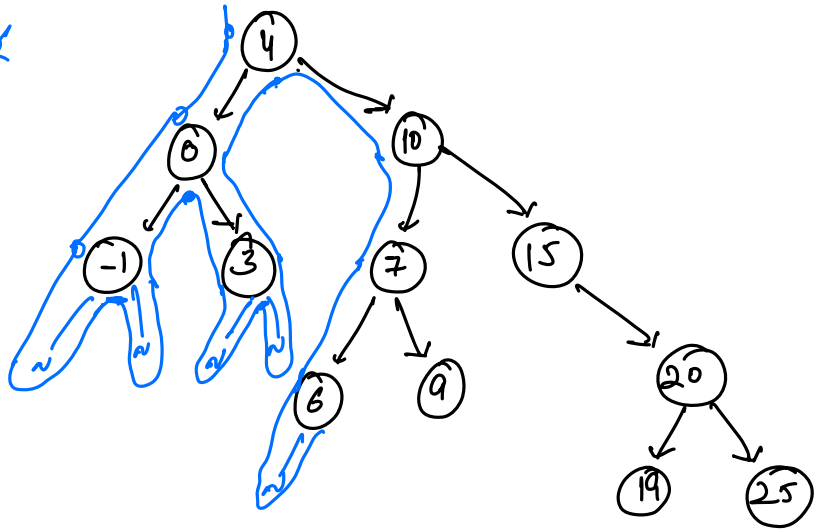
Today's content

- ✓ K^{th} Smallest Element in B.S.T
- ✓ In-order traversal of Binary tree [S.C → O(1)]
- ✓ Find an element in Binary tree
- ✓ L.C.A of Binary Tree
- ✓ L.C.A of Binary Search Tree

① Find K^{th} smallest element in B.S.T.

count of 2 BST's



K=3 ans = 3

K=5 ans = 6

K=8 ans = 10

[-1, 0, 3, 4, 6, 7, 9, 10, 15, 19, 20, 25]

idea-1. Do inorder traversal & store the elements in an array → return arr[K-1]

T.C → O(N), S.C → O(N)

idea-2. Maintain the count of nodes which have been traversed in in-order.

```
count=0 , int ans = -1
void traversal( root, K){
    if (root == NULL) { return }
    traversal( root.left, K);
      count ++;
    if( count == K) { ans = root.val ; }
    traversal (root.right, K);
}
```

T.C → O(N)
S.C → O(Ht.)

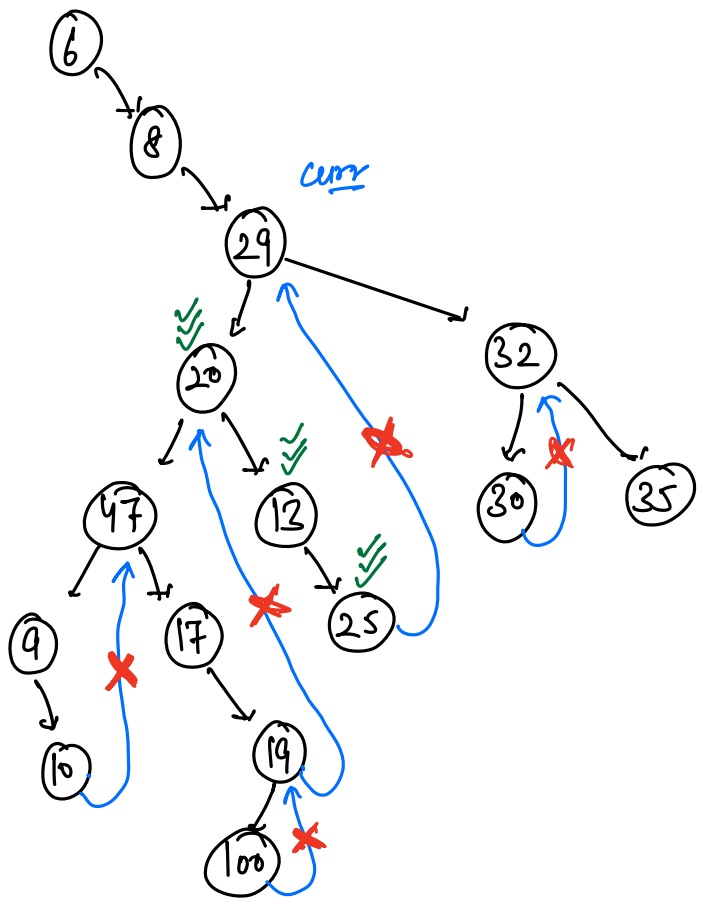
In-Order Traversal Of a Binary Tree

S.C → O(1)

Recursion ? X

iterative ✓

we have to use existing node references to come back / return.



in-order predecessor → of a node doesn't have the right child.

in-order [6, 8, 9, 10, 47, 17, 100, 19, 20, 13, 25, 29, 30, 32, 35]

```
temp = curr.left;
while( temp.right != null){
    temp = temp.right;
}
temp.right = curr;
```

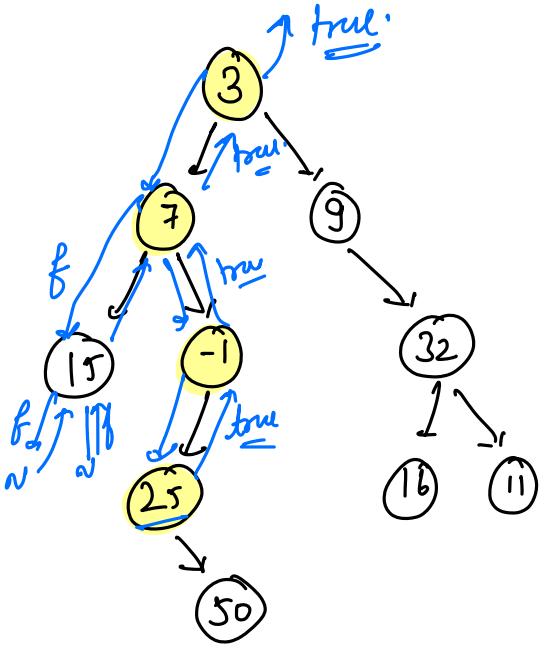
Morris Traversal

code →

```
Node curr = root;
while ( curr != NULL) {
    if (curr.left == NULL){
        print(curr.val);
        curr = curr.right;
    }
    else{
        //Find in-order predeccessor
        Node temp = curr.left;
        while( temp.right !=NULL && temp.right != curr){
            temp = temp.right;
        }

        if ( temp.right == NULL){
            //Creak the connection
            temp.right = curr;
            curr = curr.left;
        }
        else if ( temp.right == curr){
            //break the connection
            temp.right = NULL;
            print( curr.val);
            curr = curr.right;
        }
    }
}
```

T.C → O(N)
S.C → O(1)

→ Find an element in Binary Tree →



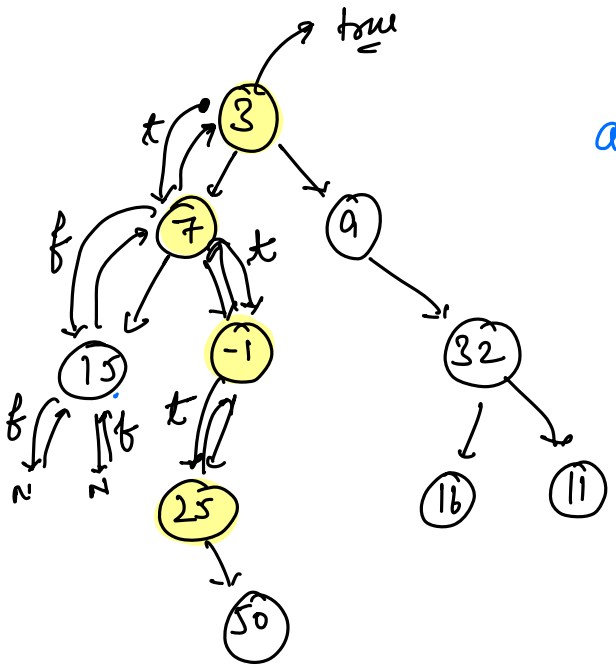
search(25) → true

code →

```
boolean search( Node root, int K){
    if (root == NULL) { return false }
    if (root.val == K) { return true }

    return search(root.left, K) || search (root.right, K);
}
```

T.C → O(N)
S.C → O(Ht)

Path from root to node →



ans → [3, 7, -1, 25]

↑ Reverse

[25, -1, 7, 3]

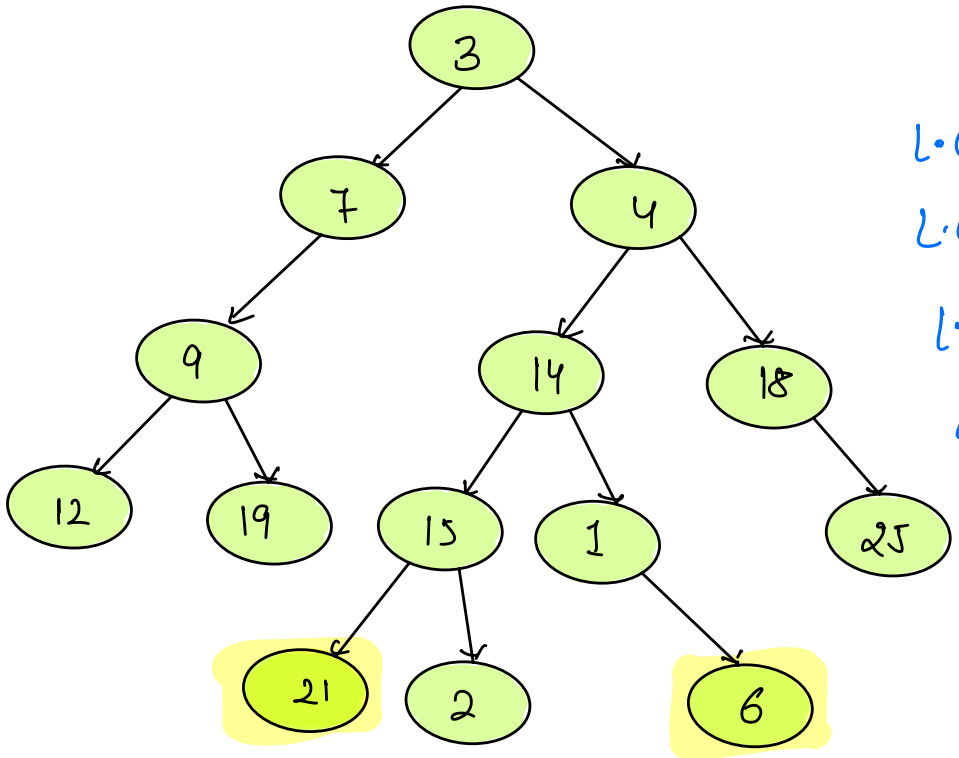
```
boolean search ( root, k, list<integer> ans){
    if (root == NULL) { return false}
    if ( root.val == k){
            ans.insert(root.val);
            return true;
    }

    boolean rr = search(root.left, k, ans) || search(root.right, k, ans);
    if ( rr == true){
            ans.insert( root.val);
            return true;
    }
    return false;
}
```

T.C → O(N)
S.C → O(Ht.)

// Reverse the list to get desired answer.

Lowest Common Ancestor →

x, y



L·C·A(21, 6) = 14

L·C·A (2, 6) = 14

L·C·A(21, 4) = 4

L·C·A(12, 6) = 3

21 → [21, 15, 14, 4, 3]

6 → [6, 1, 14, 4, 3]

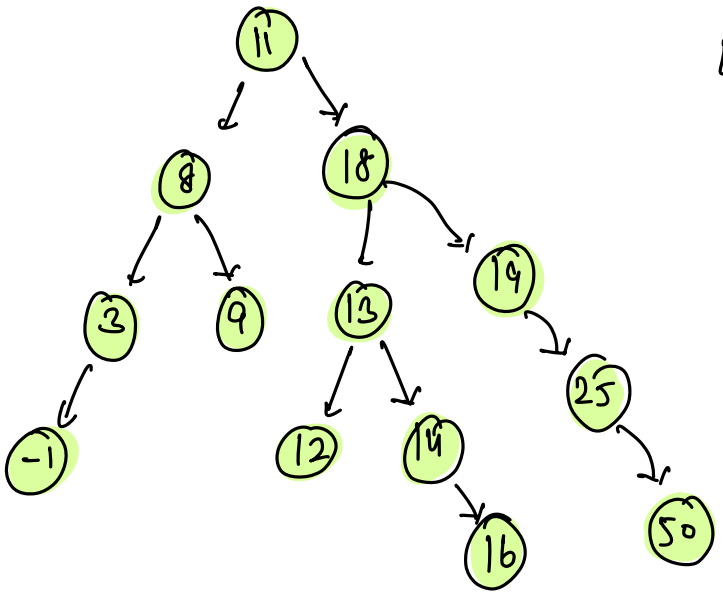
code →

```
list<int> a1, a2;
search(root, x, a1);
search(root, y, a2);
i = a1.size()-1 , j = a2.size() - 1
```

T.C → O(N)
S.C → O(Ht.)

do it yourself.

L.C.A in B.S.T →



LCA(12, 50) ⇒ 18

LCA(12, 16) ⇒ 13

x, y are present in B.S.T.

Code. →

```
Node curr = root;
while( true ){
    if( x < curr.val && y < curr.val ){
        curr = curr.left;
    }
    else if( x > curr.val && y > curr.val){
        curr = curr.right;
    }
    else{
        break;
    }
}
return curr.val;
```

T.C → O(Ht.)
S.C → O(1)