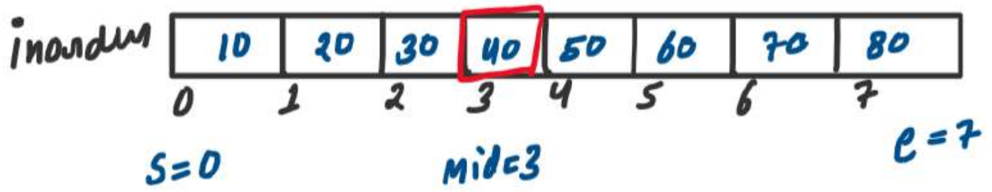
09/12/2023

BINARY SEARCH TREE
CLASS - 2

1. Constructor BST from In-order Traversal (LNR)

EX1



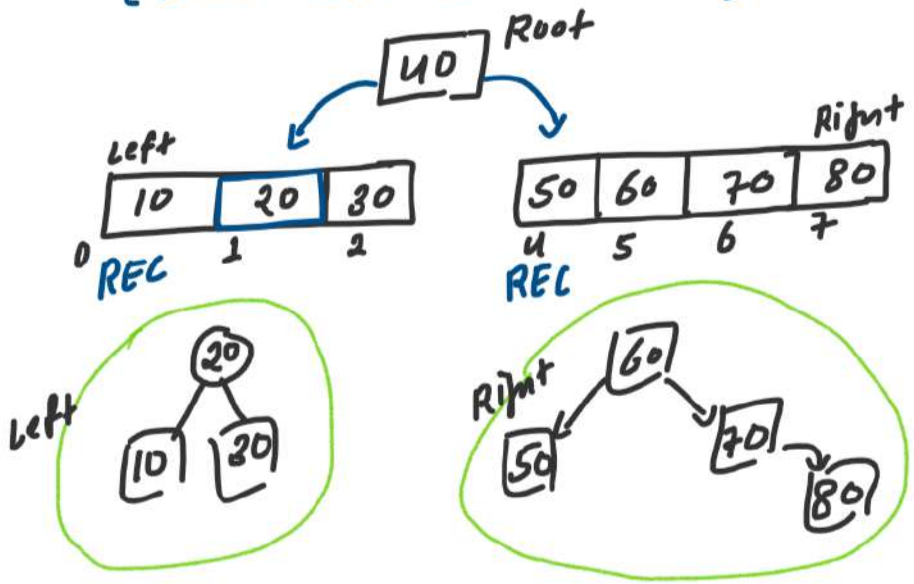
Base case

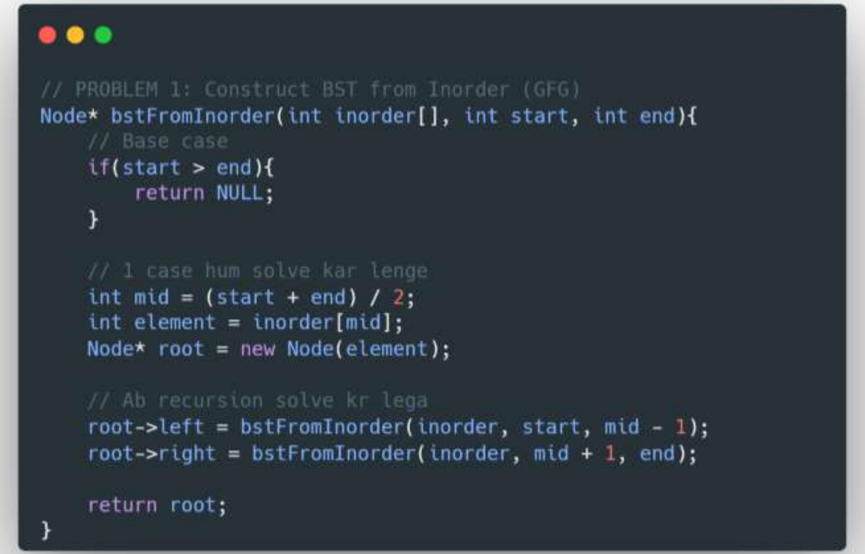
if (S > e)
  return Null

① Find mid

$mid = \frac{s+e}{2}$

② Take mid as a Root Node
(1 case hum solve kar lenge)





```cpp
// PROBLEM 1: Construct BST from Inorder (GFG)
Node* bstFromInorder(int inorder[], int start, int end){
    // Base case
    if(start > end){
        return NULL;
    }

    // 1 case hum solve kar lenge
    int mid = (start + end) / 2;
    int element = inorder[mid];
    Node* root = new Node(element);

    // Ab recursion solve kr lega
    root->left = bstFromInorder(inorder, start, mid - 1);
    root->right = bstFromInorder(inorder, mid + 1, end);

    return root;
}
```

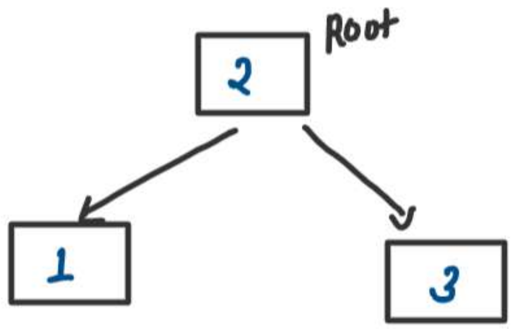
Time complexity: *O(N)*

Space Complexity:
Skewed BST: O(N) in the worst case
Balanced BST: O(log N) in the average case

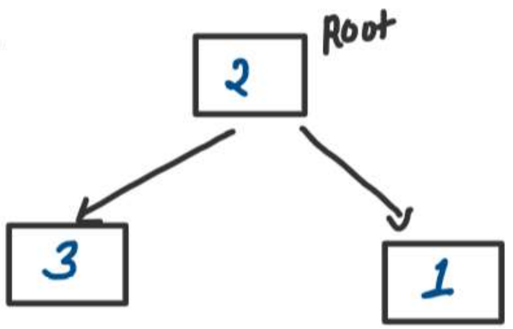
Where N is number of elements

2. Validate BST (Leetcode-98)

Ex 1



Ex 2



Output TRUE

Output False

Root > 1
Root < 3 } A BST

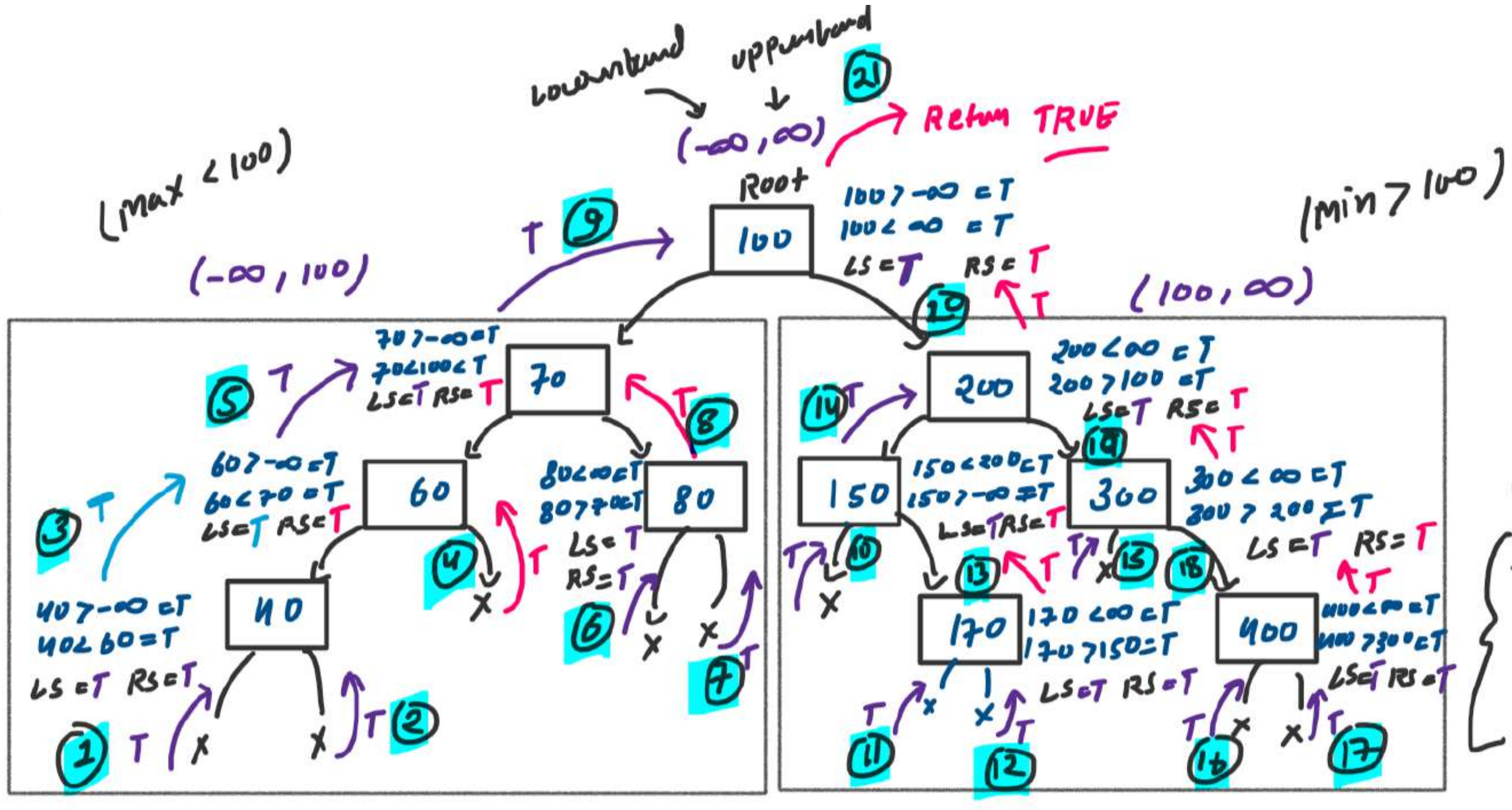
Root < 3
Root > 1 } Not A BST

For Every Root

$\boxed{\text{Left} < \text{Root} < \text{Right}}$

↳ this condition verifies the BST

EX 3

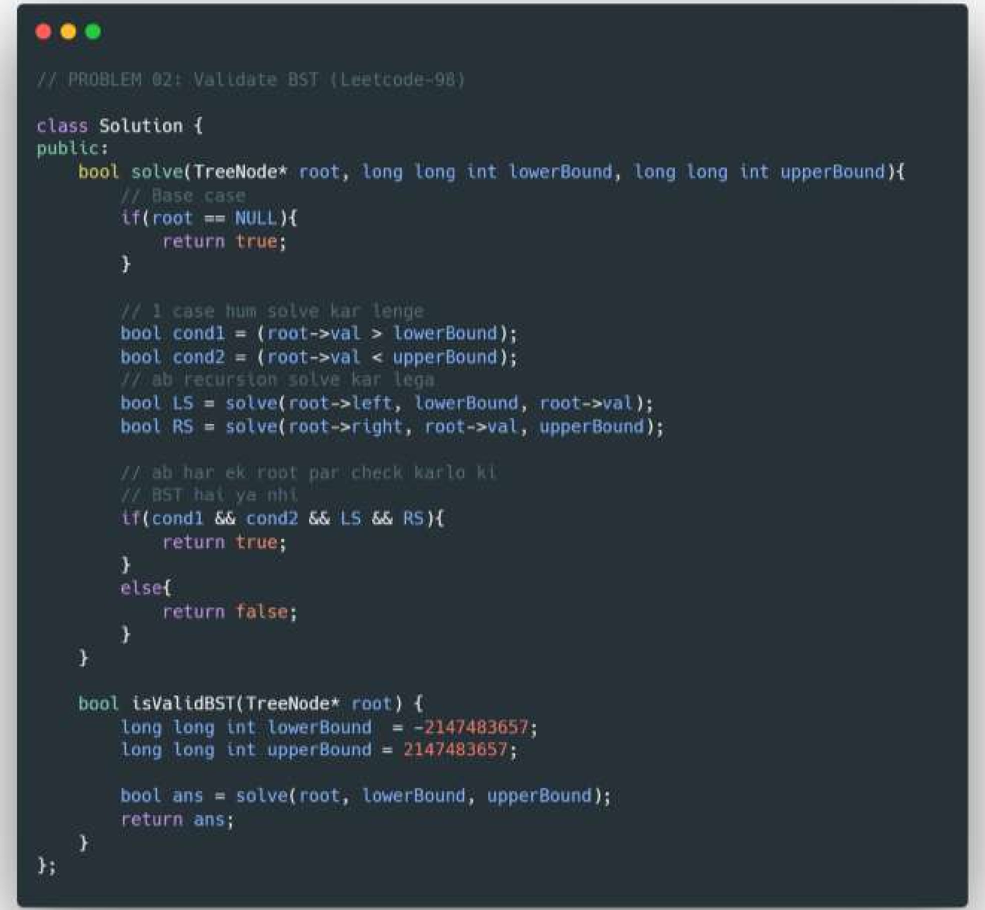


BST TRUE kab Hogi ?

$$\left\{\begin{array}{l} 100 > -\infty \ \&\& \\ 100 < \infty \ \&\& \\ LsubT = T \ \&\& \\ RsubT = T \end{array}\right\}$$

↳ TRUE

In-Order Traversal ⇒ LNR



```cpp
// PROBLEM 02: Validate BST (Leetcode-98)

class Solution {
public:
    bool solve(TreeNode* root, long long int lowerBound, long long int upperBound){
        // Base case
        if(root == NULL){
            return true;
        }

        // 1 case hum solve kar lenge
        bool cond1 = (root->val > lowerBound);
        bool cond2 = (root->val < upperBound);
        // ab recursion solve kar lega
        bool LS = solve(root->left, lowerBound, root->val);
        bool RS = solve(root->right, root->val, upperBound);

        // ab har ek root par check karlo ki
        // BST hai ya nhi
        if(cond1 && cond2 && LS && RS){
            return true;
        }
        else{
            return false;
        }
    }

    bool isValidBST(TreeNode* root) {
        long long int lowerBound  = -2147483657;
        long long int upperBound = 2147483657;

        bool ans = solve(root, lowerBound, upperBound);
        return ans;
    }
};
```

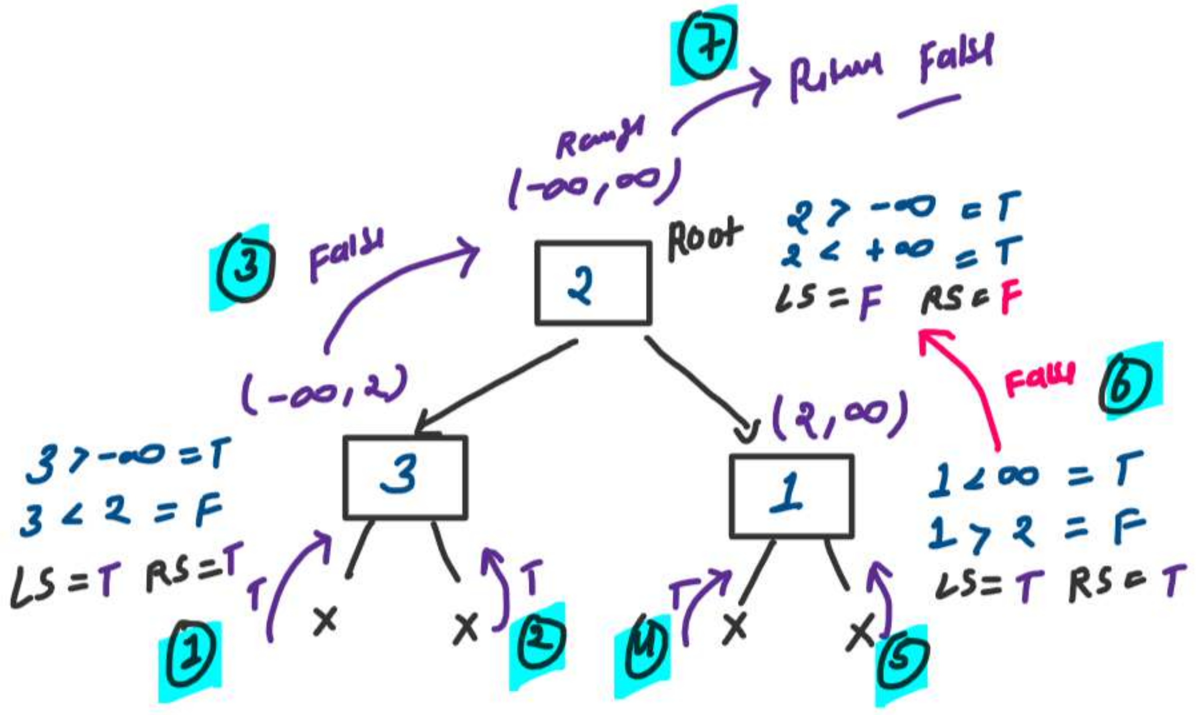
Time Complexity: *O(N)*
Space Complexity: *O(N)*

Where N is number of nodes

DRY RUN

In ORDER

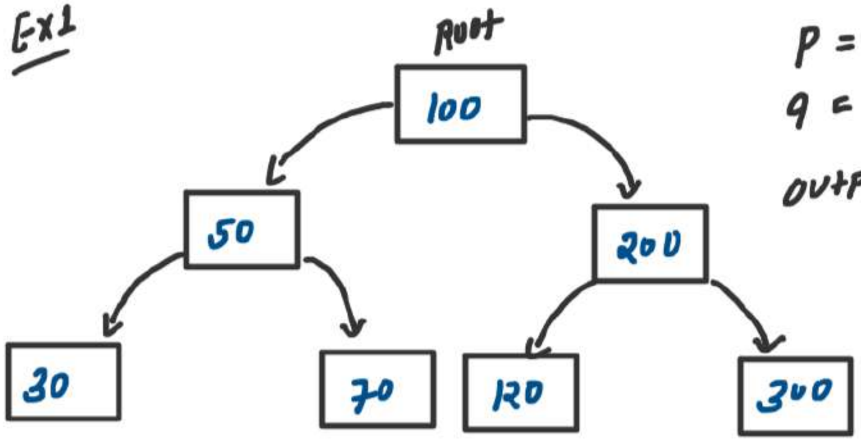
LNR



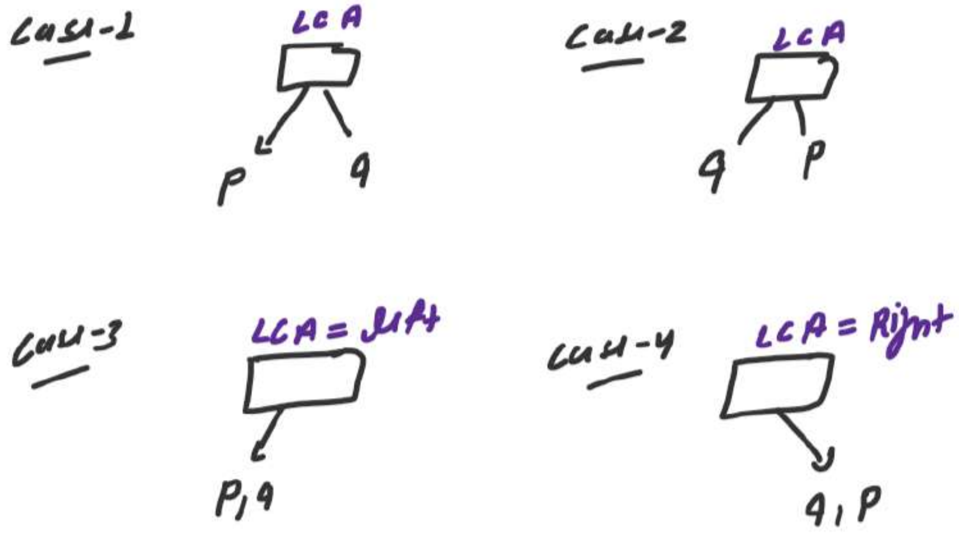
Output = False

3. Lowest Common Ancestor of a BST (Leetcode-235)

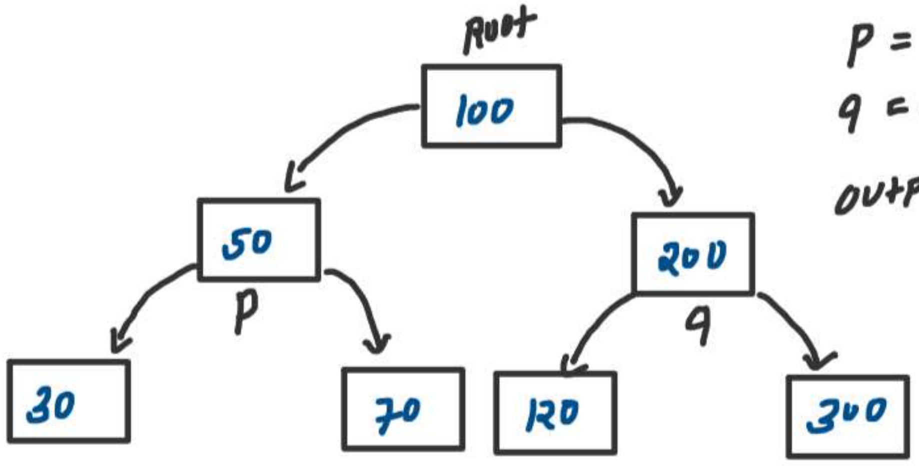
Ex1



p = 30
q = 70
output = 50



Case-1



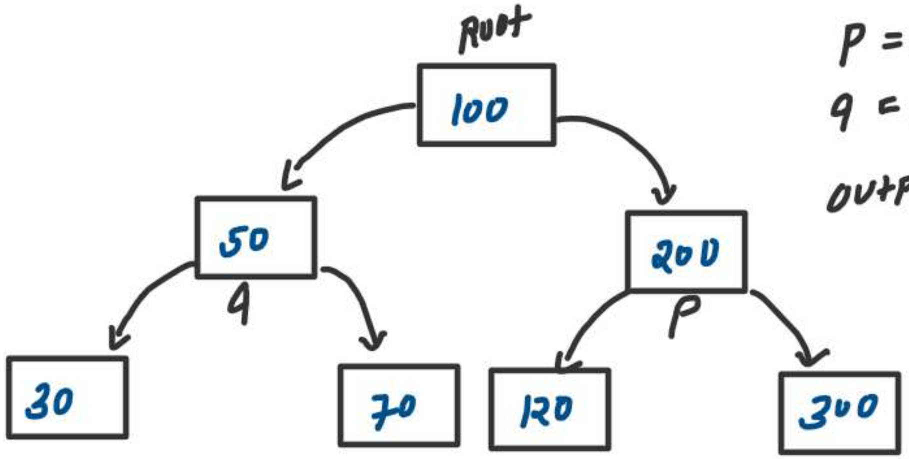
p = 50
q = 200
output = 100

Condition

(p < 100 && q > 100)

return $\boxed{200}$

Case-2



$p = 200$

$q = 50$

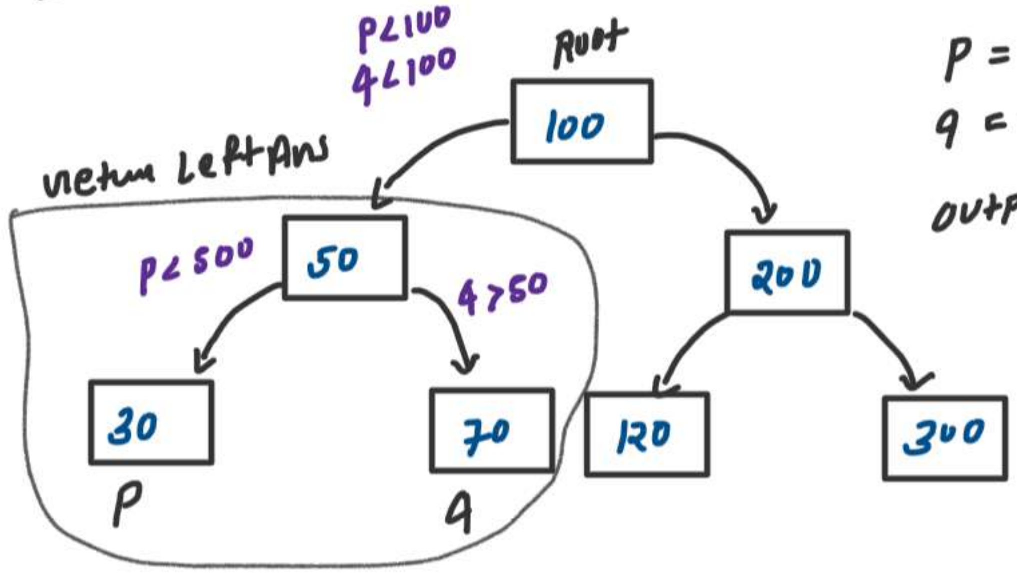
output = 100

condition

$(q < 100 \ \&\& \ p > 100)$

return $\boxed{200}$

Case-3



p = 30

q = 70

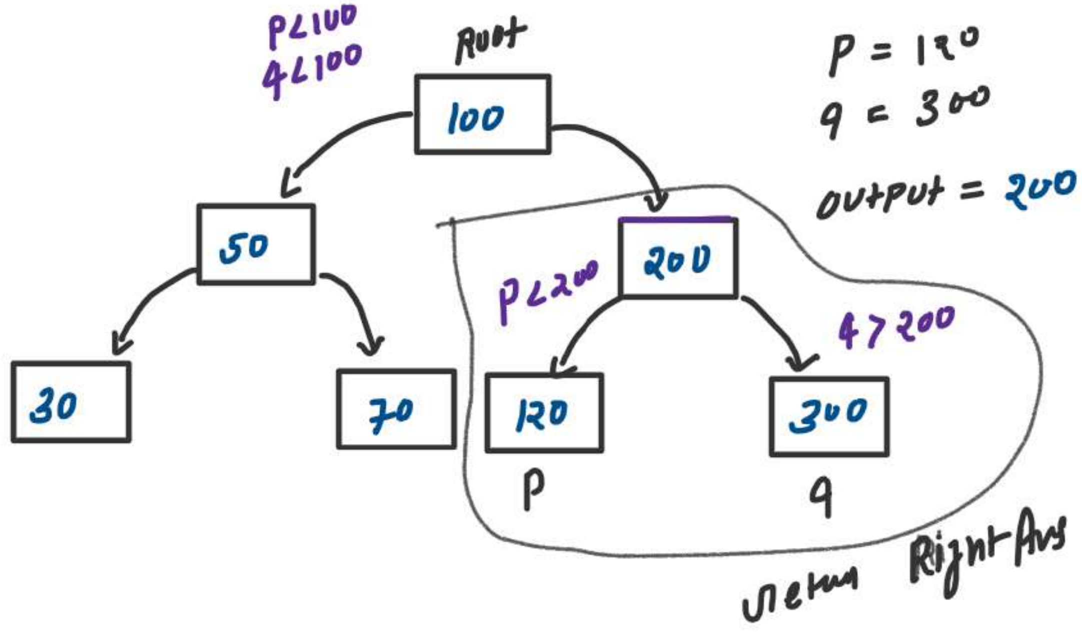
output = 50

Condition

(p < 100 && q < 100)

$\hookrightarrow$ return $\boxed{50}$

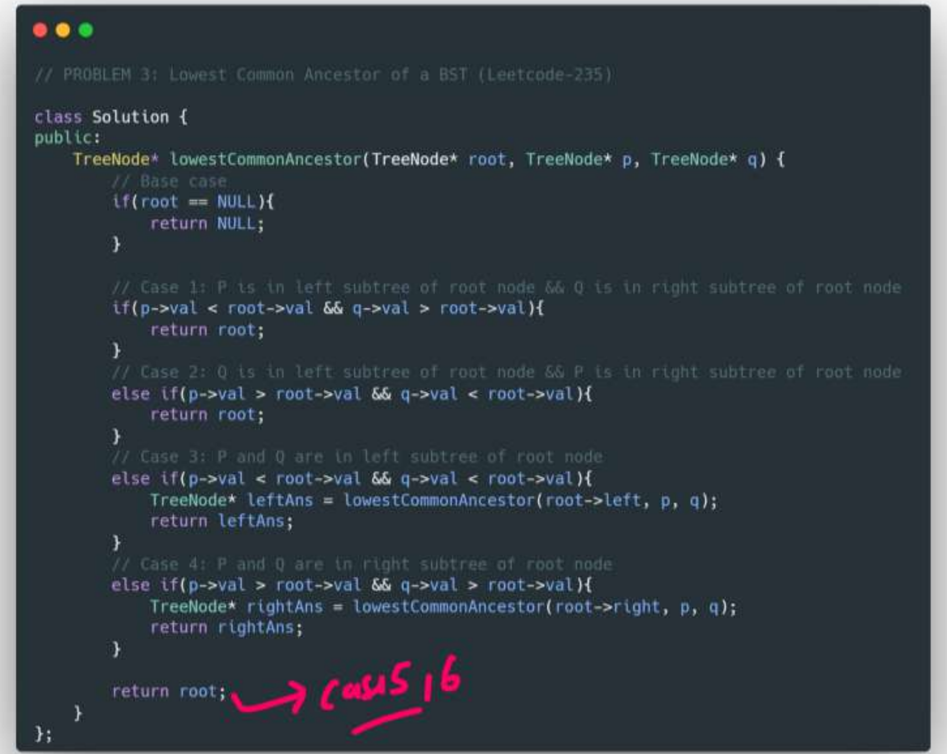
Case - 4



$P = 120$
$q = 300$
output = 200

Condition

$(P > 100 \;\&\&\; q > 100)$
return $\boxed{200}$

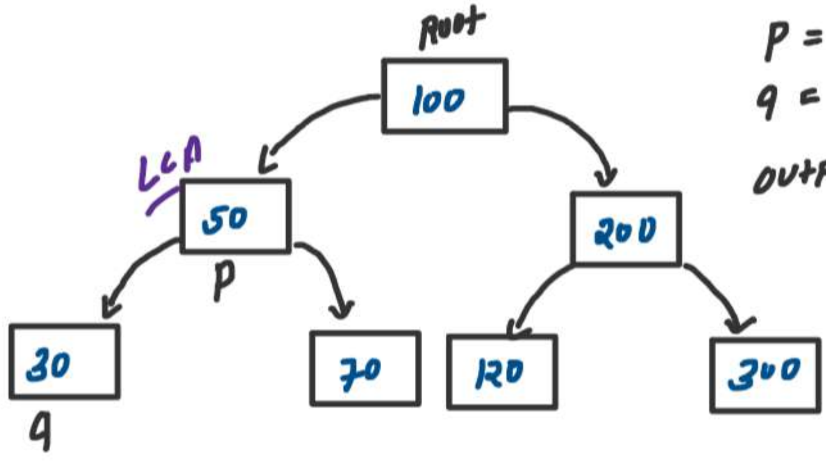


```cpp
// PROBLEM 3: Lowest Common Ancestor of a BST (Leetcode-235)

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        // Base case
        if(root == NULL){
            return NULL;
        }

        // Case 1: P is in left subtree of root node && Q is in right subtree of root node
        if(p->val < root->val && q->val > root->val){
            return root;
        }
        // Case 2: Q is in left subtree of root node && P is in right subtree of root node
        else if(p->val > root->val && q->val < root->val){
            return root;
        }
        // Case 3: P and Q are in left subtree of root node
        else if(p->val < root->val && q->val < root->val){
            TreeNode* leftAns = lowestCommonAncestor(root->left, p, q);
            return leftAns;
        }
        // Case 4: P and Q are in right subtree of root node
        else if(p->val > root->val && q->val > root->val){
            TreeNode* rightAns = lowestCommonAncestor(root->right, p, q);
            return rightAns;
        }

        return root; → case 5 | 6
    }
};
```

Time Complexity: *O(H)*
Space Complexity: *O(H)*

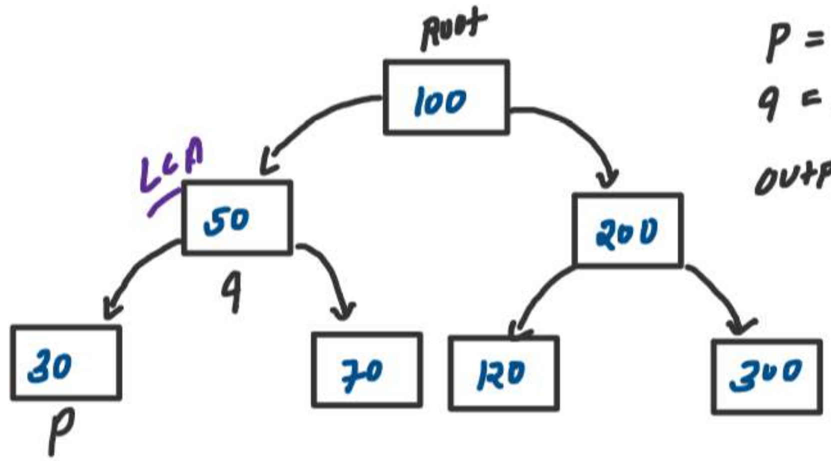
Where H is height of BST

Case-5



P = 50
q = 30
output = P

Case-6

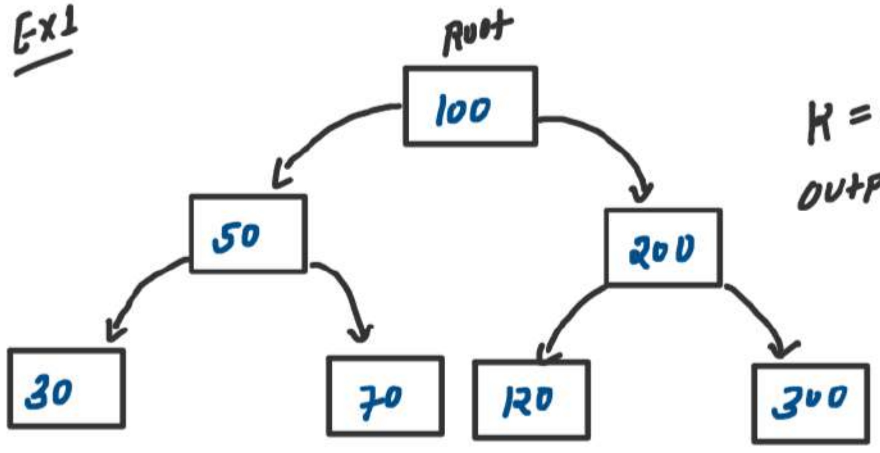


P = 30
q = 50
output = q

Jab case-1, 2, 3, 4 ⇒ TRUE Nahi Hongi to Hum direct root ko Return karte Hai ⇒ To usi time pari case 5 or 6 ke according Ans mil jata Hai.

4. Kth Smallest Element in a BST (Leetcode-230)

Ex1



K = 3

output = 70

BRUTE FORCE APPROACH

[1] STORE All Nodes in the Array using in-ORder Traversal

30	50	70	100	120	200	300
0	1	2	3	4	5	6

[2] get Kth Element

$\boxed{K=3}$ Ans = inOrder [K-1];

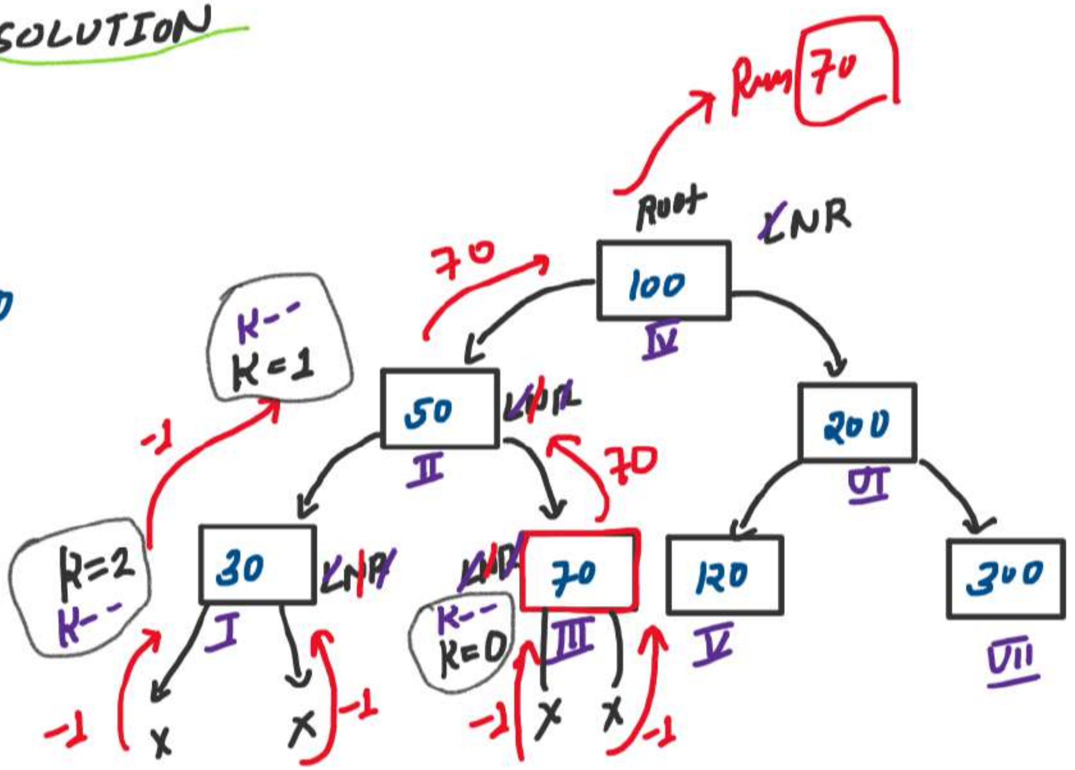
= 70

OPTIMAL SOLUTION

Ex1

K = 3

output = 70



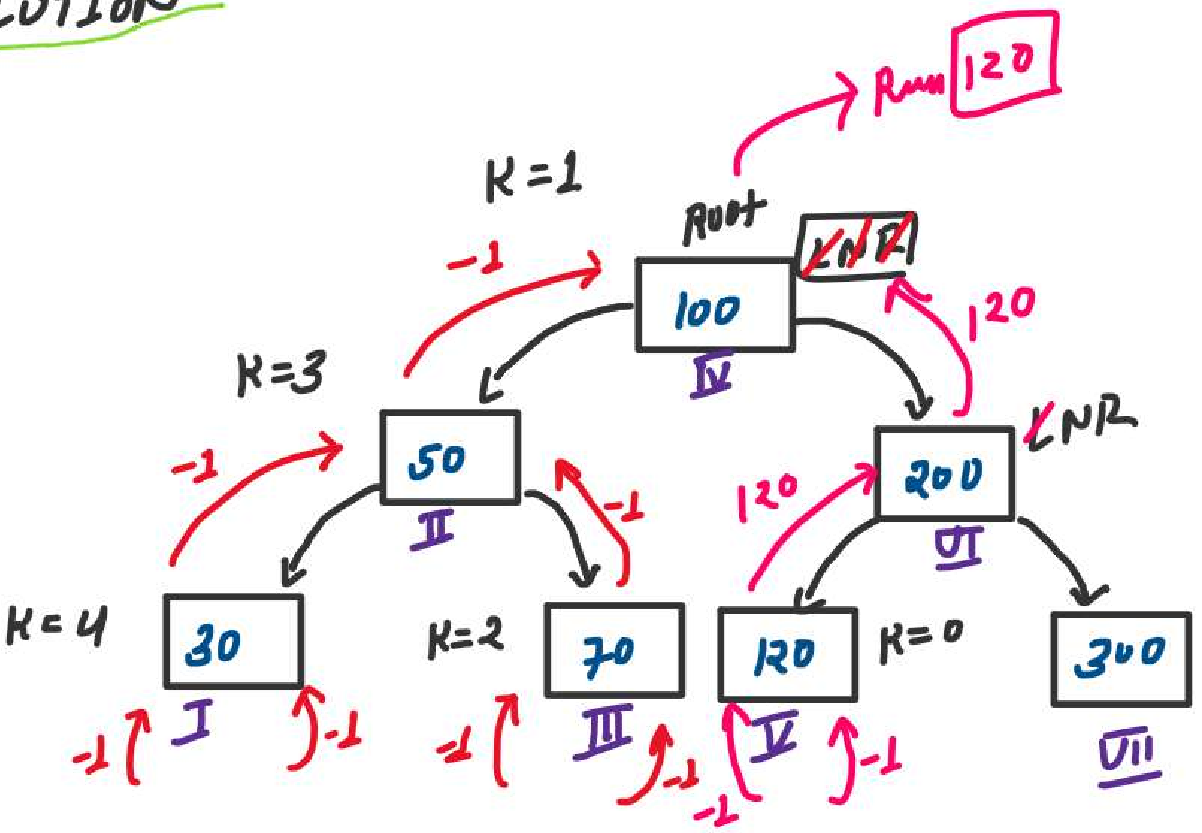
In-Order Traversal (LNR)
↳ Numbering the nodes using this traversal and get the Ans.

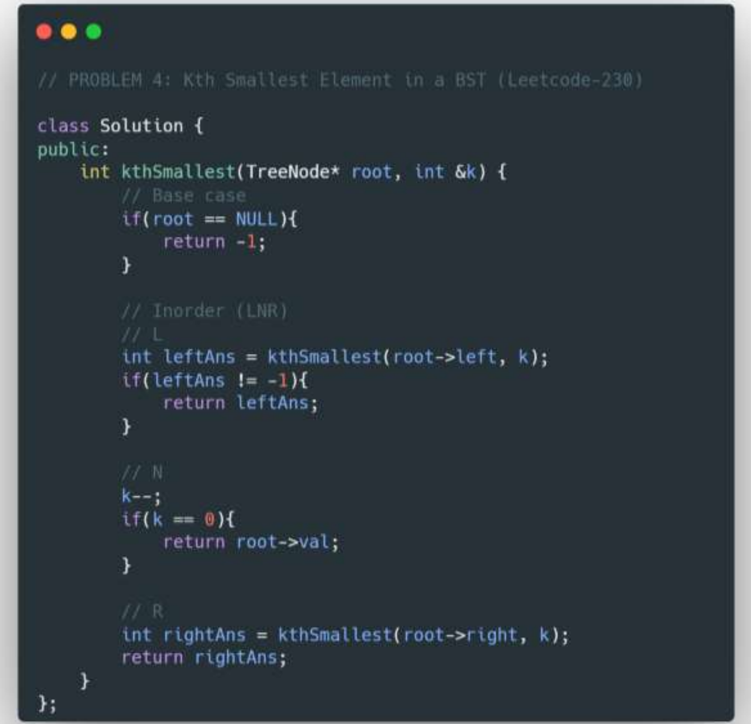
OPTIMAL SOLUTION

Ex2

K = 5

output = R0





```cpp
// PROBLEM 4: Kth Smallest Element in a BST (Leetcode-230)

class Solution {
public:
    int kthSmallest(TreeNode* root, int &k) {
        // Base case
        if(root == NULL){
            return -1;
        }

        // Inorder (LNR)
        // L
        int leftAns = kthSmallest(root->left, k);
        if(leftAns != -1){
            return leftAns;
        }

        // N
        k--;
        if(k == 0){
            return root->val;
        }

        // R
        int rightAns = kthSmallest(root->right, k);
        return rightAns;
    }
};
```

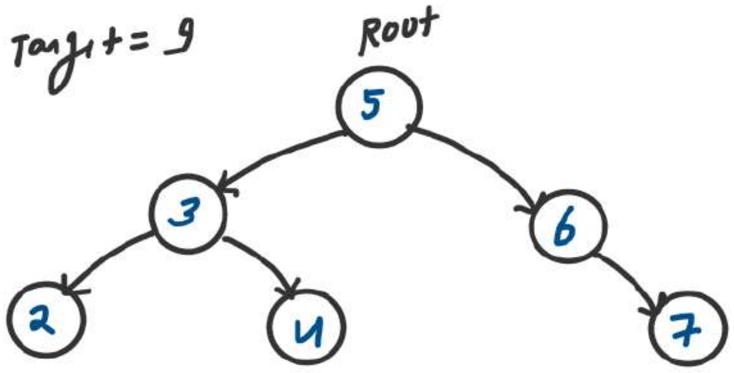
Time Complexity: *O(N)*
Space Complexity: *O(H)*

Where N and H are number of nodes
and Height of BST respectively

5. Two Sum in a BST (Leetcode-653)

EX1

Target = 9



output = True

STEP1

inorder

2	3	4	5	6	7
0	1	2	3	4	5

Pairs

(2,3) (3,4) $\boxed{(4,5)}$ 9 (5,6) (6,7)

(2,4) (3,5) (4,6) (5,7)

(2,5) $\boxed{(3,6)}$ 9 (4,7)

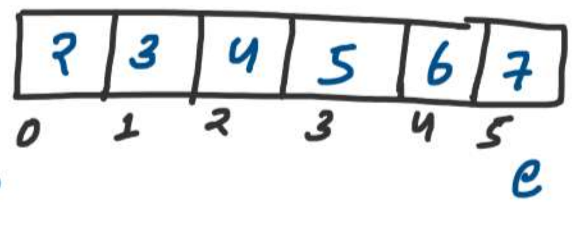
(2,6) (3,7)

$\boxed{(2,7)}$ → 9

Ans TRUE

Optimal solution

STEP1 inorder



$\boxed{T.C. = O(N)}$ where N is number of nodes

STEP2

(inorder[s] + inorder[e] < Target)

↳ s++

(inorder[s] + inorder[e] > Target)

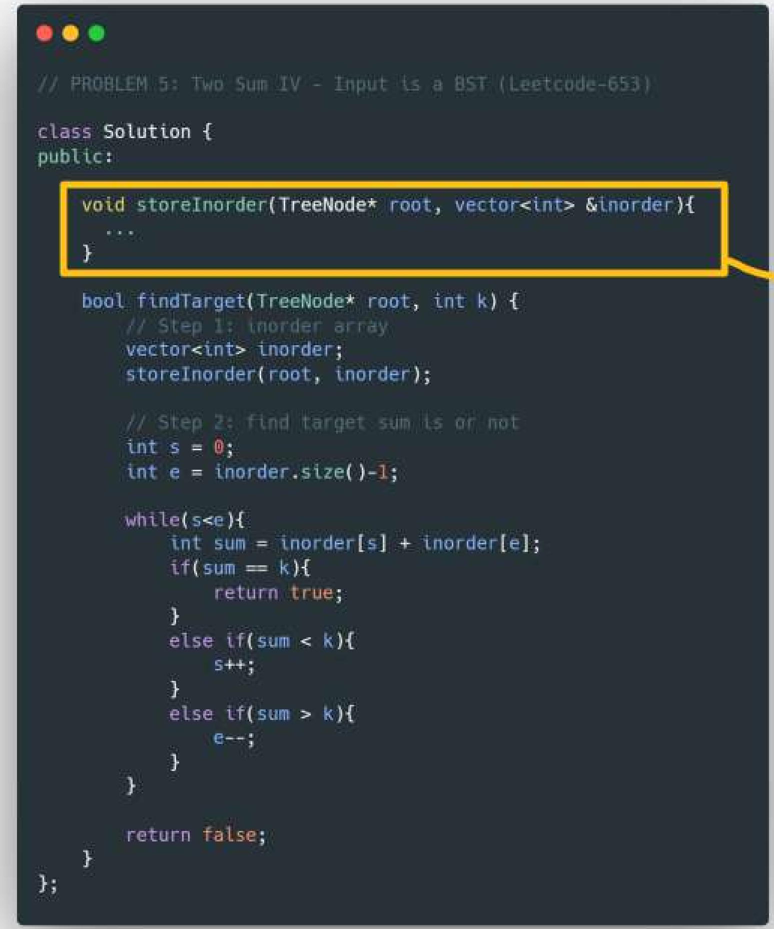
↳ e--

(inorder[s] + inorder[e] == Target

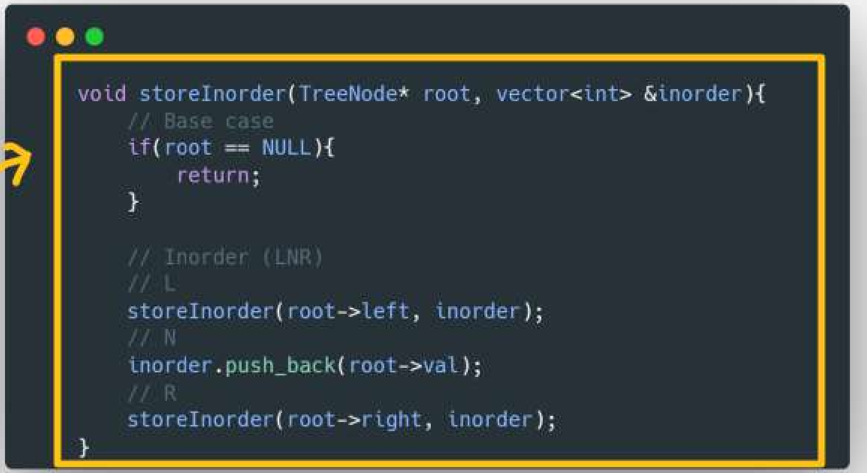
return True;

2+7 = 9

↳ True



```cpp
// PROBLEM 5: Two Sum IV - Input is a BST (Leetcode-653)

class Solution {
public:

    void storeInorder(TreeNode* root, vector<int> &inorder){
      ...
    }

    bool findTarget(TreeNode* root, int k) {
        // Step 1: inorder array
        vector<int> inorder;
        storeInorder(root, inorder);

        // Step 2: find target sum is or not
        int s = 0;
        int e = inorder.size()-1;

        while(s<e){
            int sum = inorder[s] + inorder[e];
            if(sum == k){
                return true;
            }
            else if(sum < k){
                s++;
            }
            else if(sum > k){
                e--;
            }
        }

        return false;
    }
};
```



```cpp
void storeInorder(TreeNode* root, vector<int> &inorder){
    // Base case
    if(root == NULL){
        return;
    }

    // Inorder (LNR)
    // L
    storeInorder(root->left, inorder);
    // N
    inorder.push_back(root->val);
    // R
    storeInorder(root->right, inorder);
}
```

Time Complexity: *O(N)*
Space Complexity: *O(N)*

Where N is number of nodes of BST