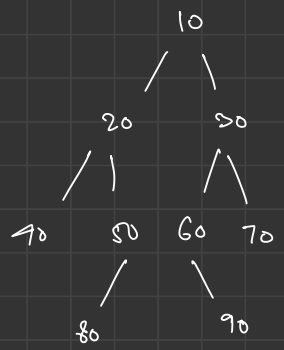
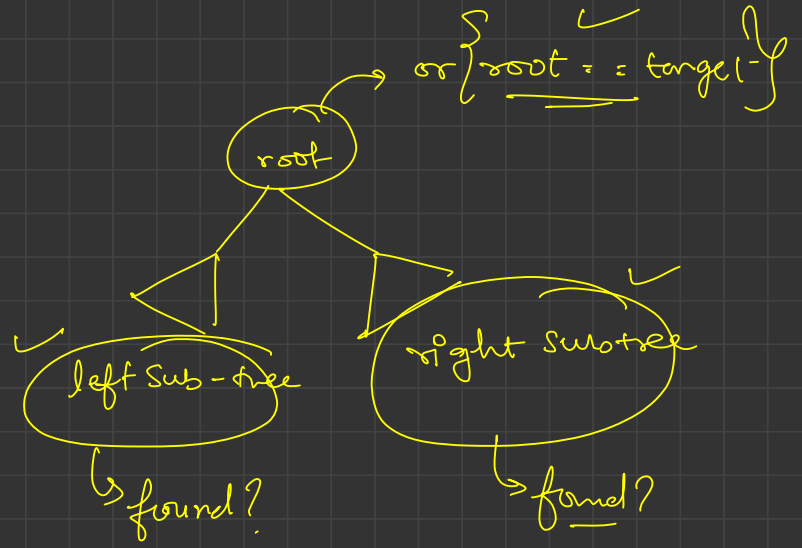
find a given node in a tree

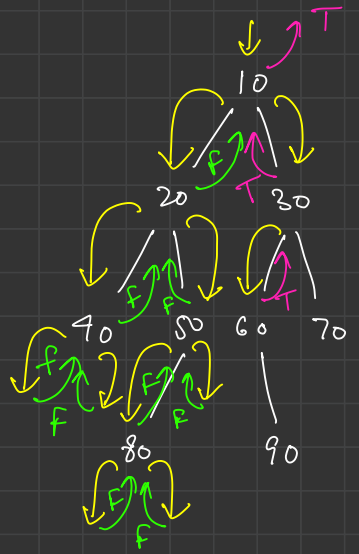


find 60



```java
boolean find(Node root, int val) {
    if (root == null) {
        return false;
    }

    if (root.data == val) {
        return true;
    }

    boolean filc = find (root.left, val);
    if (filc == true) {
        return true;
    }

    boolean firc = find (root.right, val);
    if (firc == true) {
        return true;
    }

    return false;
}
```

60

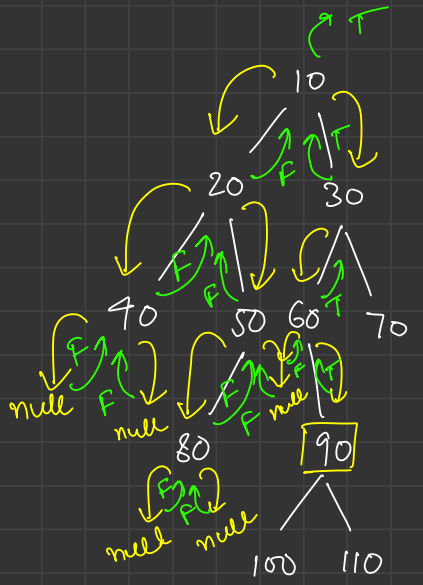


target = 60

Node to Root Path

```java
ArrayList<Integer> n2r = new ArrayList<>();

boolean Node2Root(Node root, int val) {
    if (root == null) {
        return false;
    }

    if (root.data == val) {
        n2r.add(root.data);
        return true;
    }

    boolean filc = Node2Root(root.left, val);
    if (filc == true) {
        n2r.add(root.data);
        return true;
    }

    boolean firc = Node2Root(root.right, val);
    if (firc == true) {
        n2r.add(root.data);
        return true;
    }

    return false;
}
```

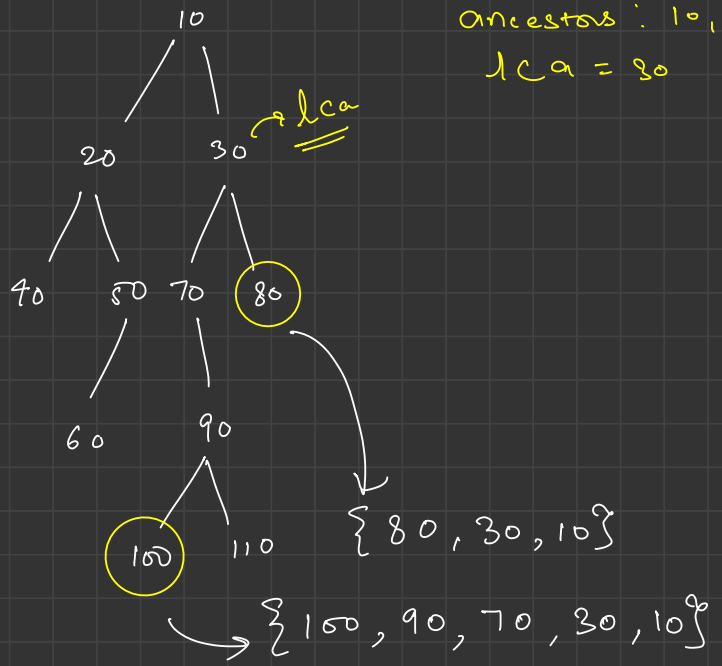
90



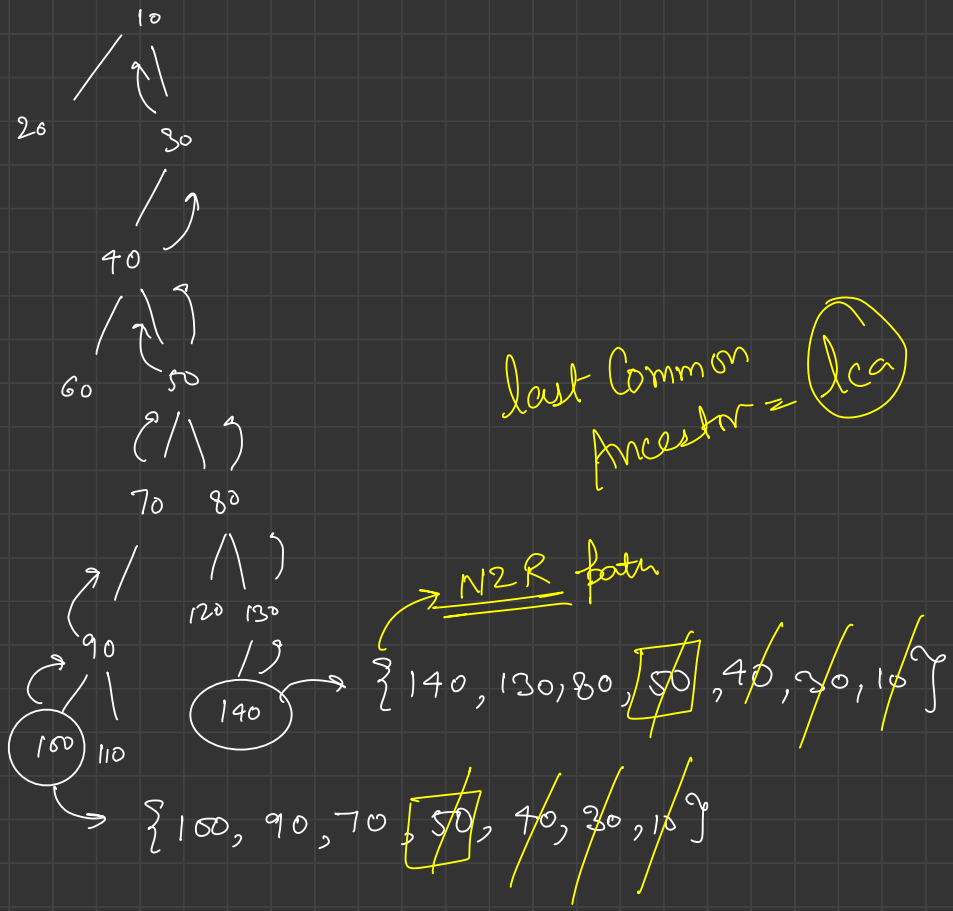
{90, 60, 30, 10} N2R path !

(n2r → reverse(n2r))

Lowest Common Ancestor (LCA)



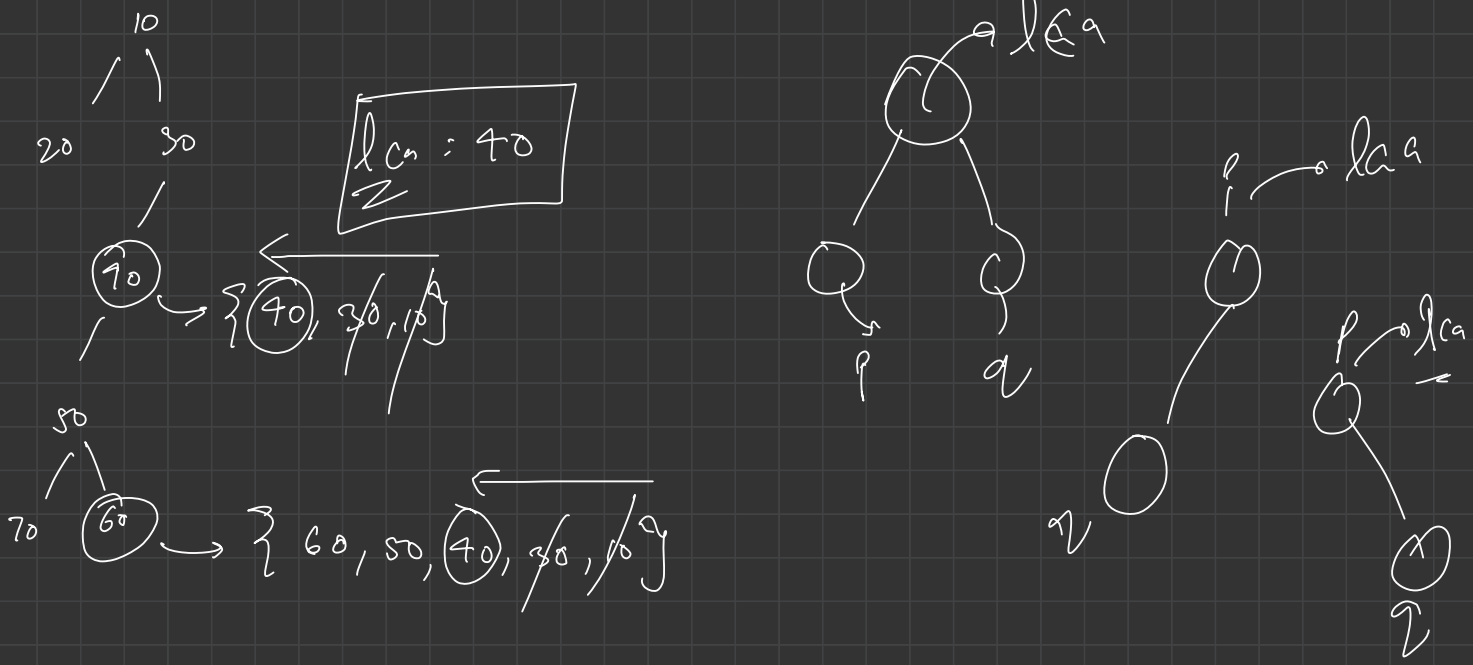
ancestors : 10, 30
lca = 30

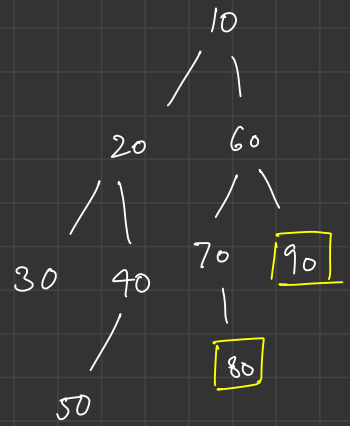

10
20
30
40
60
50
70
80
90
120
130
100
110
140
Last Common Ancestor = lca
N2R path
{140, 130, 80, 50, 40, 30, 10}
{100, 90, 70, 50, 40, 30, 10}

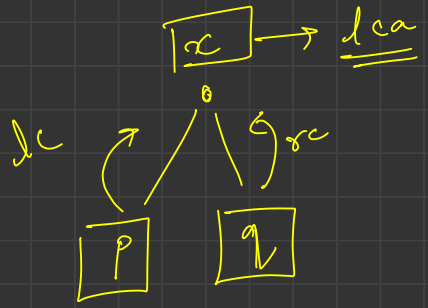
Step1. get All the ancestors, i.e. N2R path

Step2. find last common ancestor in them.

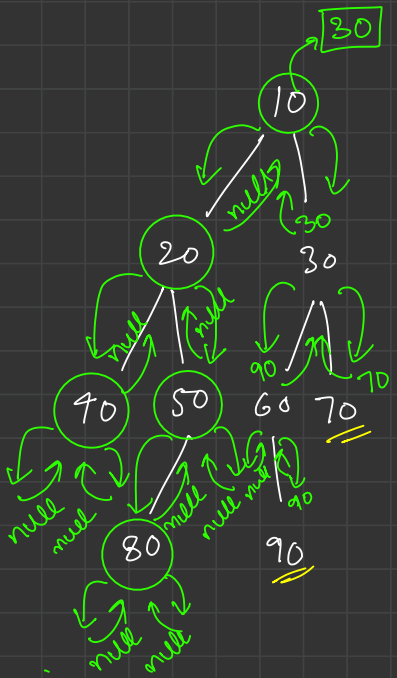
↳ lca


10
20
30
lca : 40
40
{40, 30, 10}
50
70
60
{60, 50, 40, 30, 10}
lca
p
q
lca
p
q
lca
p
q



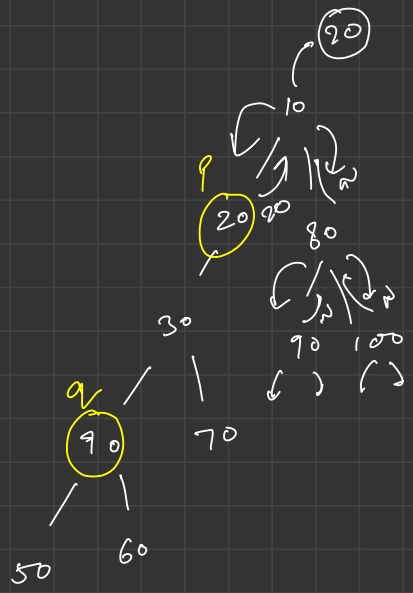


{ faith find any one of them in my child }


30
10
null
30
20
30
null
null
90
70
40
50
60
70
null
null
null
null
90
80
90
null
null

assumption:
Both are present in the tree

```
if (root == null)
    return null;

if (root.data == p or q)
    return root;

Node leftChild = findLca(root.left, p, q);
Node rightChild = findLca(root.right, p, q);

if (leftChild != null and rightChild != null)
    return root;

if (leftChild != null)
    return leftChild;
if (RightChild != null)
    return right child;
return null;
```

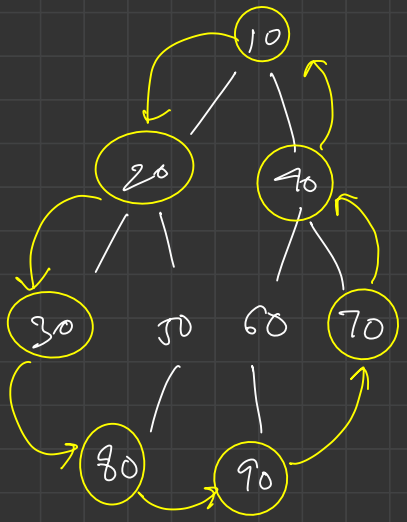

20
10
p
20
90
80
30
90
100
q
90
70
50
60

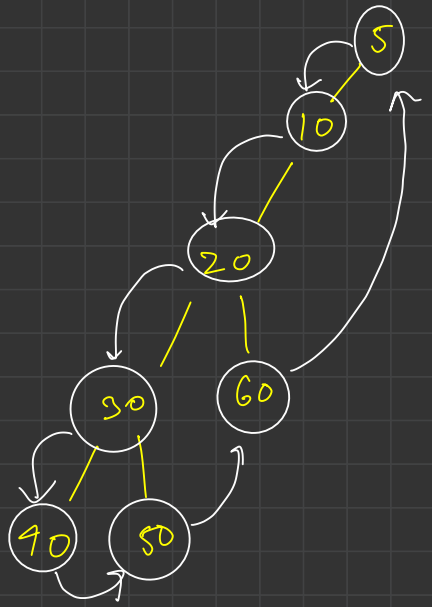
Boundry Traversal

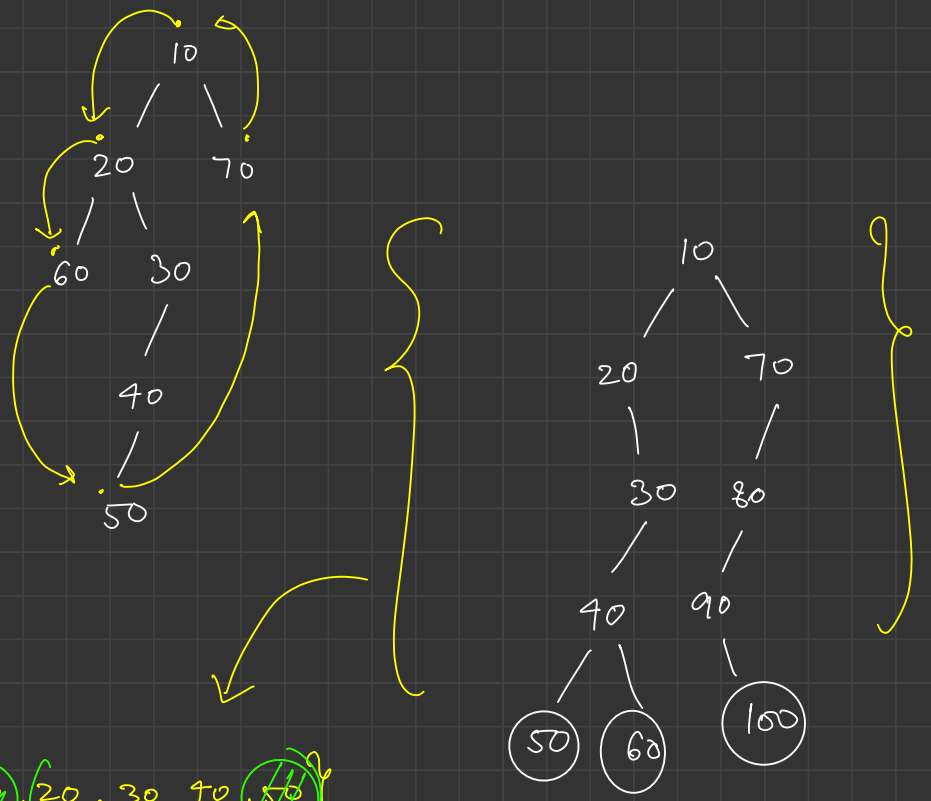
{10, 20, 30, 80, 90, 70, 40}

↓

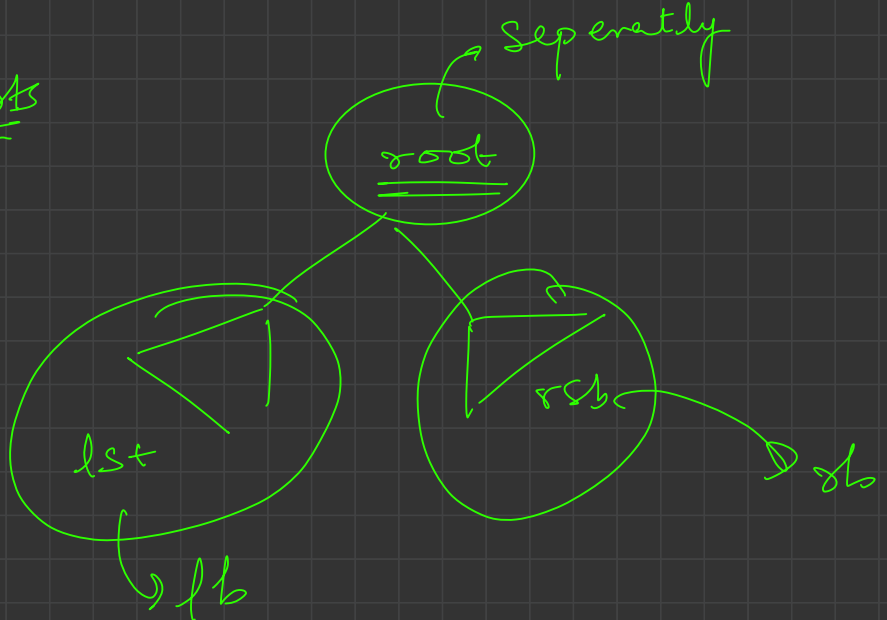
Boundry




5
10
20
30
60
40
50


10
20
70
60
30
40
50
10
20
70
30
80
40
90
50
60
100
lb
{10, 20, 30, 40, 50}
No leaf
Bb
{50, 60, 100}
No leaf
rb
{100, 90, 80, 70, 60}
(Post order work)

4 parts



root

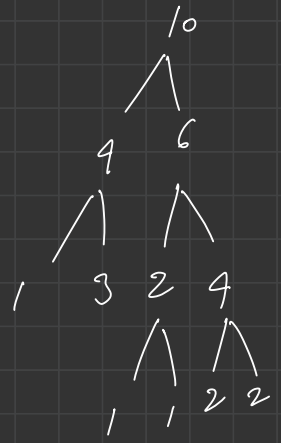
lb(root.left)

leafNode

rb(root.right)

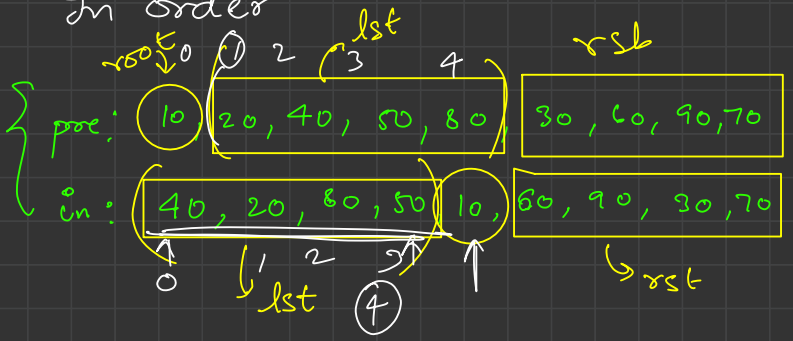
→ No leaf

Children Sum Property



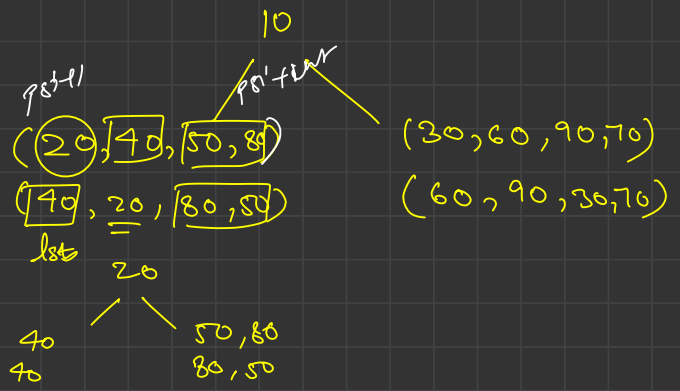
psi + cnt1 != pei

Build a tree from pre - In order



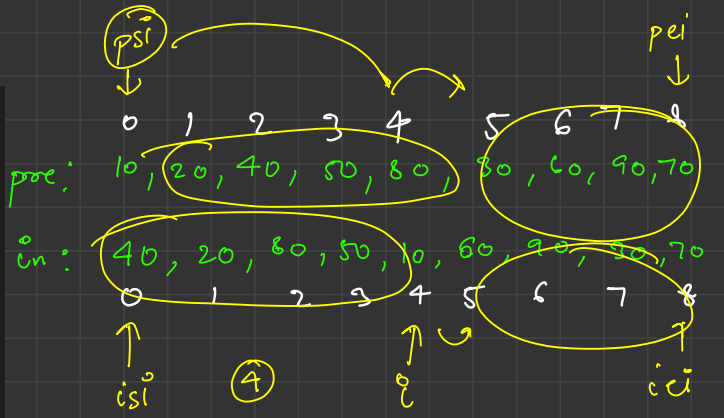
pre: root left right

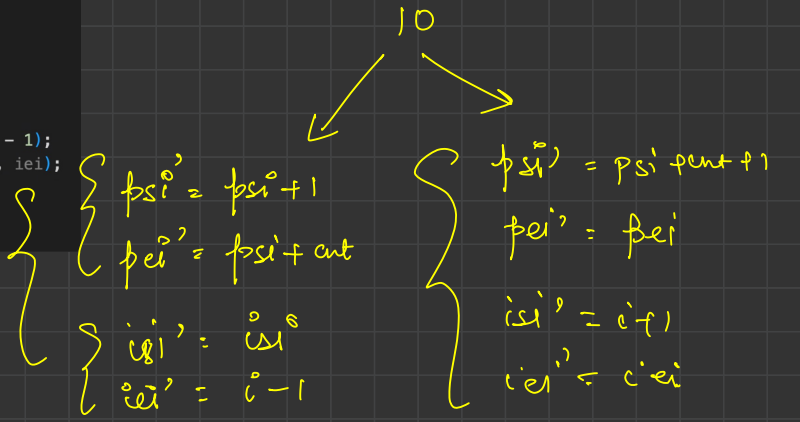
in: left root right

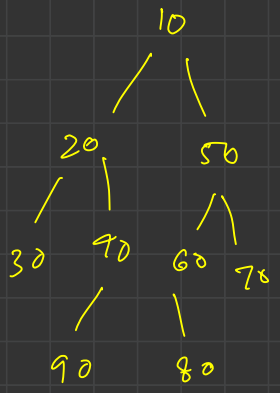


TC : O(N) SC : O(h)

```java
static Node construct(int[] preorder, int psi, int pei, int[] inorder, int isi, int iei) {
    if (psi > pei) {
        return null;
    }

    if (isi > iei) {
        return null;
    }

    Node root = new Node(preorder[psi]);

    int val = preorder[psi];
    int cntNodeInLeftSubTree = 0;
    int i = isi;

    while (inorder[i] != val) {
        cntNodeInLeftSubTree += 1;
        i++;
    }

    root.left = construct(preorder, psi + 1, psi + cntNodeInLeftSubTree, inorder, isi, i - 1);
    root.right = construct(preorder, psi + cntNodeInLeftSubTree + 1, pei, inorder, i + 1, iei);

    return root;
}
```







post = 30, 90, 40, 20, 80, 60, 70, 50, 10

qm = 30, 20, 90, 40, 10, 60, 80, 50, 70