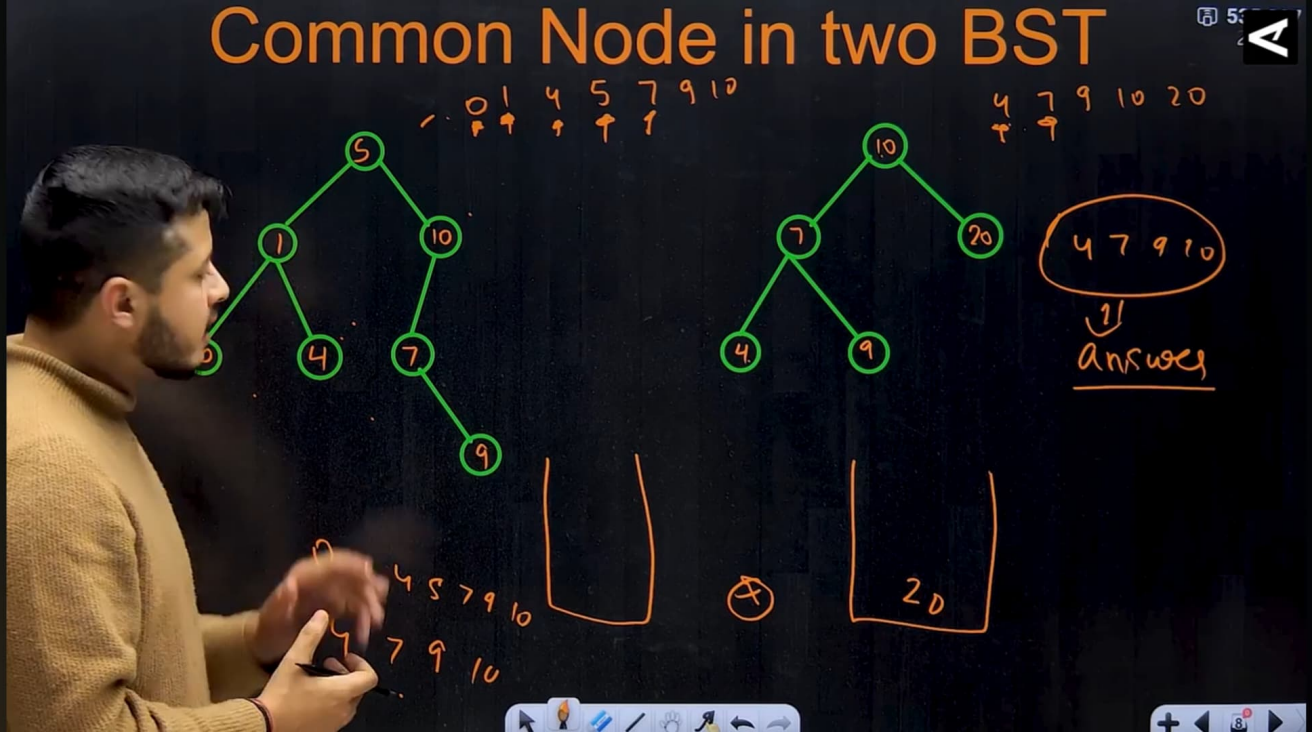





100
80
120
70
110
200
20
60
180
300
40
150
30
55


L N R
1 Left most element ko stack ke andar daal do,
2 Pop the element, print it
3 Right side, Left ko stack ke andar daal do
100
80
120
70
110
200
20
60
180
300
40
150
30
55
80
100
150 180 200 300
20 70 30 40 55 60 80 100 110 120

Common Node in two BST



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:

BST2:

Output: 4 7 9 10

Example 2:


```
93      //Your code here
94        vector<int>ans;
95        stack<Node*>s1,s2;
96        while(root1) // all the left side push into sta
97        {
98            s1.push(root1);
99            root1=root1->left;
100       }
101
102       while(root2) //all the left side push into stac
103       {
104           s2.push(root2);
105           root2=root2->left;
106       }
107
108
109   }
110};
111
112
113
```

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:

BST2:

Output: 4 7 9 10

Example 2:


```
100         }
101
102         while(root2) //all the left side push into stack
103         {
104             s2.push(root2);
105             root2=root2->left;
106         }
107
108         while(!s1.empty()&&!s2.empty())
109         {
110             // top element equal
111             // s1 > s2
112             // s2 > s1
113         }
114
115     }
116 };
117
118
119
120 // } Driver Code Ends
```

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:

BST2:

Output: 4 7 9 10

Example 2:


```
109
110     while(!s1.empty()&&!s2.empty())
111     {
112         // top element equal
113         if(s1.top()->data==s2.top()->data)
114         {
115             ans.push_back(s1.top()->data);
116             root1 = s1.top()->right;
117             s1.pop();
118             root2 = s2.top()->right;
119             s2.pop();
120         }
121         // s1 > s2
122         else if(s1.top()->data>s2.top()->data)
123         {
124             root2 = s2.top()->right;
125             s2.pop();
126         }
127         // s2 > s1
128         else
129         {
130
```

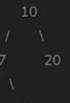
Given two Binary Search Trees. Find the nodes that are common in both of them, ie- find the intersection of the two BSTs.

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

Example 1:

Input:
BST1:

BST2:



Output: 4 7 9 10

Example 2:


```
126            }
127            // s2 > s1
128            else
129            {
130                root1 = s1.top()->right;
131                s1.pop();
132            }
133
134            while(root1) // all the left side push into
135            {
136             s1.push(root1);
137             root1=root1->left;
138            }
139
140            while(root2) //all the left side push into st
141            {
142             s2.push(root2);
143             root2=root2->left;
144            }
145        }
146
```