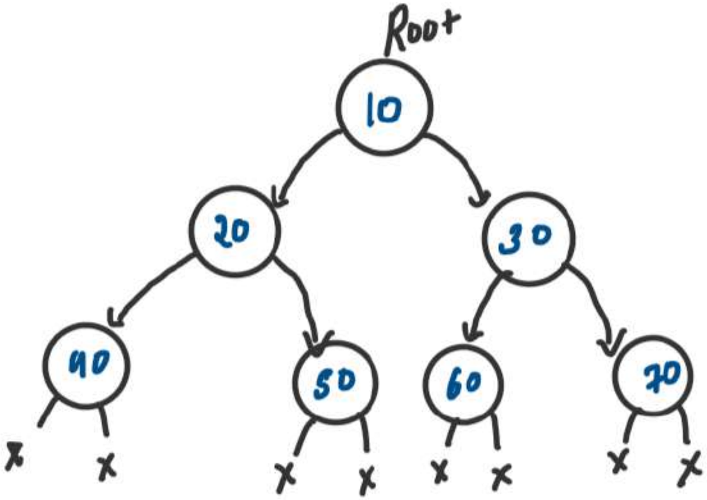
03/12/2023

BINARY TREE
CLASS - 2

1. Balanced Binary Tree (Leetcode-110)

Input



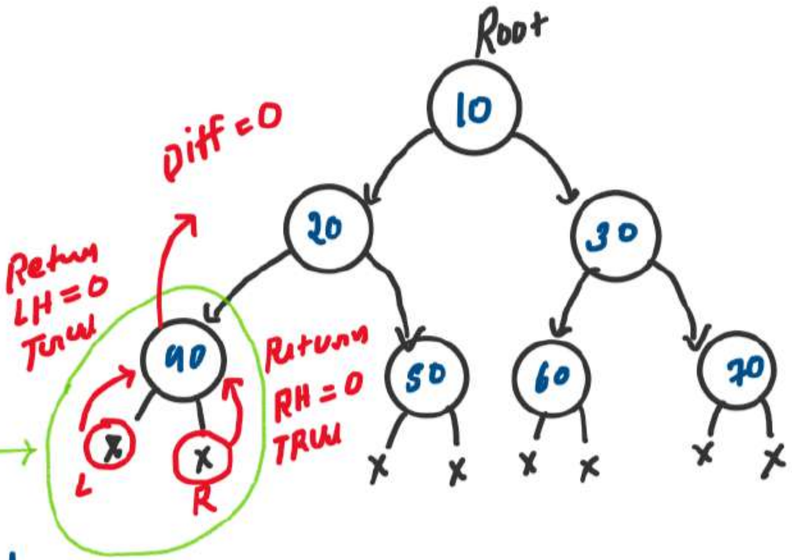
Output TRUE

What is Balanced Binary Tree?

Left and Right sub Tree ki height ka diff.ⁿ at most 1 Hona chahiye.

$$\boxed{L_H - R_H \in [0,1]}$$

Logic Boilding



Leaf Node

Leaf Node ke Right and Left Node ko Balanced manlo

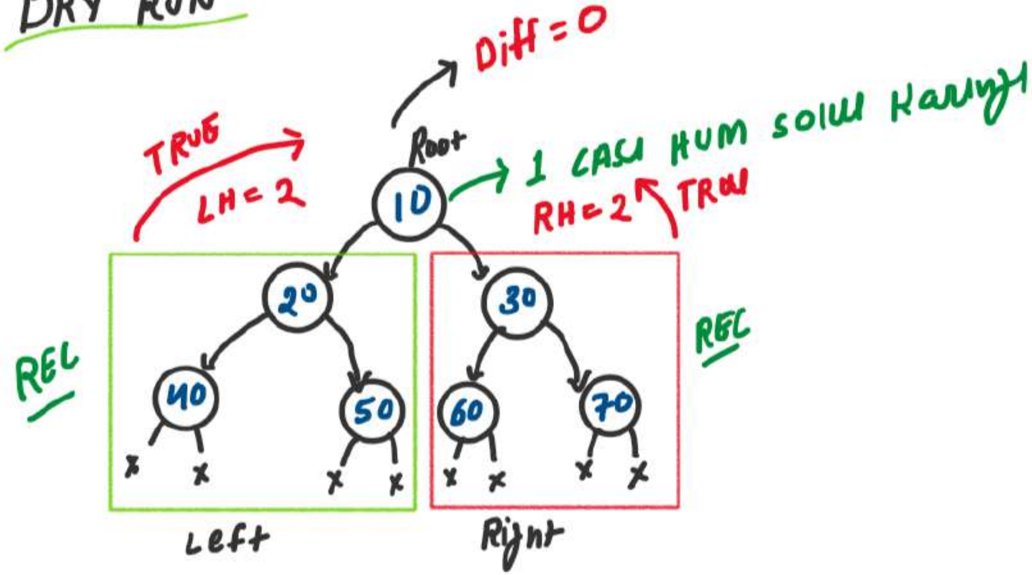
Balanced Tree kab Hogi

- Left Sub Height (LH) = Balanced
- Right sub Height (RH) = Balanced
- Diff ⇒ abs(LH - RH) <= 1

B.C.

```
if( Root == NULL)
{
    // Balance Tree maanlo leaf Node ko
       return True
}
```

DRY RUN



Base case

```
if ( Root = = Null )
    return True
```

1 case hum solve karenge

LH = 2

RH = 2

Diff = 0 ⇒ LH - RH = 2-2 = 0

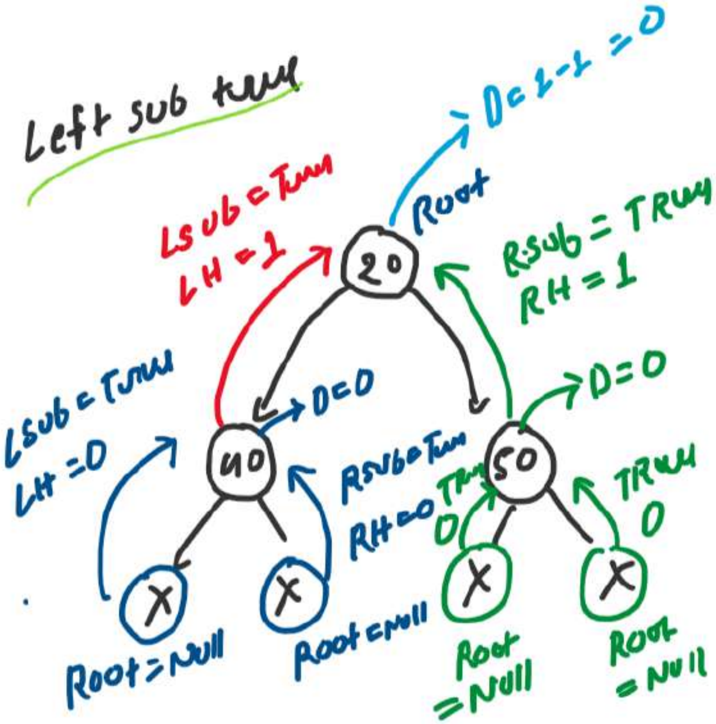
REC

Lsub = True

Rsub = True

```
if ( Diff <= 1 && Lsub &&
        Rsub )
{
    Return True;
}
Else {
    Return False;
}
```

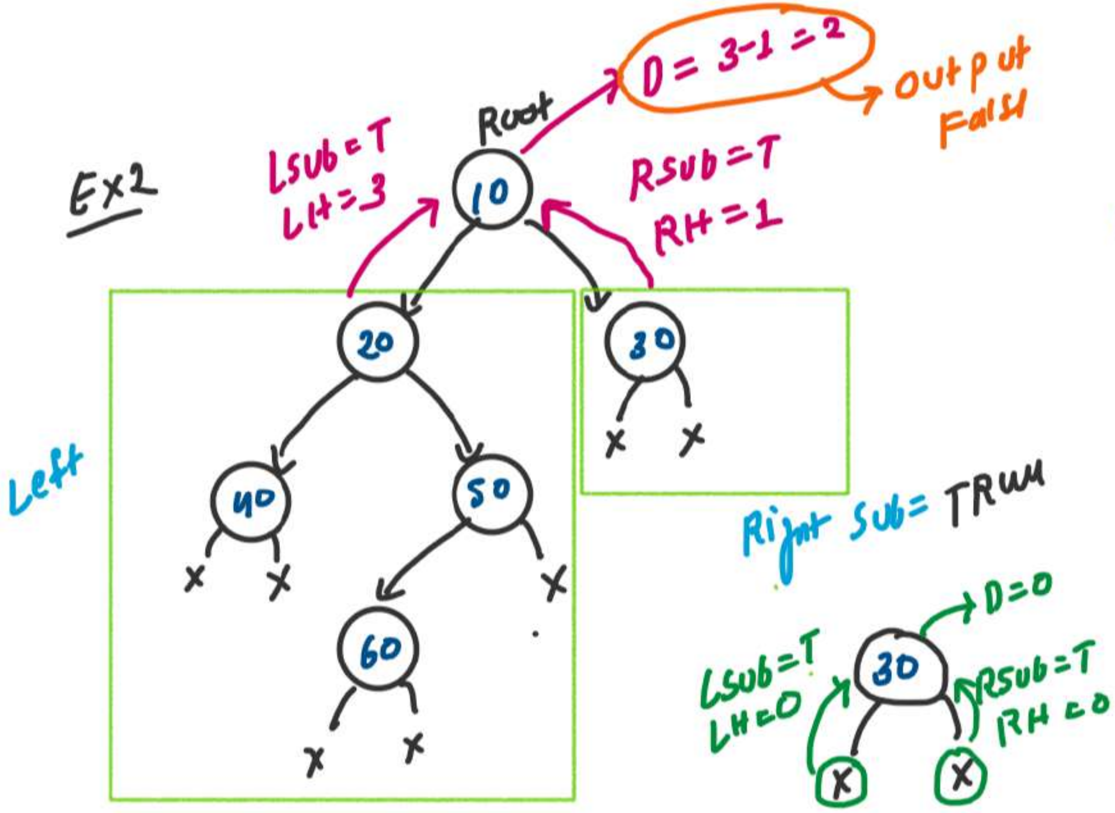
Left sub tree



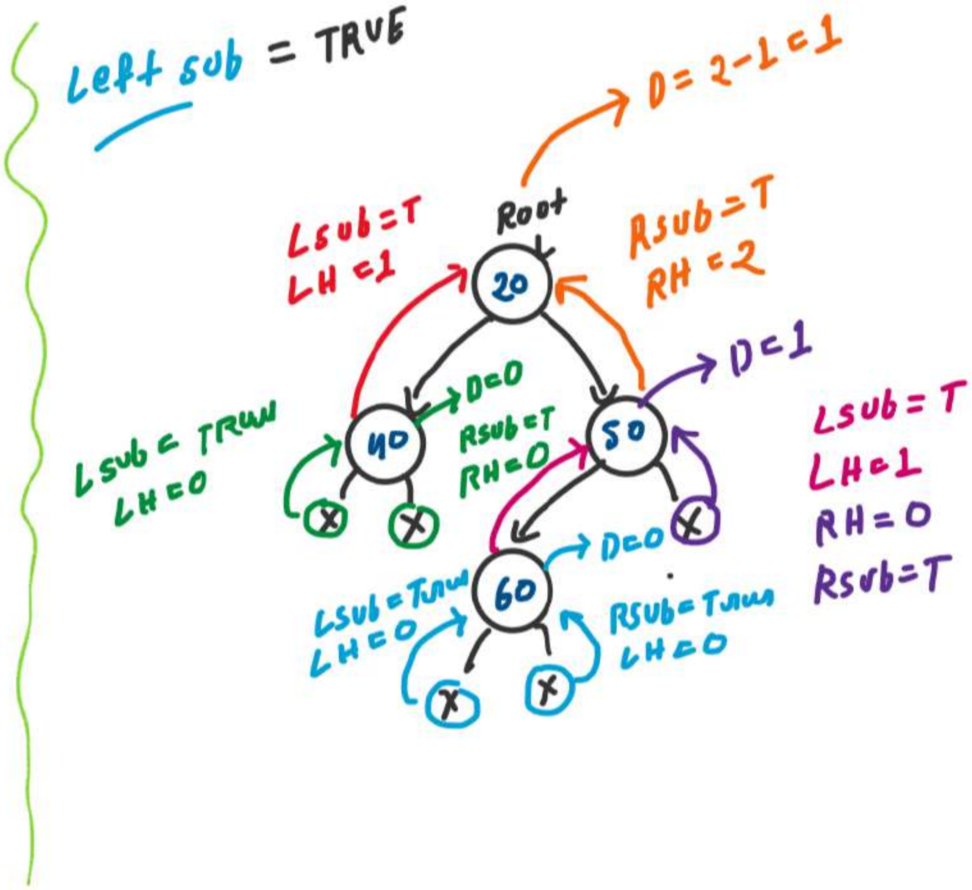
Left sub tree Balanced Hai

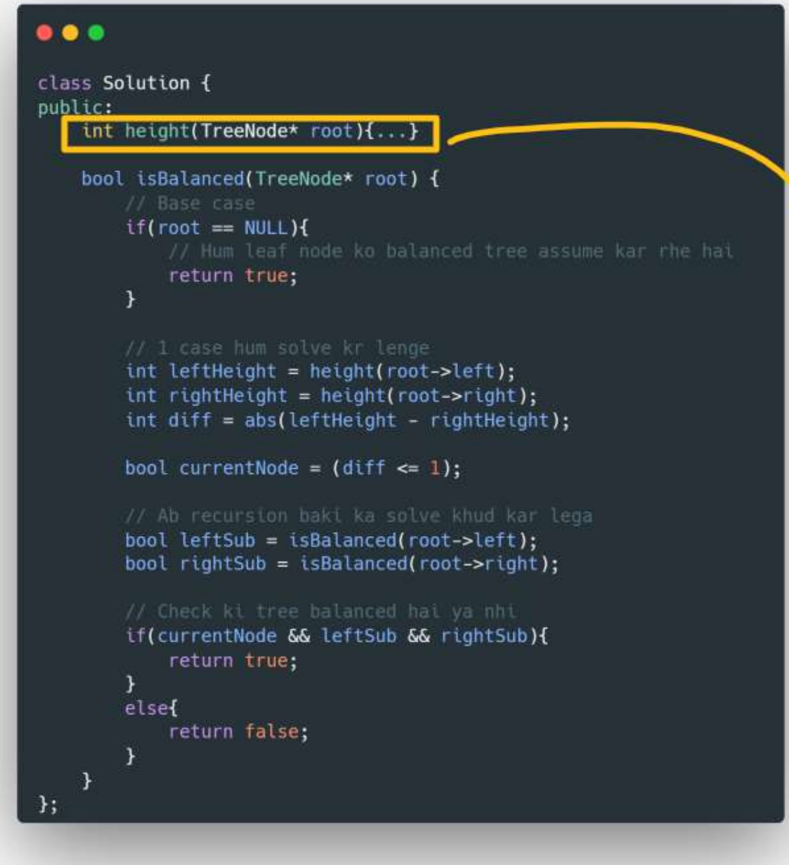
↳ D<=1 and LSub = True and RSub = True

↳ Issi way me Right sub Tree check Hoga
Hi wo Balanced Hai ya Nahi
D<=1 and LSub = True and RSub = True

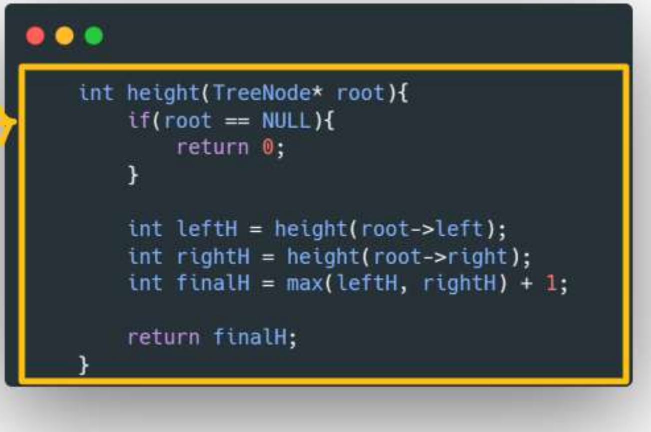


Output = False





```
class Solution {
public:
    int height(TreeNode* root){...}

    bool isBalanced(TreeNode* root) {
        // Base case
        if(root == NULL){
            // Hum leaf node ko balanced tree assume kar rhe hai
            return true;
        }

        // 1 case hum solve kr lenge
        int leftHeight = height(root->left);
        int rightHeight = height(root->right);
        int diff = abs(leftHeight - rightHeight);

        bool currentNode = (diff <= 1);

        // Ab recursion baki ka solve khud kar lega
        bool leftSub = isBalanced(root->left);
        bool rightSub = isBalanced(root->right);

        // Check ki tree balanced hai ya nhi
        if(currentNode && leftSub && rightSub){
            return true;
        }
        else{
            return false;
        }
    }
};
```

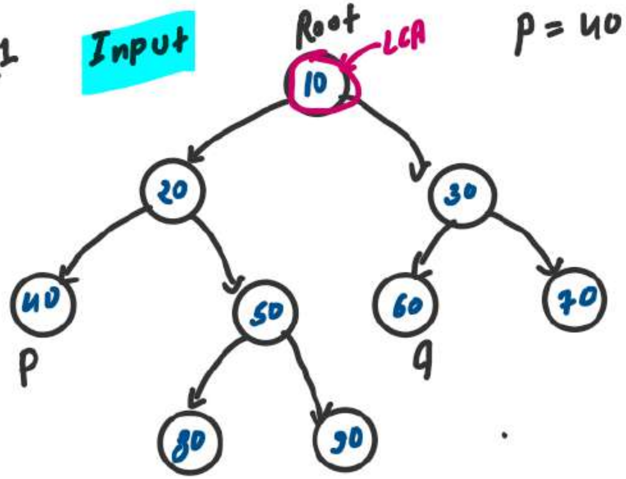


```
    int height(TreeNode* root){
        if(root == NULL){
            return 0;
        }

        int leftH = height(root->left);
        int rightH = height(root->right);
        int finalH = max(leftH, rightH) + 1;

        return finalH;
    }
```

T.C. =? and S.C. =?

2. Lowest Common Ancestor of a Binary Tree (Leetcode-236) ** ****

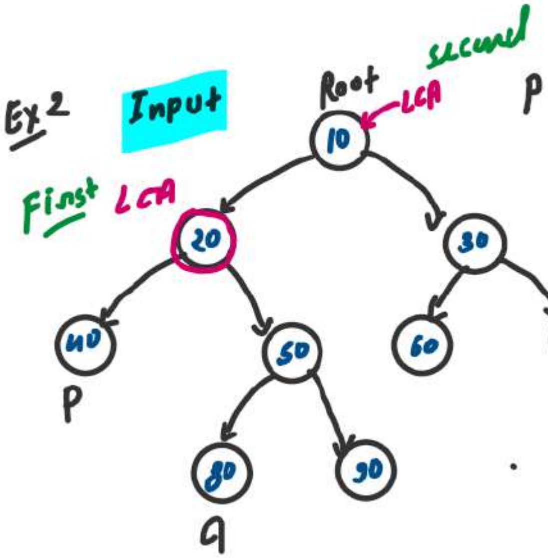
Ex 1 **Input**



p = 40 q = 60

Output 10

Ex 2 **Input**

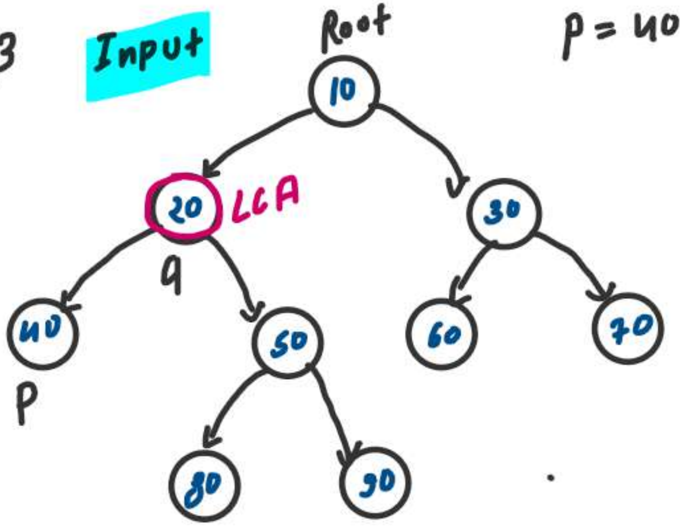


p = 40 q = 80

Output 20

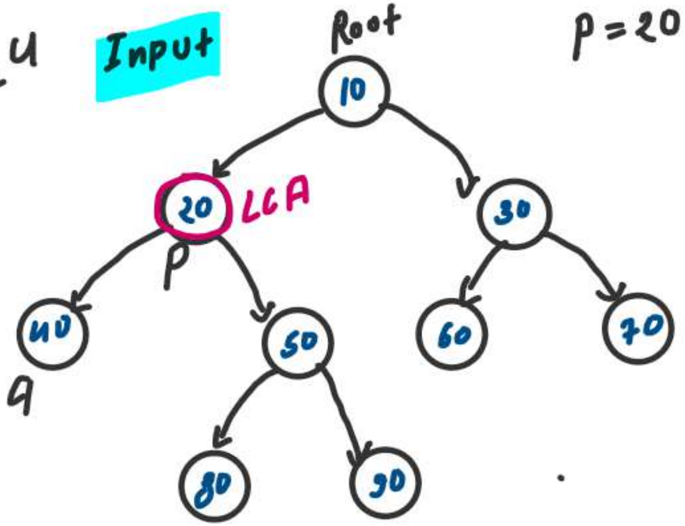
Always consider First LCA as output

Ex 3 **Input** p = 40 q = 20



Output 20

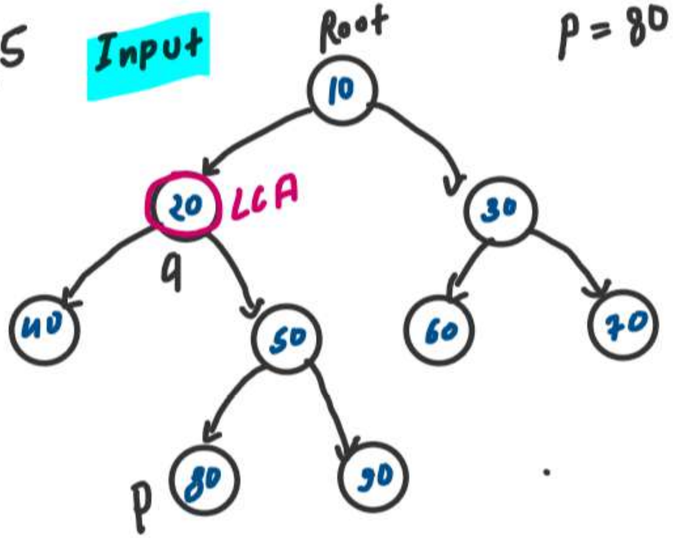
Ex 4 **Input** p = 20 q = 40



Output 20

Ex 5 **Input**

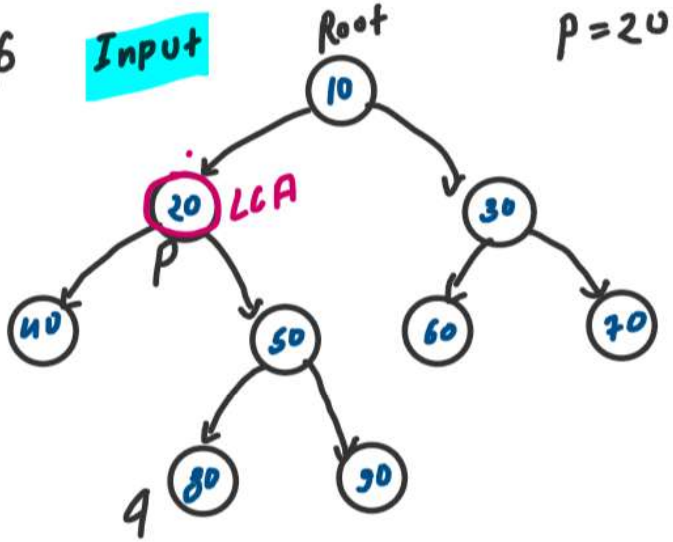
p = 80 q = 20



Output 20

Ex 6 **Input**

p = 20 q = 80



Output 20

DRY RUN

EX1

$p = 40$

$q = 60$

output = 10

⑪ $30 \neq p$ { $30 \neq q$

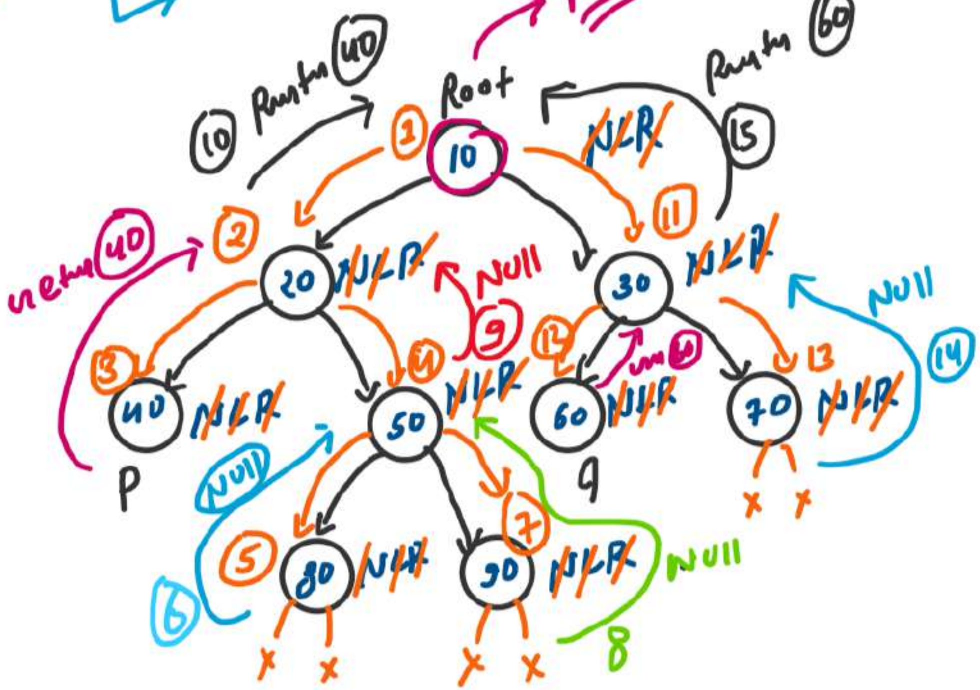
⑫ $60 = q$ return q

⑬ $70 \neq q$ { $70 \neq p$

⑭ root == Null
return Null

⑮ 60 and Null
return (60)

⑯ p == 40 and q == 60
↳ return Root = 10



⑩ 40 and NULL
return (40)

⑨ Null and Null
return Null

① First MUJHE p and q ki location find karni hogi using Preorder Traversal

① $10 \neq p$ { $10 \neq q$

② $20 \neq p$ { $20 \neq q$

③ $40 = p$ return p

④ $50 \neq p$ { $50 \neq q$

⑤ $80 \neq p$ { $80 \neq q$

⑥ root == Null
return Null

⑦ $90 \neq p$ { $90 \neq q$

⑧ root == Null
return Null

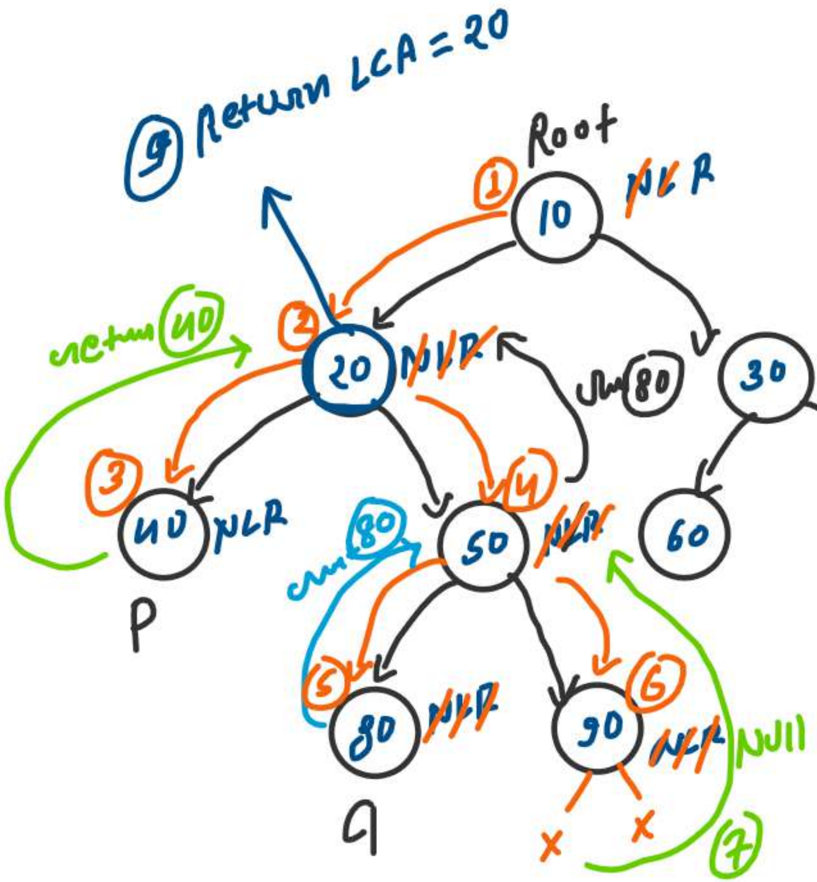
DRY RUN

EX2

P=40

q=80

O/P=20



1. 10≠q { 10≠P
2. 20≠q { 20≠P
3. 40==P Return P
4. 50≠q { 50≠P
5. 80==q return q
6. 90≠q { 90≠P
7. root == NULL return NULL
8. 80 and NULL return 80
9. P==40 and q==80 return root (LCA)

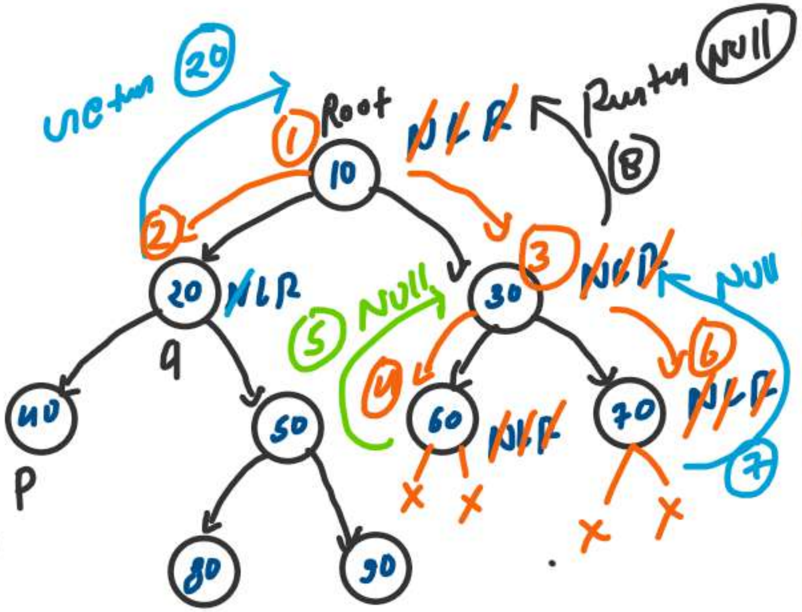
DRY RUN

EX 3

p = 40

q = 20

o/p = 20



① 10 ≠ P { 10 ≠ q

② 20 == q return q

③ 30 ≠ p and q

④ 60 ≠ p and q

⑤ Root == Null
return Null

⑥ 70 ≠ p and q

⑦ root == Null
return Null

⑧ Null and Null
return Null

⑨ q == 20 and Null

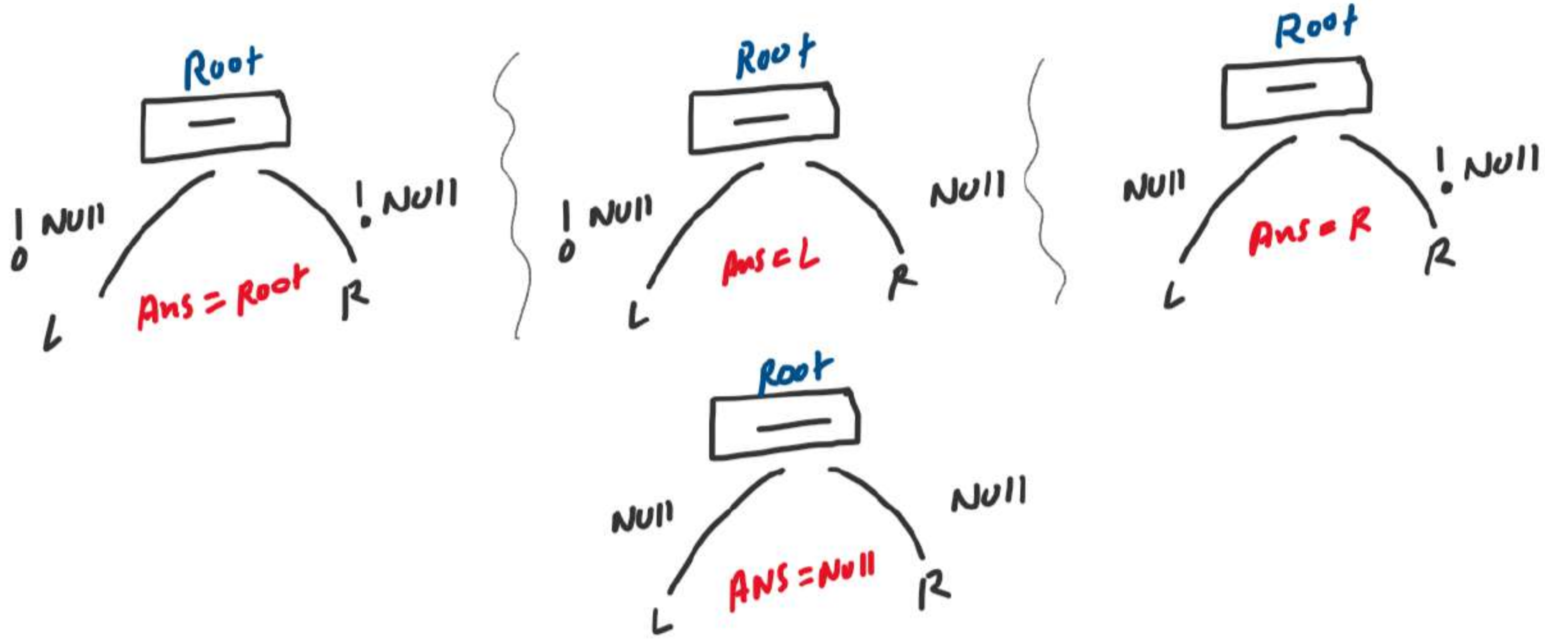
↳ Return q as LCA

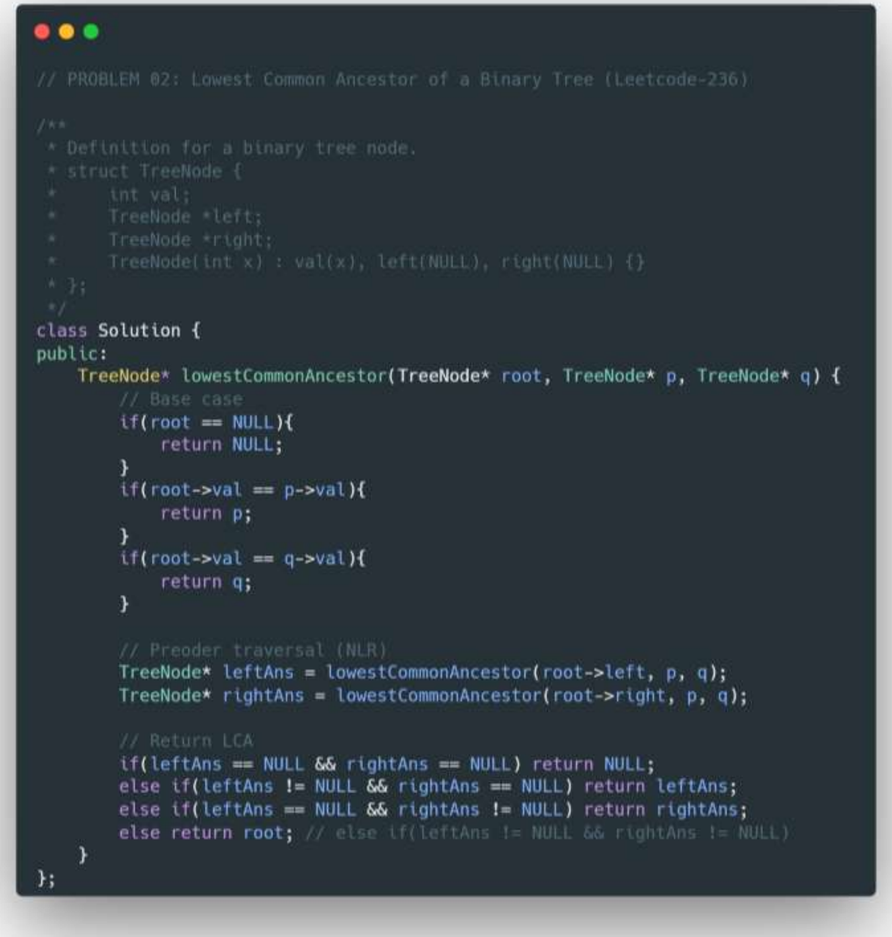
$\boxed{\text{Ans} = q}$

3 Base Cases

```
if( Root == Null )
  ↳ return Null

if( Root→val == p→val )
  ↳ Return p

if ( Root→val == q→val )
  ↳ Return q
```

4 Condition to LCA



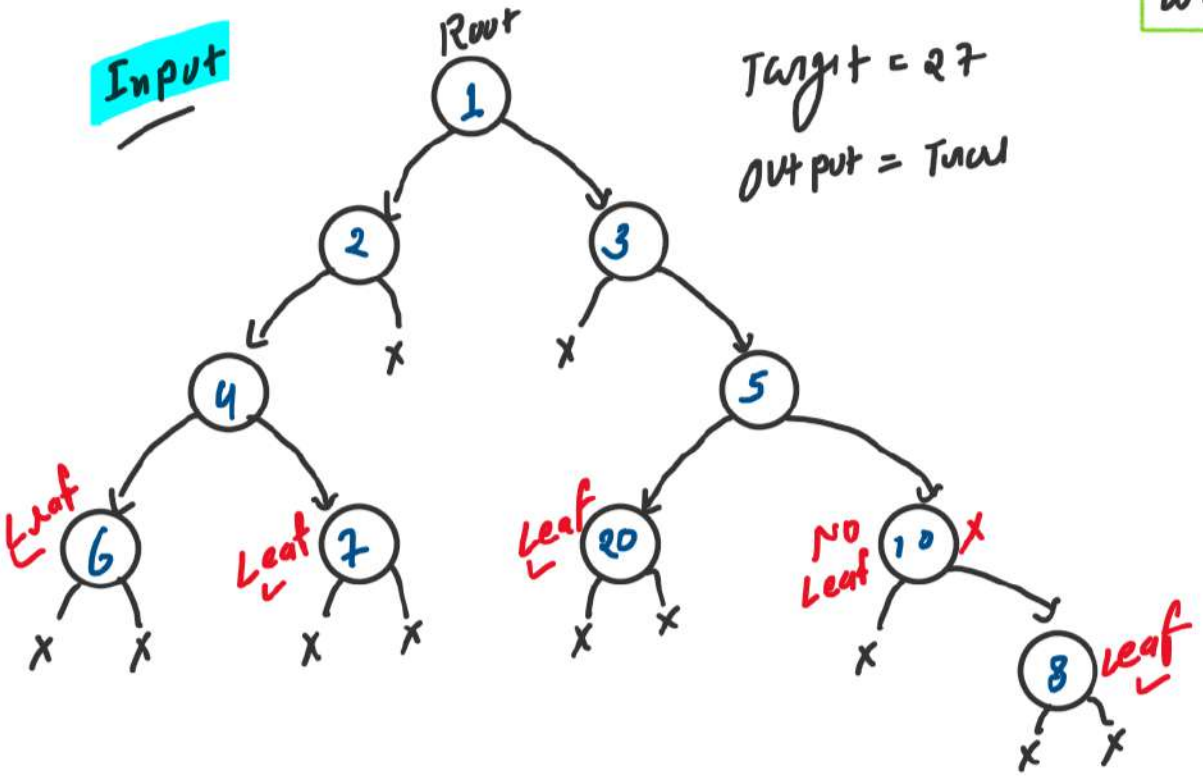


```cpp
// PROBLEM 02: Lowest Common Ancestor of a Binary Tree (Leetcode-236)

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        // Base case
        if(root == NULL){
            return NULL;
        }
        if(root->val == p->val){
            return p;
        }
        if(root->val == q->val){
            return q;
        }

        // Preoder traversal (NLR)
        TreeNode* leftAns = lowestCommonAncestor(root->left, p, q);
        TreeNode* rightAns = lowestCommonAncestor(root->right, p, q);

        // Return LCA
        if(leftAns == NULL && rightAns == NULL) return NULL;
        else if(leftAns != NULL && rightAns == NULL) return leftAns;
        else if(leftAns == NULL && rightAns != NULL) return rightAns;
        else return root; // else if(leftAns != NULL && rightAns != NULL)
    }
};
```

Time and space complexity = ?

3. Path Sum (Leetcode-112)

Input



Target = 27

Output = True

What is path? Root → Leaf Node

Explaination

① 1 + 2 + 4 + 6 = 13

② 1 + 2 + 4 + 7 = 14

③ 1 + 3 + 5 + 20 = 29

X ④ 1 + 3 + 5 + 10 = 18 X Not a path

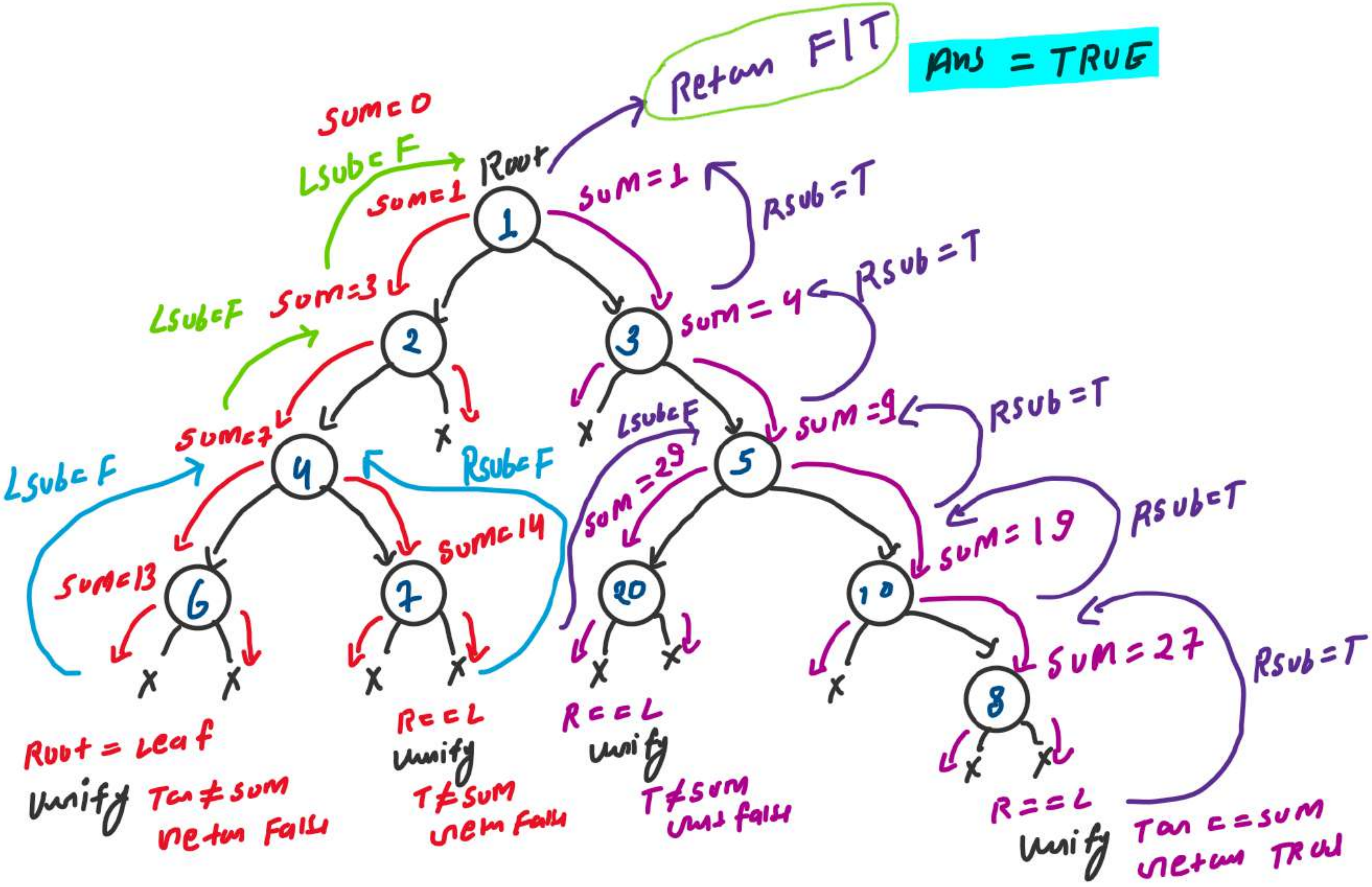
⑤ 1 + 3 + 5 + 10 + 8 = 27

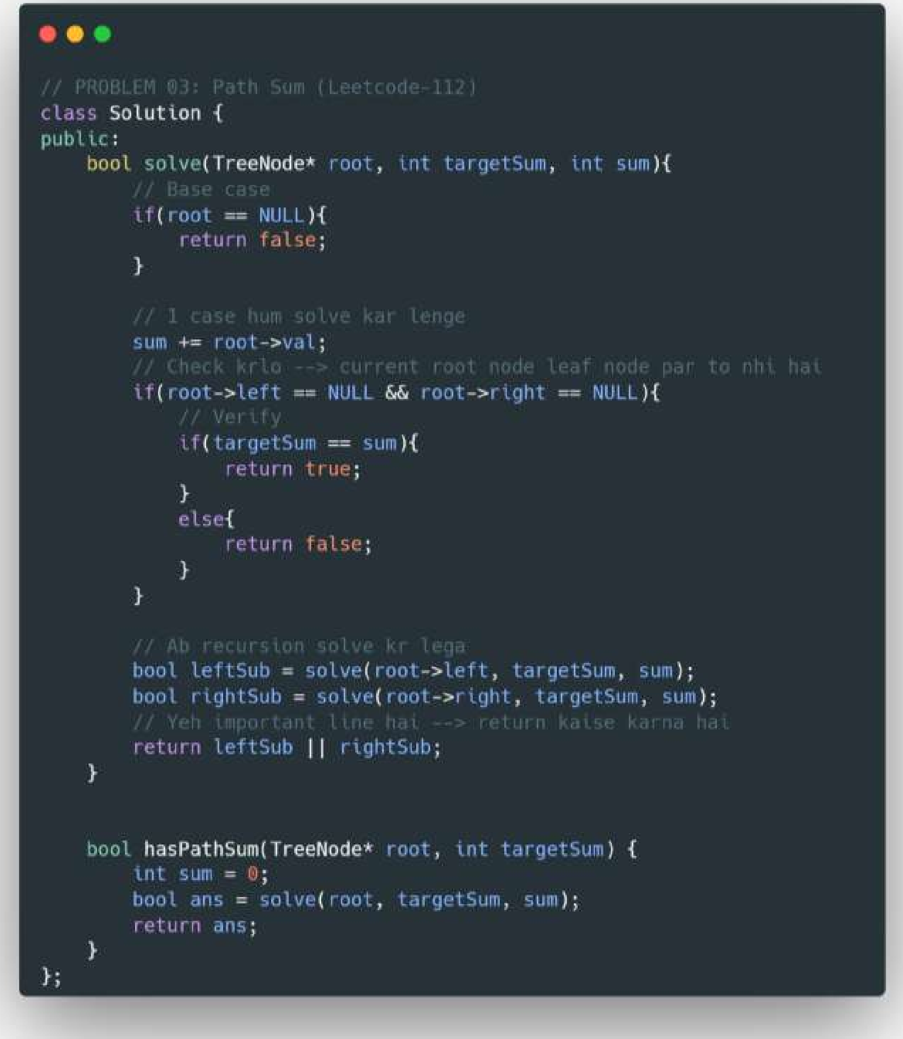
Target == path⑤

↳ Ans = TRUE

DRY RUN

Target = 27
output = True





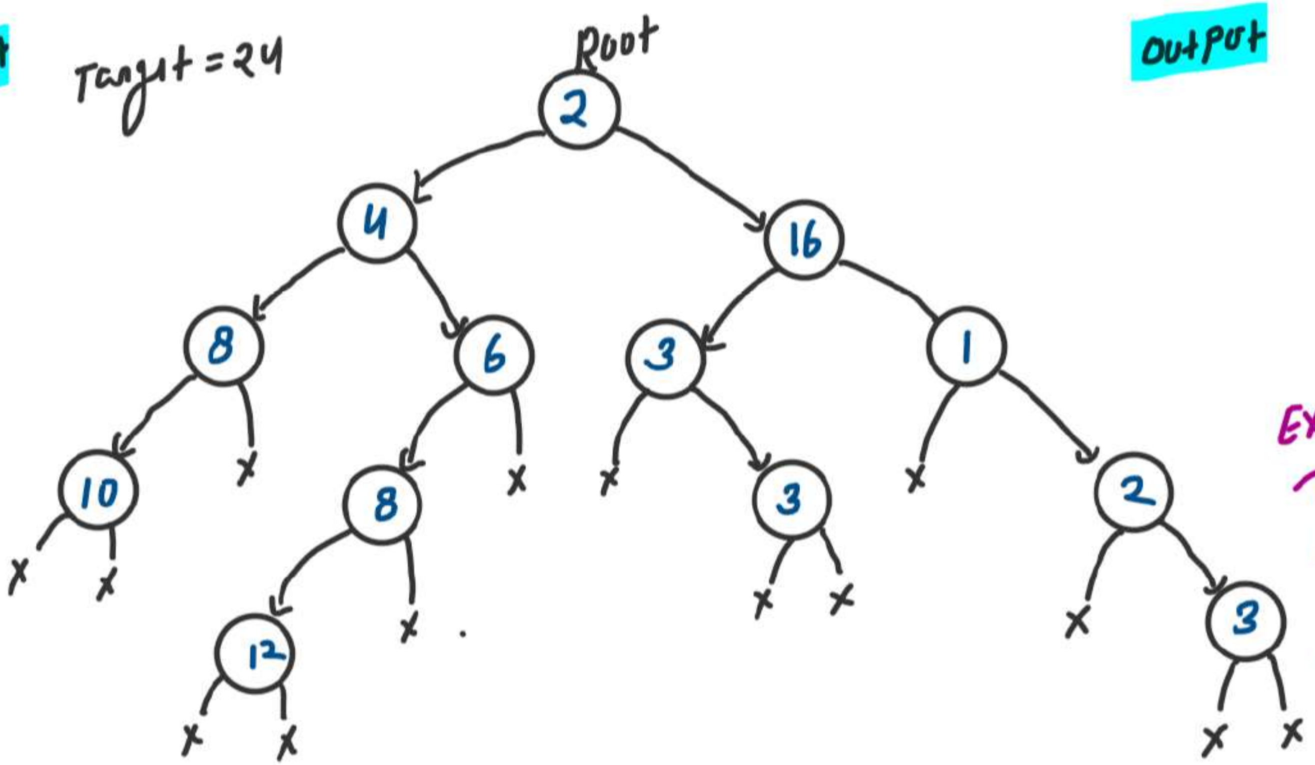
```
// PROBLEM 03: Path Sum (Leetcode-112)
class Solution {
public:
    bool solve(TreeNode* root, int targetSum, int sum){
        // Base case
        if(root == NULL){
            return false;
        }

        // 1 case hum solve kar lenge
        sum += root->val;
        // Check krlo --> current root node leaf node par to nhi hai
        if(root->left == NULL && root->right == NULL){
            // Verify
            if(targetSum == sum){
                return true;
            }
            else{
                return false;
            }
        }

        // Ab recursion solve kr lega
        bool leftSub = solve(root->left, targetSum, sum);
        bool rightSub = solve(root->right, targetSum, sum);
        // Yeh important line hai --> return kaise karna hai
        return leftSub || rightSub;
    }


    bool hasPathSum(TreeNode* root, int targetSum) {
        int sum = 0;
        bool ans = solve(root, targetSum, sum);
        return ans;
    }
};
```

T.C. = ?   and   S.C. = ?

4. Path Sum (Leetcode-113)

Input

Target = 24



Output

2D- ARRAY

2	4	8	10	
2	16	3	3	
2	16	1	2	3

vector < vector <int>> ans

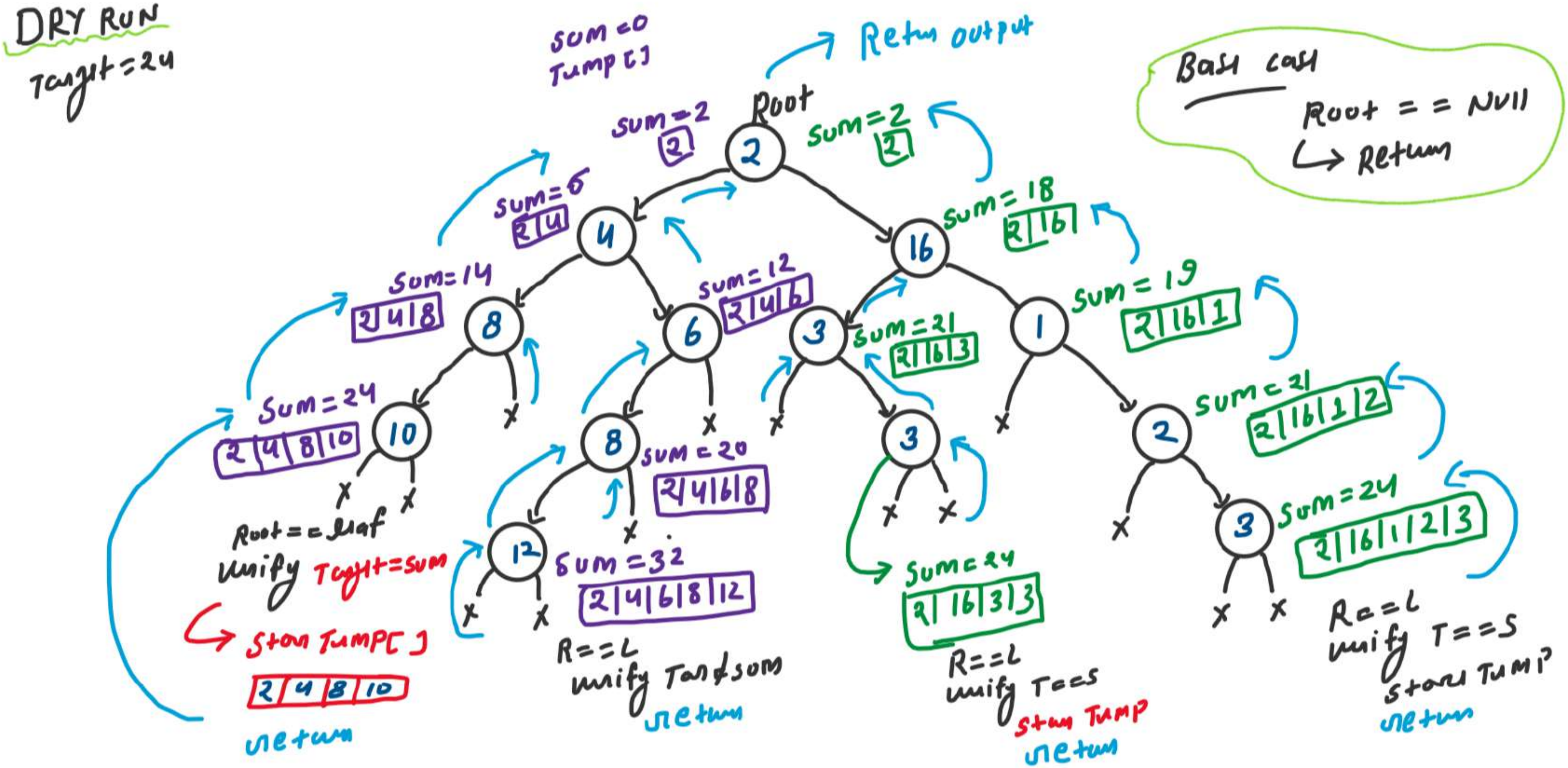
Explanation

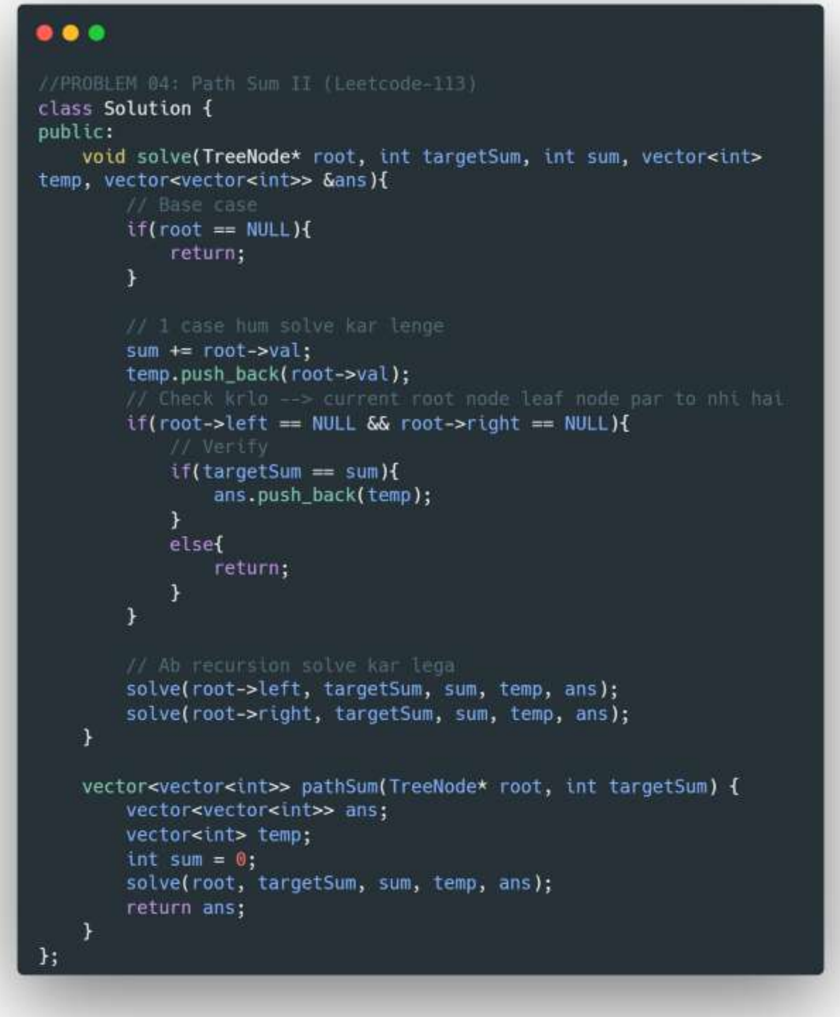
✓① 2 → 4 → 8 → 10 = 24

✗② 2 → 4 → 6 → 8 → 12 = 32

✓③ 2 → 16 → 3 → 3 = 24

✓④ 2 → 16 → 1 → 2 → 3 = 24


DRY RUN
Target=24
Sum=0
Temp[]
Retun output
Root
Base case
Root == Null
↳ Return
Sum=2
2
Sum=2
2
2
4
16
Sum=6
2|4
Sum=18
2|16
8
6
3
1
Sum=14
2|4|8
Sum=12
2|4|6
Sum=21
2|16|3
Sum=19
2|16|1
10
Sum=24
2|4|8|10
8
Sum=20
2|4|6|8
3
2
Sum=21
2|16|1|2
12
Sum=32
2|4|6|8|12
3
Sum=24
2|16|1|2|3
Sum=24
2|16|3|3
Root==leaf
verify Target=sum
↳ store Temp[ ]
2|4|8|10
return
R==L
verify Tar&sum
return
R==L
verify T==S
Store Temp
return
R==L
verify T==S
store Temp
return
x



```
//PROBLEM 04: Path Sum II (Leetcode-113)
class Solution {
public:
    void solve(TreeNode* root, int targetSum, int sum, vector<int>
temp, vector<vector<int>> &ans){
        // Base case
        if(root == NULL){
            return;
        }

        // 1 case hum solve kar lenge
        sum += root->val;
        temp.push_back(root->val);
        // Check krlo --> current root node leaf node par to nhi hai
        if(root->left == NULL && root->right == NULL){
            // Verify
            if(targetSum == sum){
                ans.push_back(temp);
            }
            else{
                return;
            }
        }

        // Ab recursion solve kar lega
        solve(root->left, targetSum, sum, temp, ans);
        solve(root->right, targetSum, sum, temp, ans);
    }

    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<vector<int>> ans;
        vector<int> temp;
        int sum = 0;
        solve(root, targetSum, sum, temp, ans);
        return ans;
    }
};
```

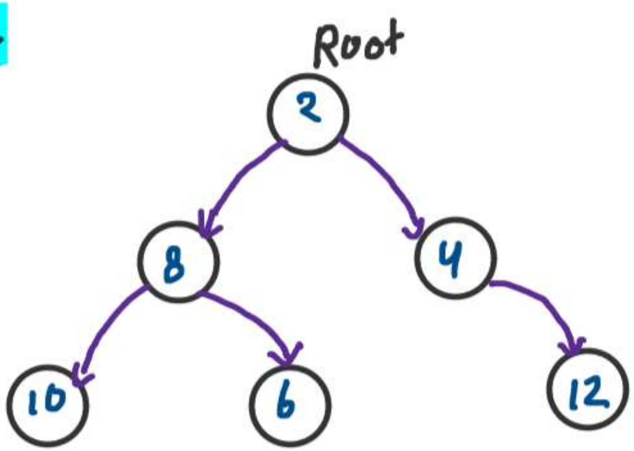
T.C. and S.C. = ?

📂 6. Construct Binary Tree from In-Order and Pre-Order Traversal (Leetcode-105)

Input

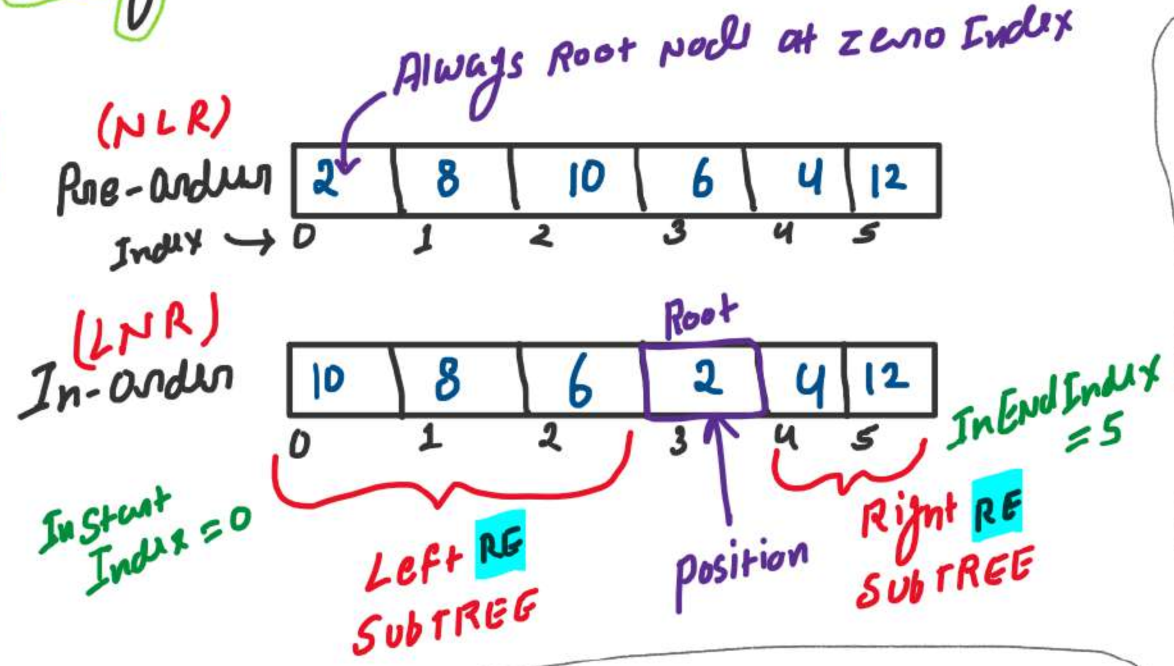
Pre-order	2	8	10	6	4	12
In-order	10	8	6	2	4	12

Output



Logic building ---

Input



(NLR) Pre-order	2	8	10	6	4	12
Index →	0	1	2	3	4	5

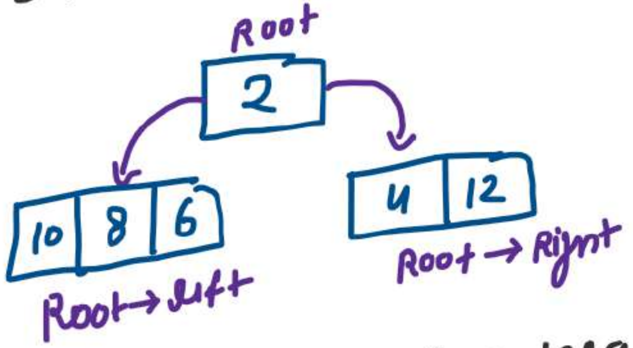
(LNR) In-order	10	8	6	2	4	12
	0	1	2	3	4	5

Base case Index >= size || InStartIndex > InEND Index

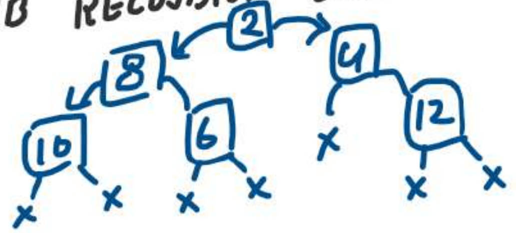
→ Return Null

STEP 01 Find position index of Root Node in In-Order Array

1 case Hum solve karenge



AB Recursion solve kar lega

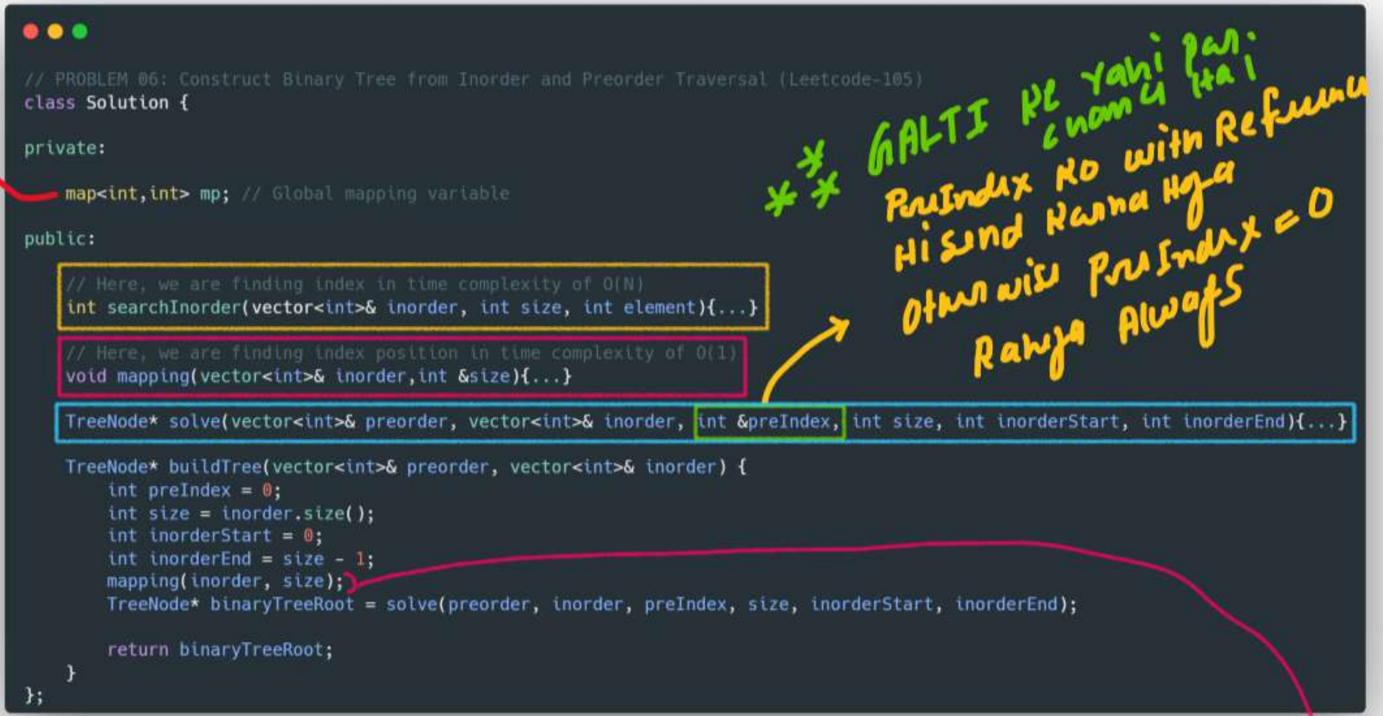


In-order

10	8	6	2	4	12
0	1	2	3	4	5

Map

EME-	Index
10	0
8	1
6	2
2	3
4	4
12	5

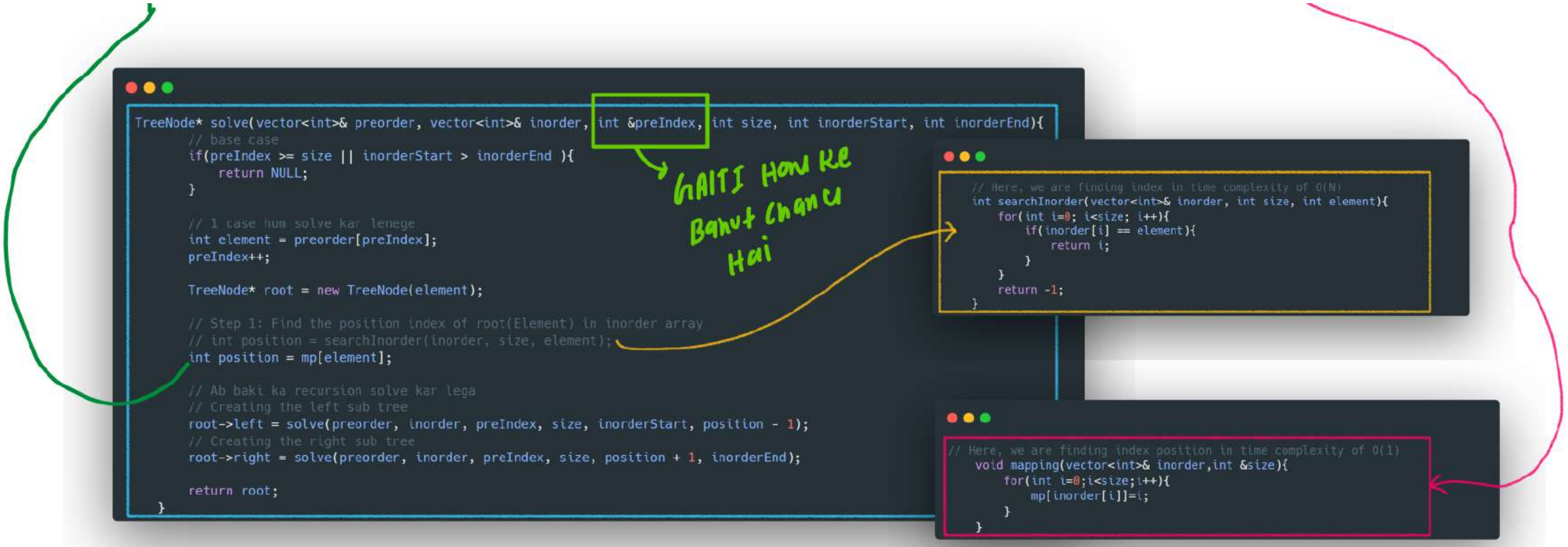


```
// PROBLEM 06: Construct Binary Tree from Inorder and Preorder Traversal (Leetcode-105)
class Solution {

private:

    map<int,int> mp; // Global mapping variable

public:

    // Here, we are finding index in time complexity of O(N)
    int searchInorder(vector<int>& inorder, int size, int element){...}

    // Here, we are finding index position in time complexity of O(1)
    void mapping(vector<int>& inorder,int &size){...}

    TreeNode* solve(vector<int>& preorder, vector<int>& inorder, int &preIndex, int size, int inorderStart, int inorderEnd){...}

    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        int preIndex = 0;
        int size = inorder.size();
        int inorderStart = 0;
        int inorderEnd = size - 1;
        mapping(inorder, size);
        TreeNode* binaryTreeRoot = solve(preorder, inorder, preIndex, size, inorderStart, inorderEnd);

        return binaryTreeRoot;
    }
};
```

** GALTI Ke Yahi pari chance Hai

PreIndex Ko with Reference Hi Send Karna Hoga otherwise PreIndex = 0 Rahega Always

To Find the position Index of Element ⇒ map is best due to T.C. = O(1) Rather than O(N).



```cpp
TreeNode* solve(vector<int>& preorder, vector<int>& inorder, int &preIndex, int size, int inorderStart, int inorderEnd){
        // base case
        if(preIndex >= size || inorderStart > inorderEnd ){
            return NULL;
        }

        // 1 case hum solve kar lenege
        int element = preorder[preIndex];
        preIndex++;

        TreeNode* root = new TreeNode(element);

        // Step 1: Find the position index of root(Element) in inorder array
        // int position = searchInorder(inorder, size, element);
        int position = mp[element];

        // Ab baki ka recursion solve kar lega
        // Creating the left sub tree
        root->left = solve(preorder, inorder, preIndex, size, inorderStart, position - 1);
        // Creating the right sub tree
        root->right = solve(preorder, inorder, preIndex, size, position + 1, inorderEnd);

        return root;
    }
```

GAlTI Hon Ke Bahut Chance Hai

```cpp
    // Here, we are finding index in time complexity of O(N)
    int searchInorder(vector<int>& inorder, int size, int element){
        for( int i=0; i<size; i++){
            if( inorder[i] == element){
                return i;
            }
        }
        return -1;
    }
```

```cpp
// Here, we are finding index position in time complexity of O(1)
    void mapping(vector<int>& inorder,int &size){
        for(int i=0;i<size;i++){
            mp[inorder[i]]=i;
        }
    }
```

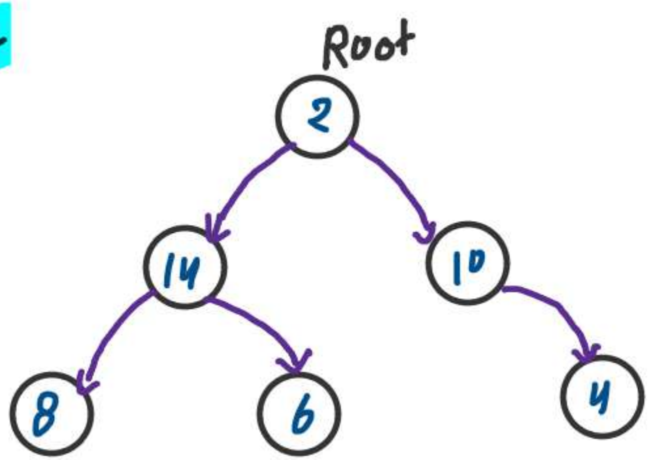
***Time** and **space complexity** is O(N), Where N is number of elements in array*

7. Construct Binary Tree from In-Order and Post-Order Traversal (Leetcode-106)

Input

(LRN) Post-order	8	6	14	4	10	2
(LNR) In-order	8	14	6	2	10	4

Output



Logic Building

Post order me Root Node Always Right most Element He Hoga

(LRN)
Post-order

0	1	2	3	4	5
8	6	14	4	10	2

5 → Pre Index

(LNR)
In-order

0	1	2	3	4	5
8	14	6	2	10	4

InStart → 0

3 pos

5 → InEnd

Base case

(PostIndex < 0 || InStart > InEnd)

→ Return NULL

Yahi Important Chiz Hai

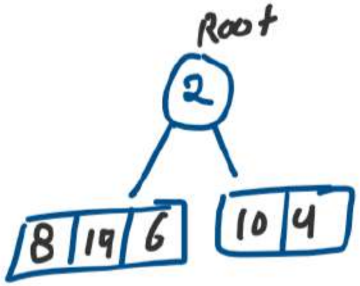
→ Right call 1st
Left call 2nd

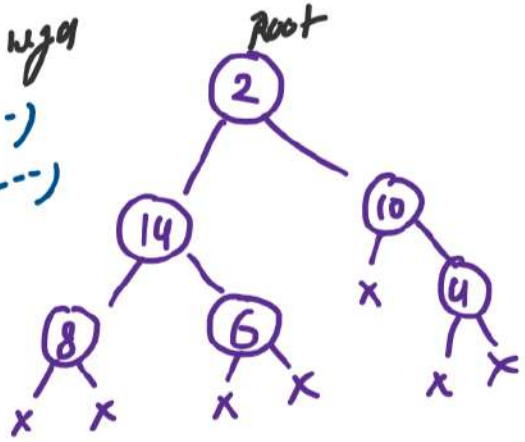
1 case Hum solve karenge

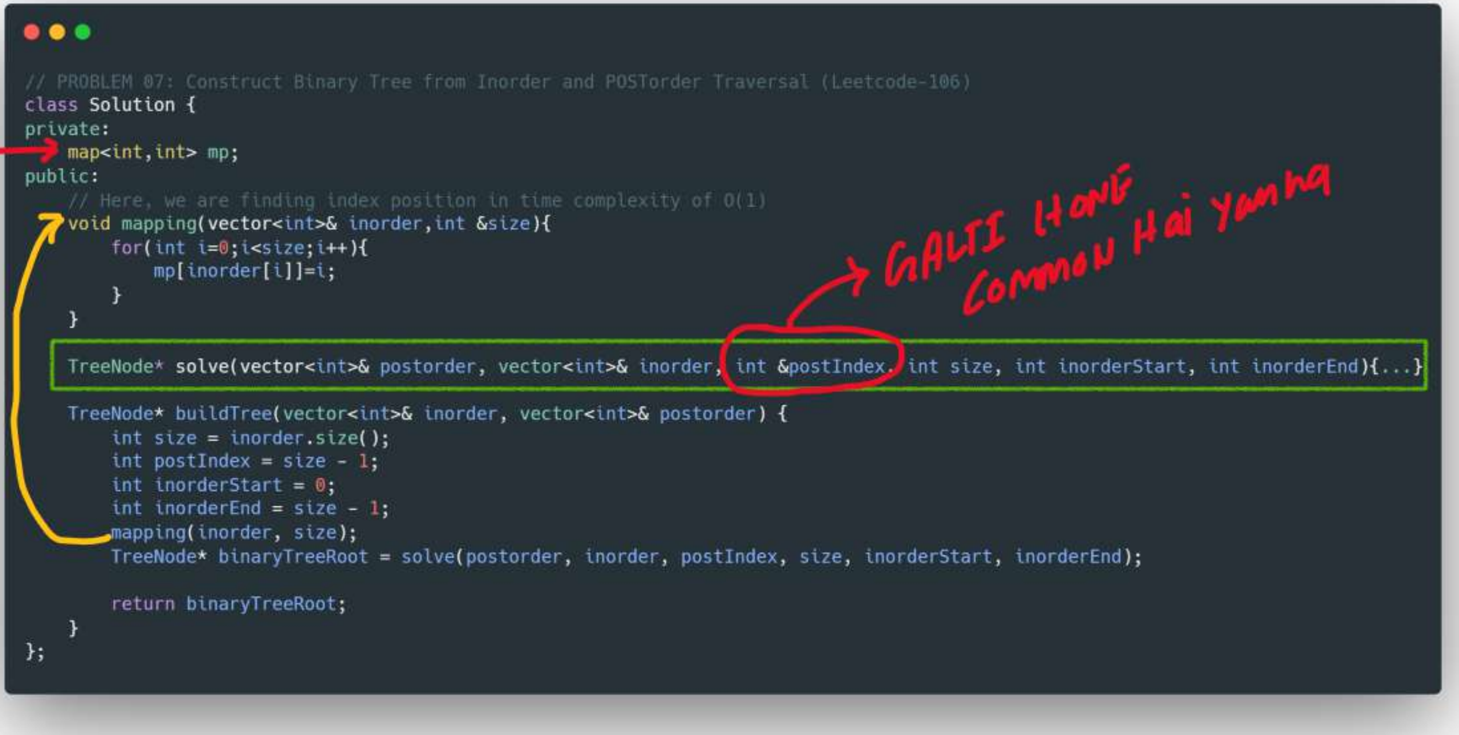
```
Element = 2
PostIndex--;
Node* Root = new Node(Element);
```

AB Recursion solve Kar dega

```
Root→Right = solve(---)
Root→Left = solve(---)
```

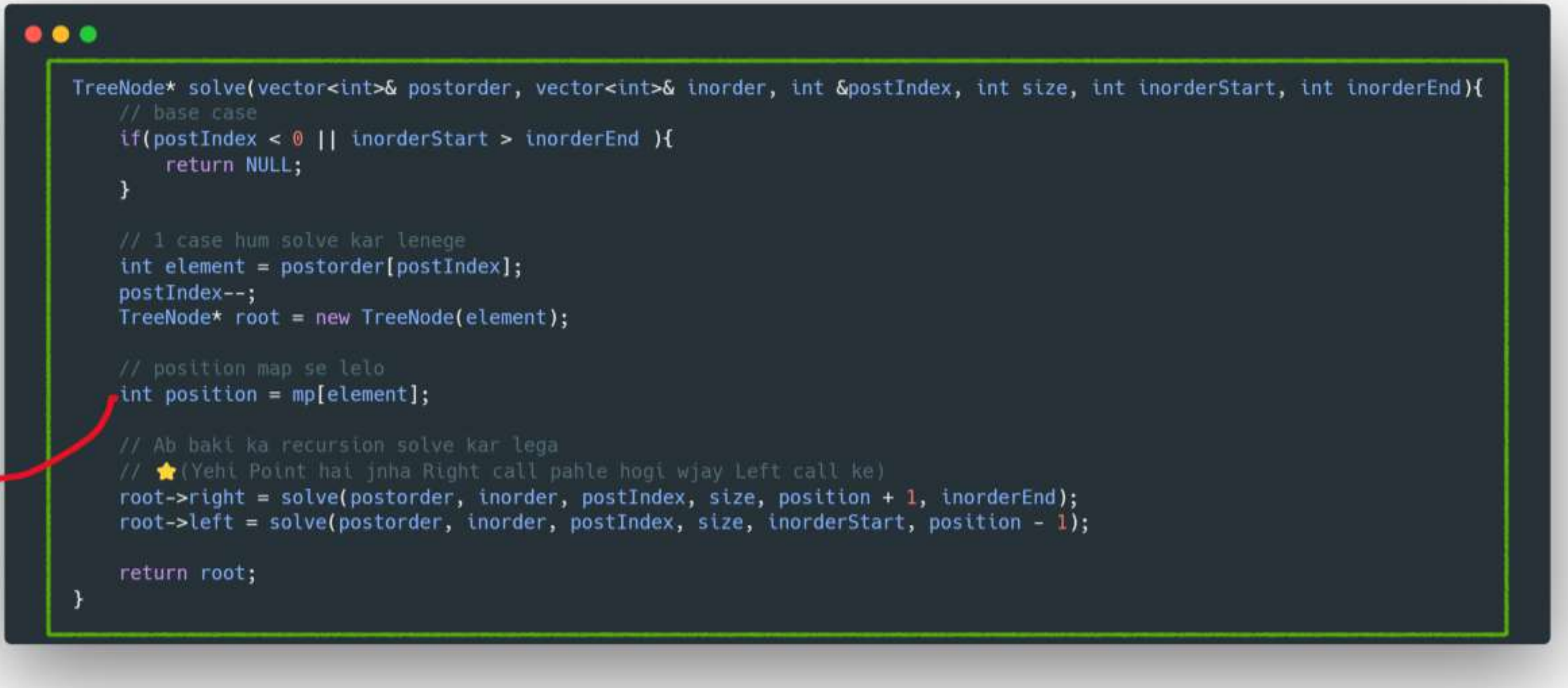






```
// PROBLEM 07: Construct Binary Tree from Inorder and POSTorder Traversal (Leetcode-106)
class Solution {
private:
    map<int,int> mp;
public:
    // Here, we are finding index position in time complexity of O(1)
    void mapping(vector<int>& inorder,int &size){
        for(int i=0;i<size;i++){
            mp[inorder[i]]=i;
        }
    }

    TreeNode* solve(vector<int>& postorder, vector<int>& inorder, int &postIndex, int size, int inorderStart, int inorderEnd){...}

    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int size = inorder.size();
        int postIndex = size - 1;
        int inorderStart = 0;
        int inorderEnd = size - 1;
        mapping(inorder, size);
        TreeNode* binaryTreeRoot = solve(postorder, inorder, postIndex, size, inorderStart, inorderEnd);

        return binaryTreeRoot;
    }
};
```

GALTI HONE
COMMON Hai Yanha



```
TreeNode* solve(vector<int>& postorder, vector<int>& inorder, int &postIndex, int size, int inorderStart, int inorderEnd){
    // base case
    if(postIndex < 0 || inorderStart > inorderEnd ){
        return NULL;
    }

    // 1 case hum solve kar lenege
    int element = postorder[postIndex];
    postIndex--;
    TreeNode* root = new TreeNode(element);

    // position map se lelo
    int position = mp[element];

    // Ab baki ka recursion solve kar lega
    // ⭐(Yehi Point hai jnha Right call pahle hogi wjay Left call ke)
    root->right = solve(postorder, inorder, postIndex, size, position + 1, inorderEnd);
    root->left = solve(postorder, inorder, postIndex, size, inorderStart, position - 1);

    return root;
}
```

***Time** and **space complexity** is O(N), Where N is number of elements in array*