Binary Tree

Important Questions:

Little bit

1. print level wise (queue)
2. check Balanced / Height Tree (recursion)
3. level order traversal (queue)
4. Mirror (recursion & queue)
5. construct Tree from Inorder and preorder / postorder } IV.I
6. ———————————
7. Zig-Zag traversal
8. Remove all leaf nodes.

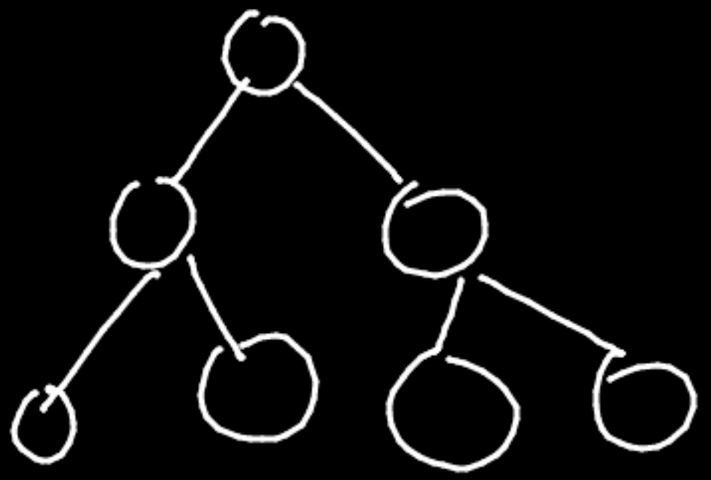
```
template <typename T>
Class BinaryTreeNode {

    T data;

    BinaryTreeNode *left;

    BinaryTreeNode *right;

    BinaryTree(T data)
    { this->data = data; left = NULL; right = NULL;
    }
    ~BinaryTreeNode() {
        delete left;
    } delete right;
};
```

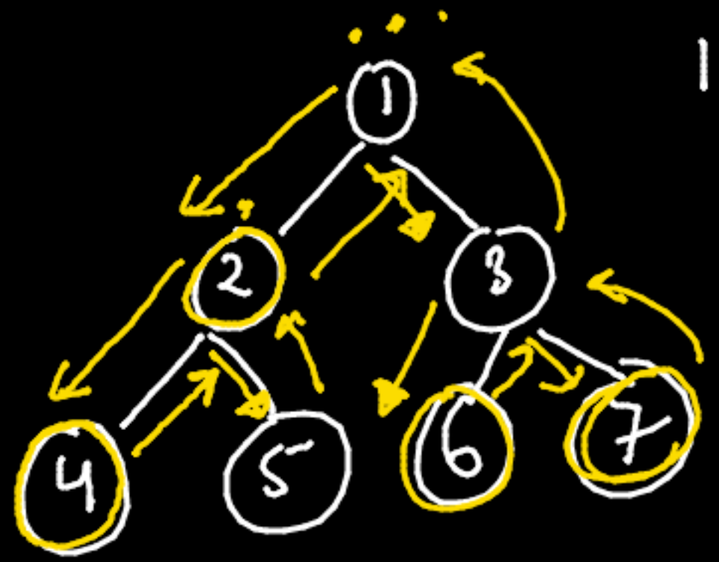


Max^m 2 child

1 2 4 5 3 6 7

Preorder BT

root , left , right



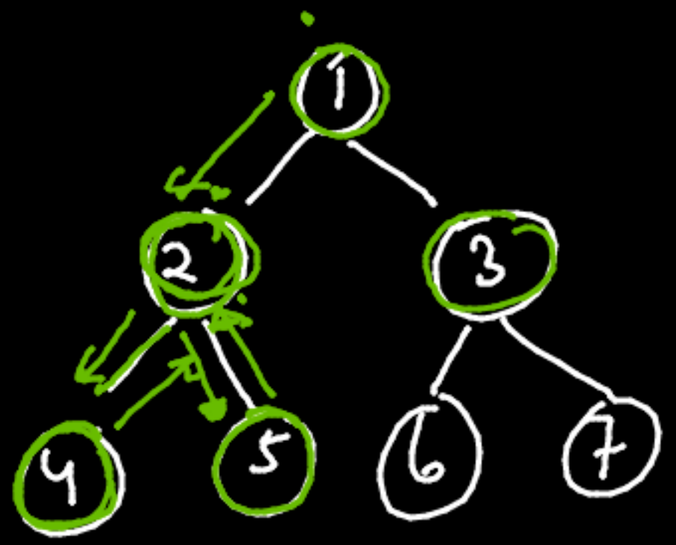
1 2 4 5 3 6 7

```
void preOrder(BinaryTreeNode<int> * root){
    if(root == NULL)
        return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}
```

4 5 2 6 7 3 1

PostOrder BT

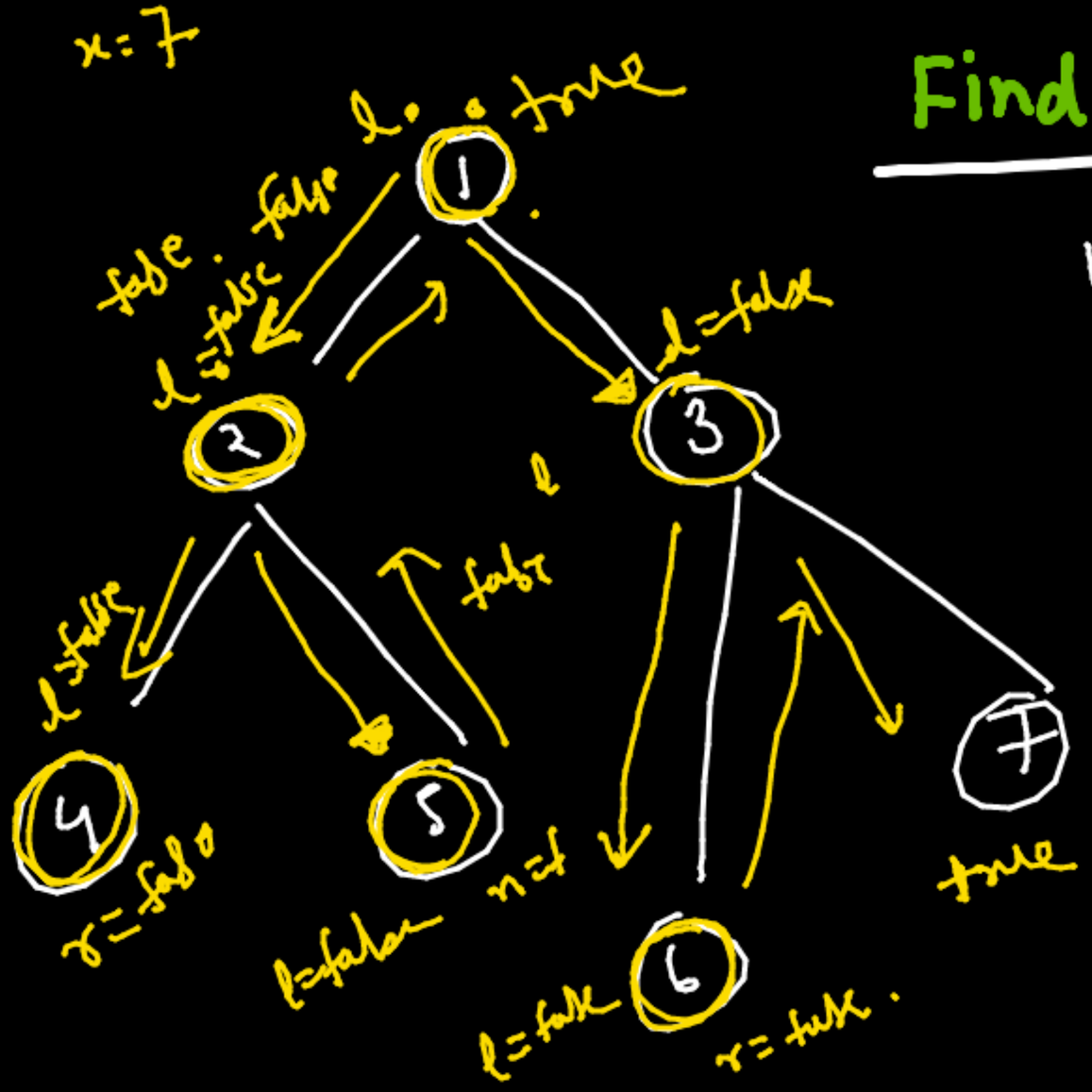
left, right, root




```
void postOrder(BinaryTreeNode<int> *root){
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right)
    cout<<root->data<<" ";
}
```

Find a Node

① queue ② recursion

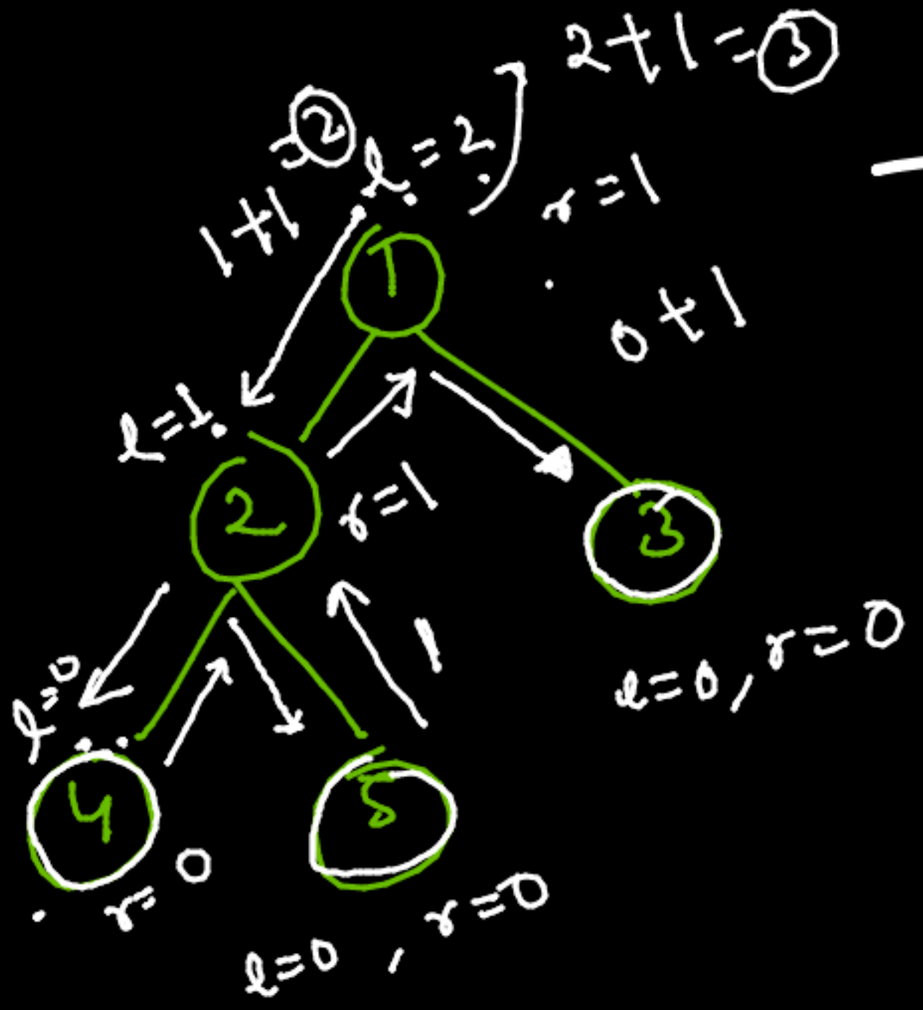



```
bool find( BinaryTreeNode <int> *root, int x) {
    if (root == NULL) return false;
    if (root->data == x)
        return true;

    bool l = find( root->left, x) ;
    bool r = find (root->right, x);

    return (l || r);
}
```

Height of BT



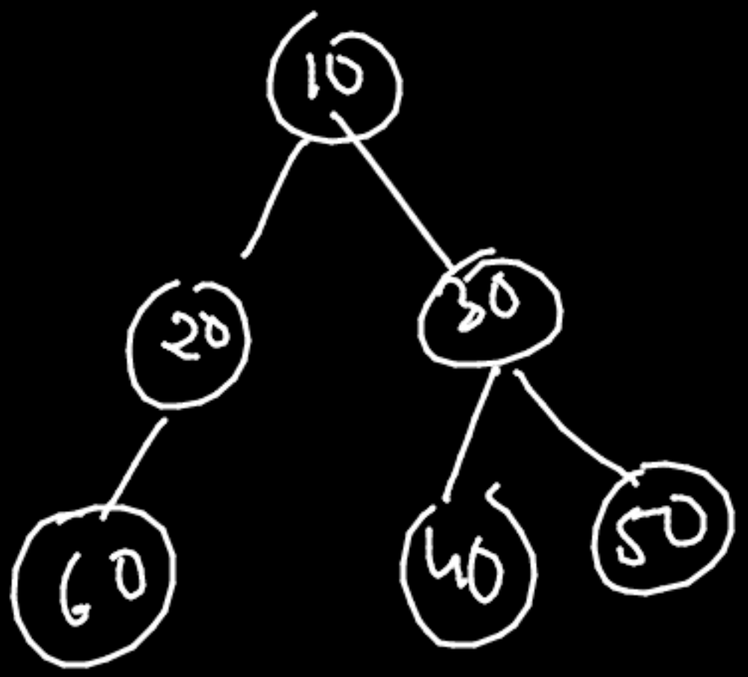

```
int height (BinaryTreeNode<int> *root){
    if(root == NULL)
        return 0;

    int l = height(root->left);
    int r = height(root->right);

    return 1 + max(l,r);
}
```

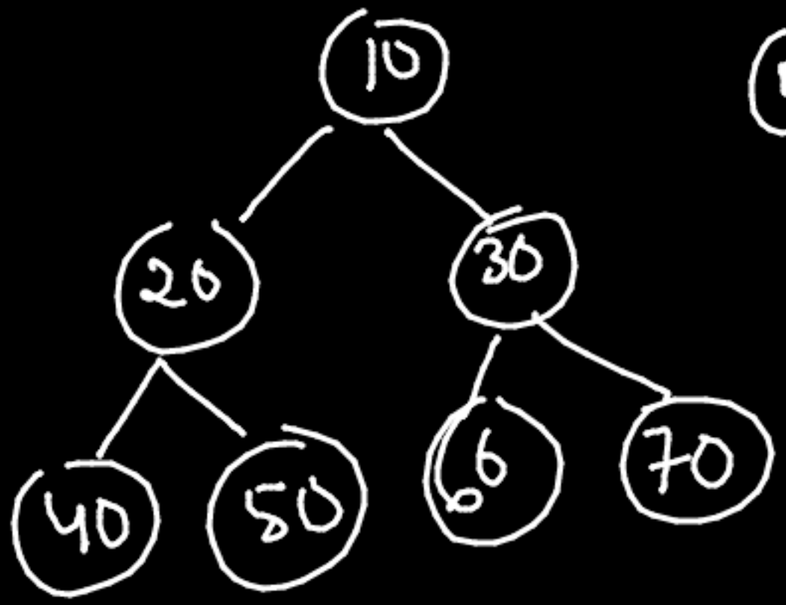
Sum of Node

(Basic)



1. queue
2. Recursion

Print in level



(1) X Z Y

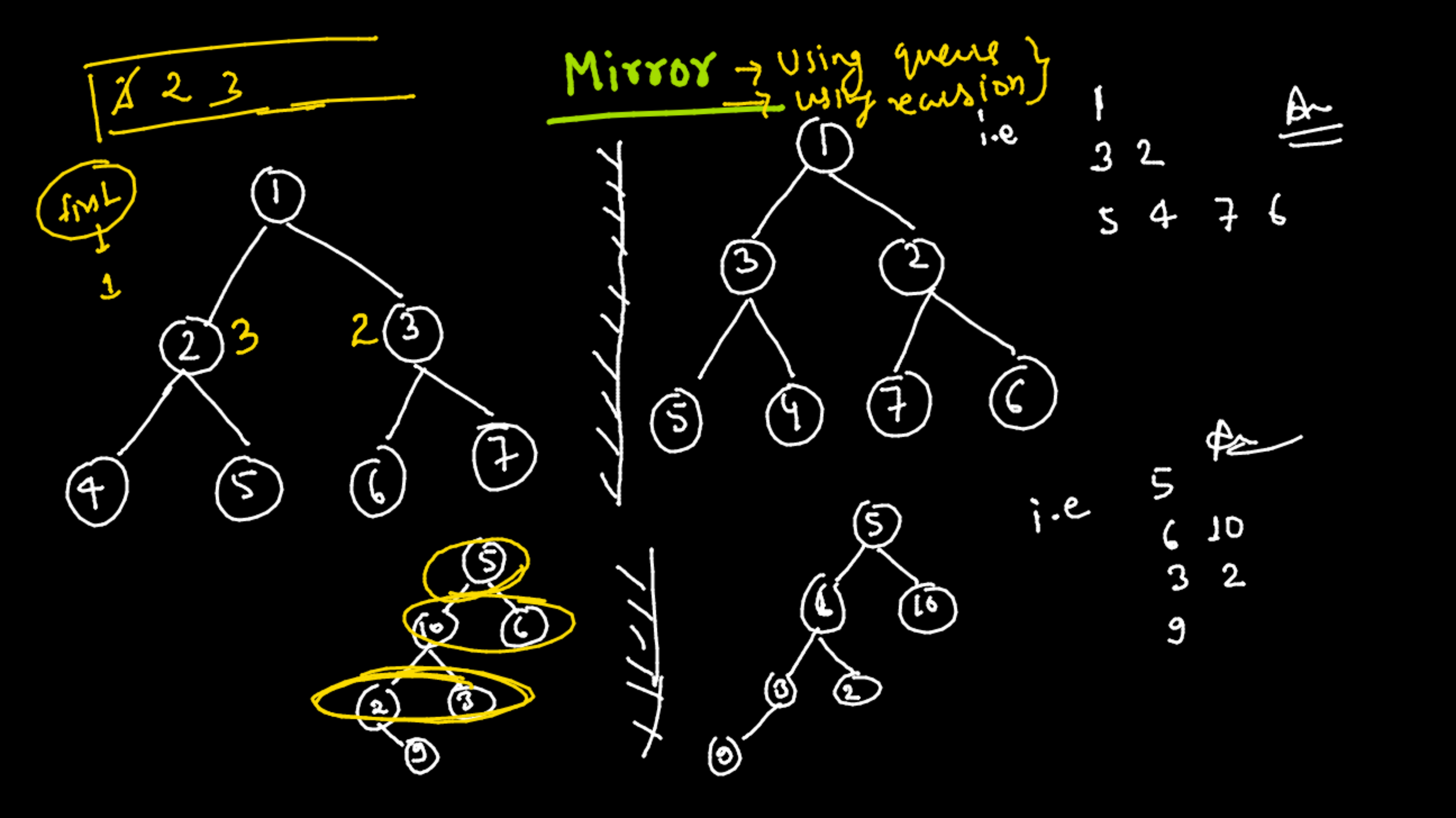
10
→ 20 30
⇒ 40 50

10 20 30 40 50 60 70

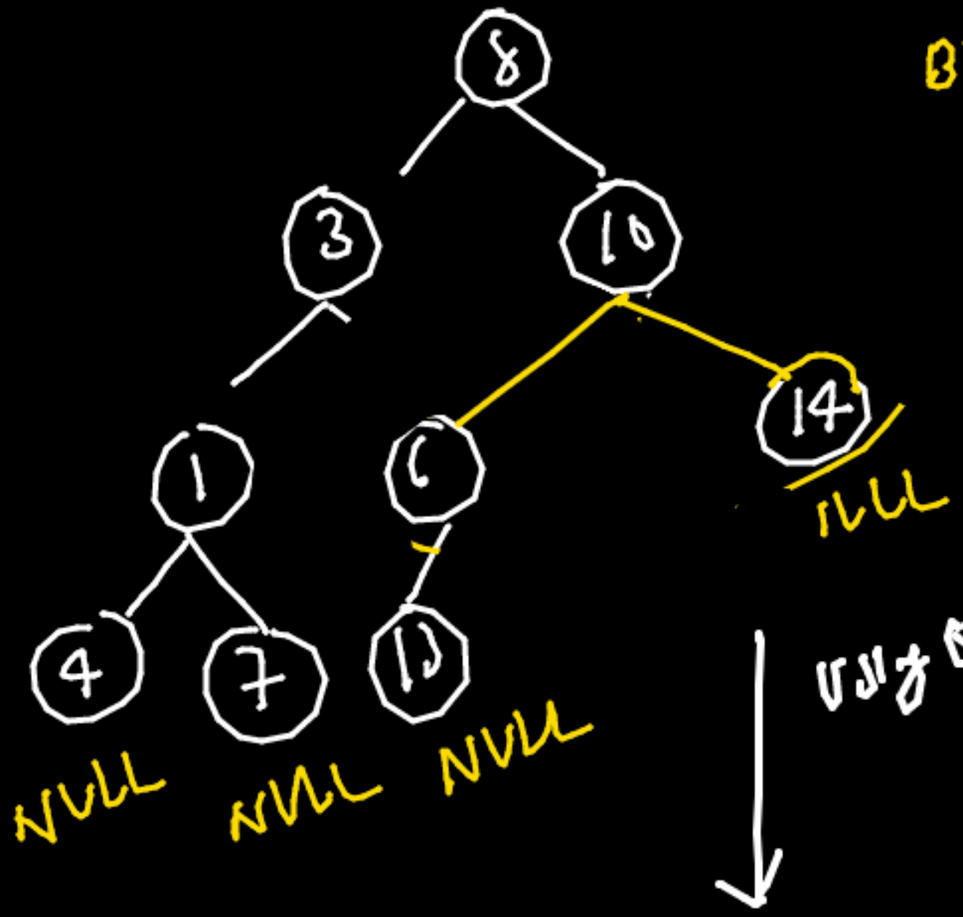
o/p
10
20 30
40 50 60 70

Code

fun → 40 50 60


A 2 3
fmL
1
1
2 3
2 3
4
5
6
7
Mirror → using queue
→ using recursion
i.e
1
3 2
5 4 7 6
An
5
10 6
2 3
9
i.e
5
6 10
3 2
9
An

Remove leaf Node → Using Recursion




```
BT * remove (BT * root) {
    if (root == NULL)
        retur
    if (root->left==NULL && root->right==NULL)
        free (root); return.
```

Using Queue

```
    root->left = reverse (root->left);
    root->right = reverse (root->right)
    return root;
}
```

Construct Tree (Preorder and Inorder)

(root, left, right)

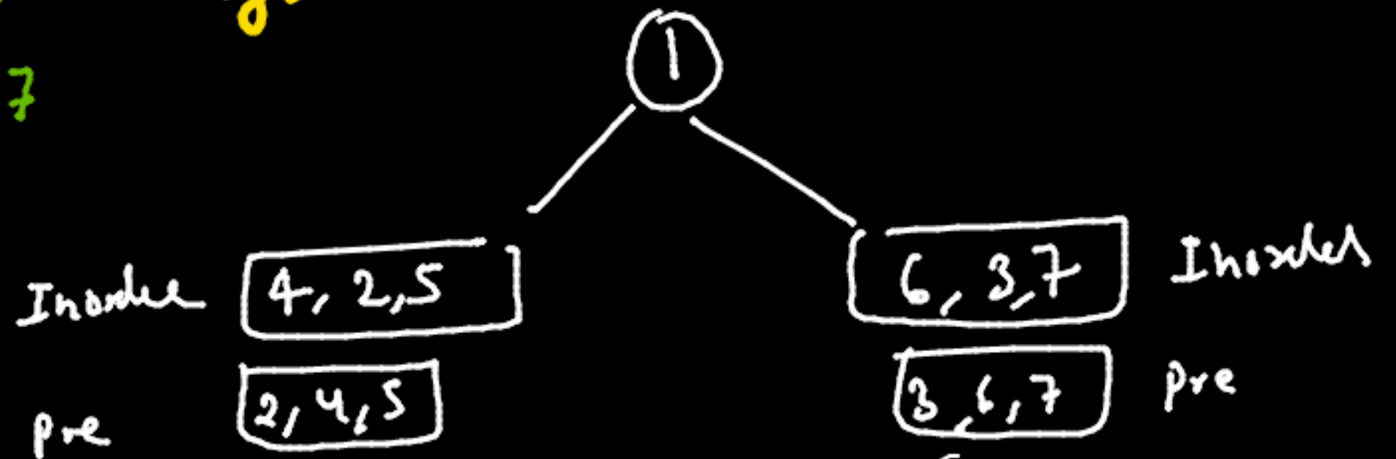
Preorder : 1 2 4 5 3 6 7

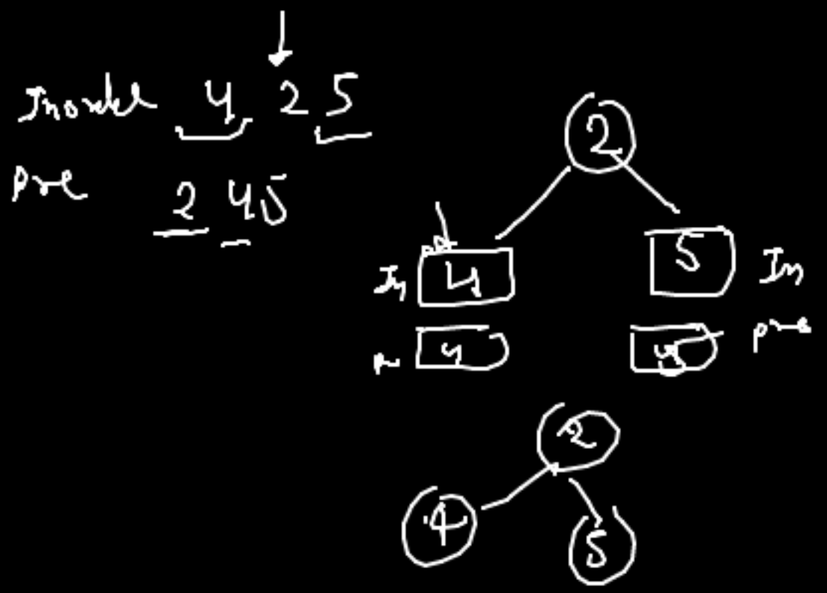
Inorder : 4 2 5 1 6 3 7

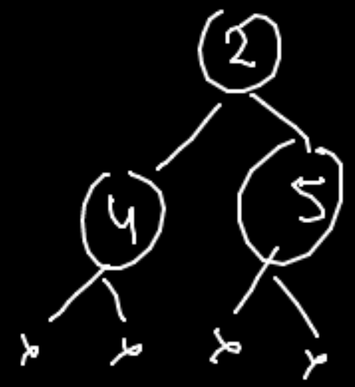
(left, root, right)

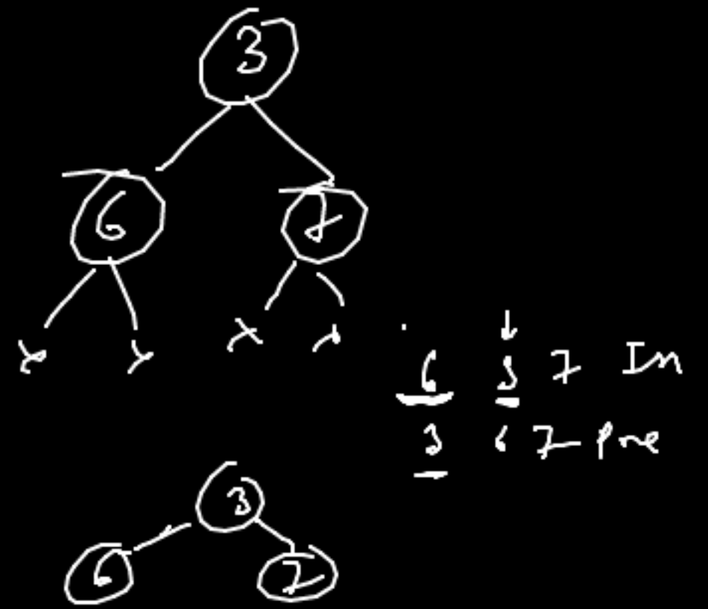
pos

eg1.









root, left, right

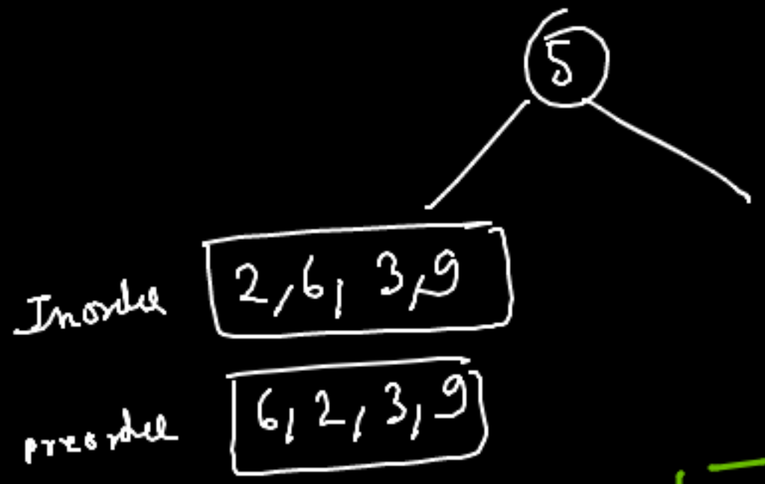
Preorder : 5, 6, 2, 3, 9, 10 ← index

Inorder 2, 6, 3, 9, 5, 10

left, right, root

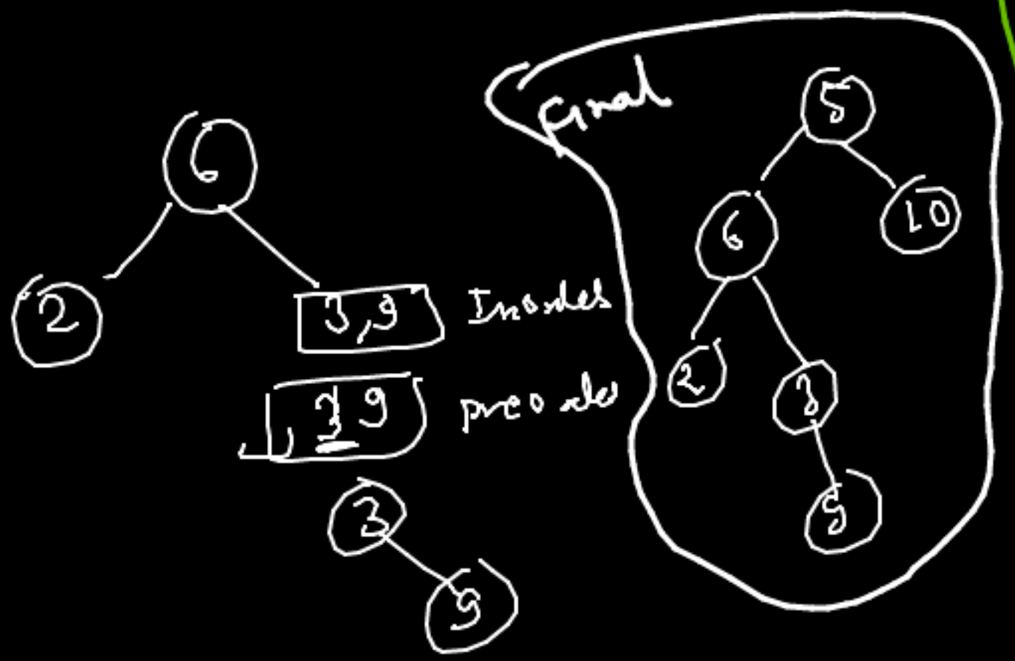
first pos inend

last



Inorder 2,6,3,9

preorder 6,2,3,9



logic

- Index = 0;
- make a root of that index element;
- find index element in Inorder and assign it to position.

recursive call

- left call fun(Instart, pos-1)
- right call fun(pos+1, inEnd)

Construct Tree (inorder & Postorder)

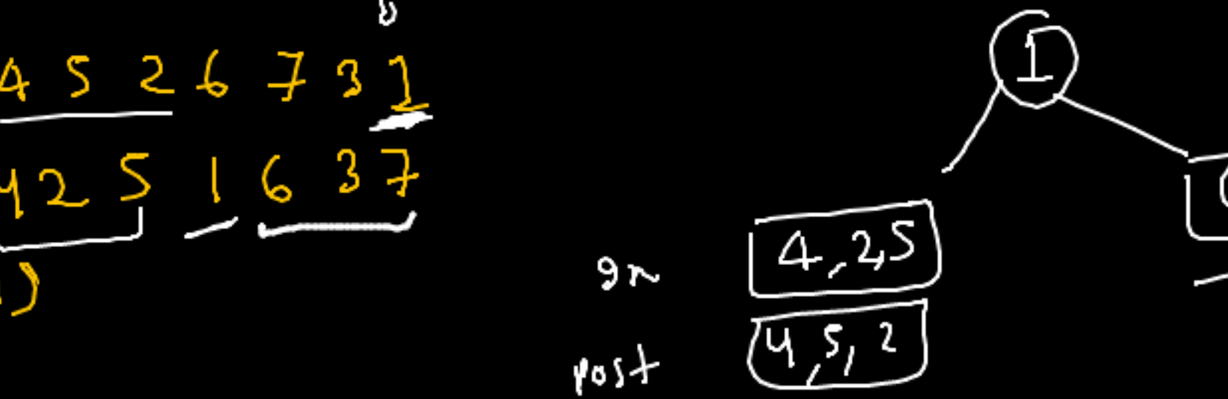
(left, right, root)

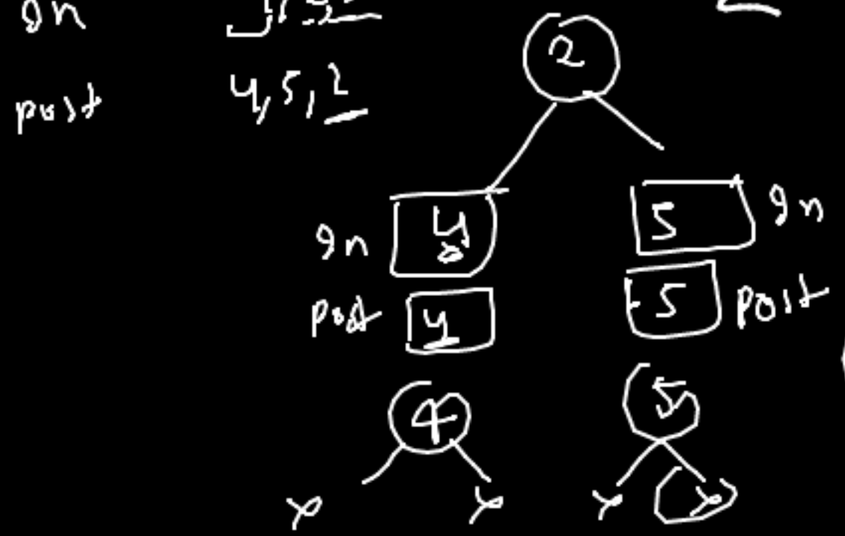
Post order: 4 5 2 6 7 3 1

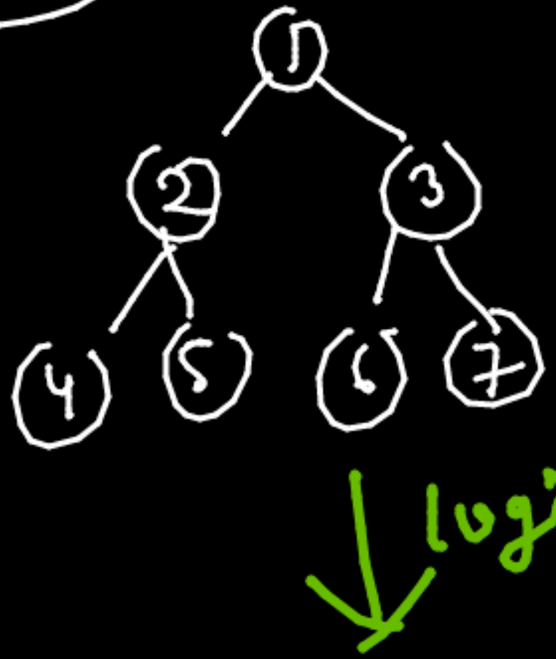
Inorder : 4 2 5 1 6 3 7

(left, root, right)

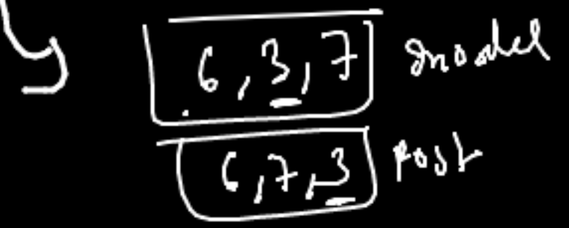
index 6

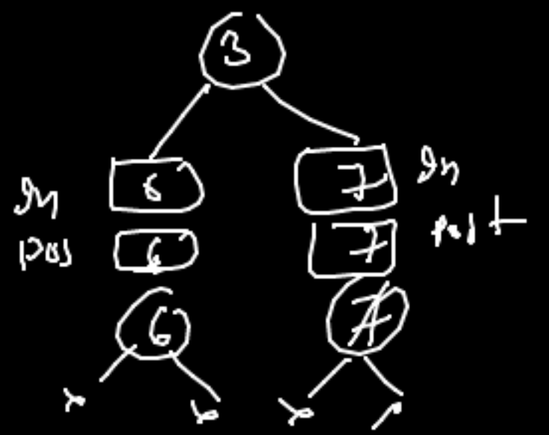






logic





Logic

→ index will point to end of postorder (as that is our root)

→ Index element is our val;

→ make a root and put index element in that.

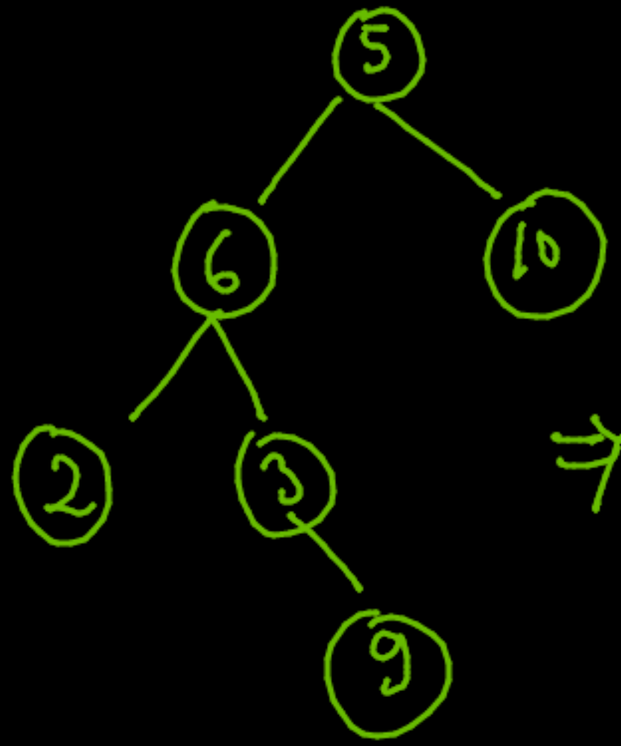
→ Find index element in inorder and put that index in position so, we can divide (&inStart, position-1) to left, and (position+1, inEnd) to right

→ // make recursive call,

✓ root -> right (post+1, inEnd) border dekr prev to index jo found in right of inorder

✓ root -> left (inStart /Pos-1)

Nodes Without Siblings



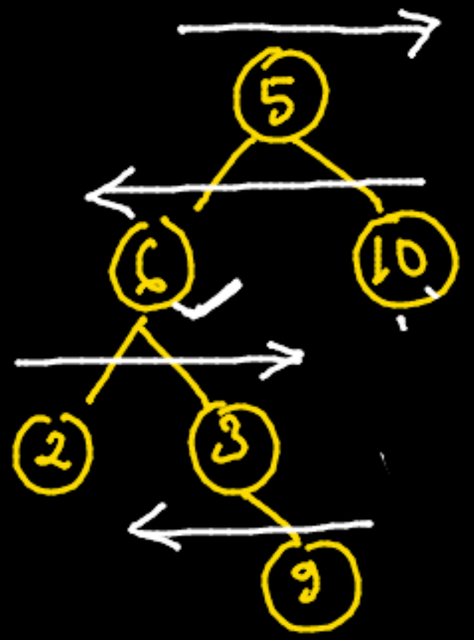

```
if (root == NULL) return;
if (root->left == NULL && root->right != NULL
    cout << root->right->data

if (root->left != NULL && root->right == NULL
    cout << root->left->data

fun(root->left);
fun(root->right);
```

⇒ Ans 9

Zig Zag traversal



solve actually as level wise
just inside while (q.size()!=0)
put vector and push the el

5

10 6

2 3 12 15

9

first → 5

if (lev % 2 == 0) → print vector
else reverse vector then print