Trees 4 : LCA + Morris Inorder Traversal

Agenda

- K^{th} smallest element in BST
- Morris inorder traversal
- LCA in binary tree and BST
- Recover BST

Question

Given a BST, find K^{th} smallest element.

We know that, inorder traversal of BST is sorted.

Idea 1 : do inorder traversal & store element in array, then return $(K-1)^{th}$ element.

$TC = O(N)$ $SC = O(N)$

Idea 2 : instead of storing elements, keep track of count of elements. When counter hits K, return that value.

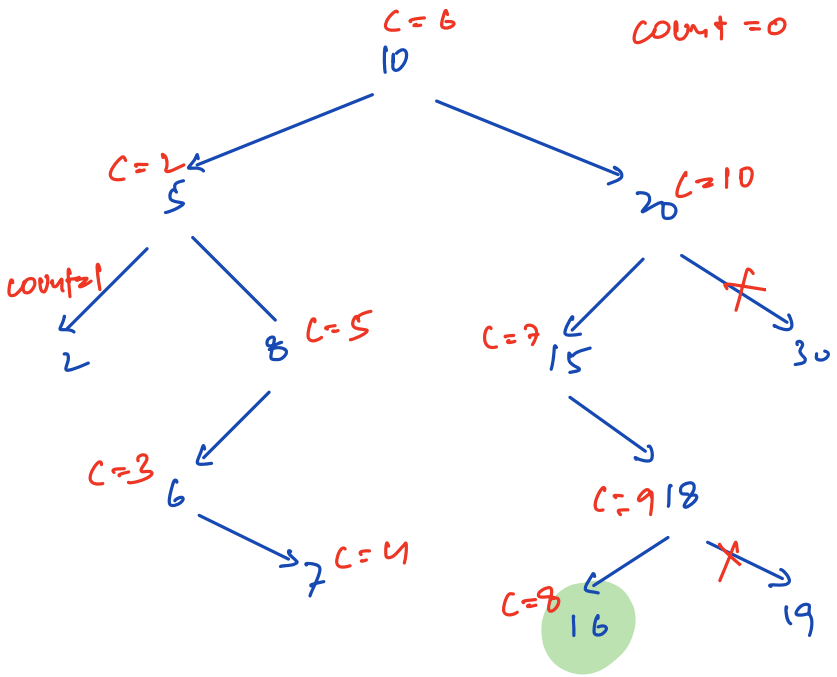
code

```
count = 0
ans = INT_MIN
void inorder ( root , K ) {
    if ( !root ) return;
    inorder ( root->left, K);
    count++;
    if ( count == K ) {
        ans = root->data;
        return;
    }
    if( ans == INT_MIN){
        inorder ( root->right, K);
    }
}
```

TC = O(N)

SC = O(height)

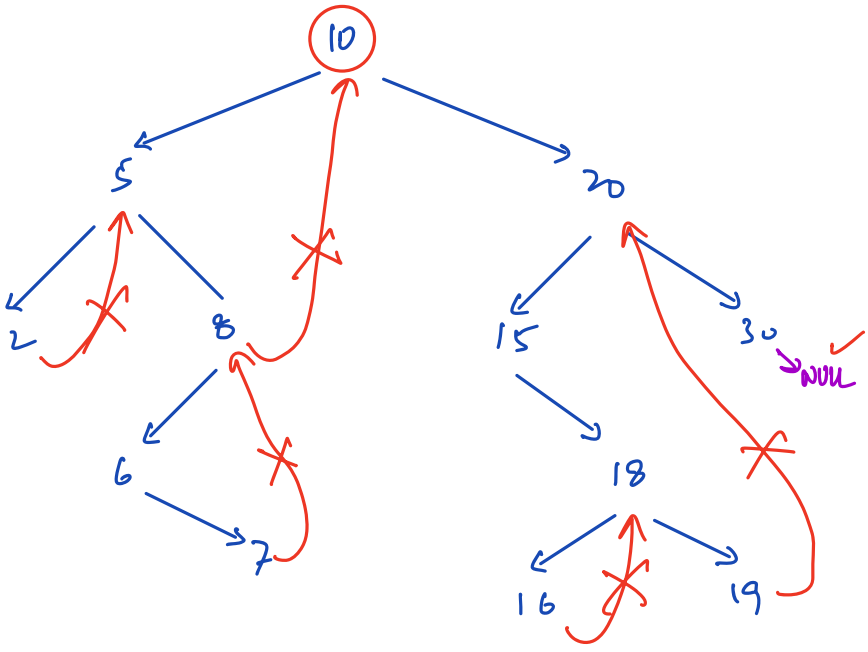
After inorder completes, "ans" will hold K^{th} smallest element in BST.



K=8

If I want to achieve O(1) space ?

Morris Inorder Traversal



Where to store current node when we go left ?

After which node, the current node is required?

Max element in left subtree

↳ inorder predecessor

right-most element in left subtree

2 5 6 7 8 10 15 16 18 19 20 30

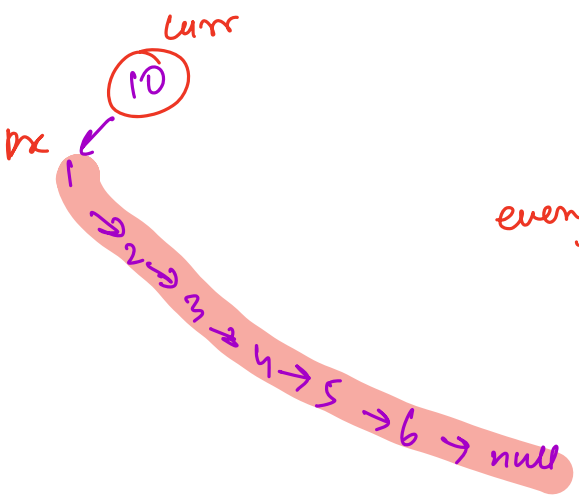
Code

```
curr = root
while ( curr != null) {
    if ( curr.left == null) {
        print ( curr.data)
        curr = curr.right
    }
    else {
        pre = curr.left
```

```
        while ( pre.right != null && pre.right != curr) {
            pre = pre.right
        }
        if ( pre.right == null) {
            pre.right = curr  // create link
            curr = curr.left
        }
        else {
            pre.right = null  // remove link
            print ( curr.data)
            curr = curr.right
        }
    }
}
```

SC = O(1)

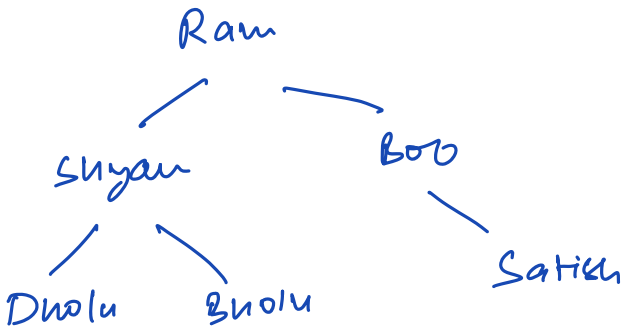
TC = O(N)



every node is visited 3 times

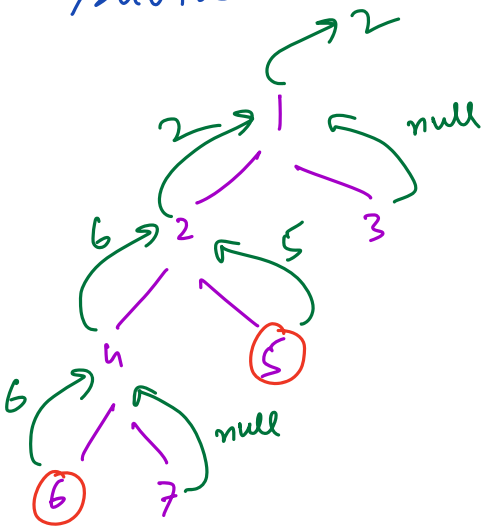
TC = O(N)

Common Ancestor



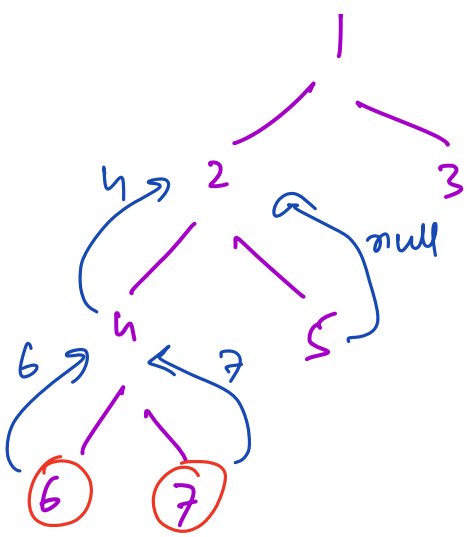
Lowest common Ancestor (LCA)

LCA of 2 nodes is the deepest node in the tree where the two nodes lies in left & right subtree.



Code

```
Node findLCA(root, node1, node2){

    if(root == null || root == node1 || root == node2){

        return root

    }

    leftLCA = findLCA(root.left, node1, node2)

    rightLCA = findLCA(root.right, node1, node2)

    if(leftLCA != null && rightLCA != null){

        return root  // current root is LCA

    }

    if(leftLCA != null)

        return leftLCA

    else

        return rightLCA

}
```



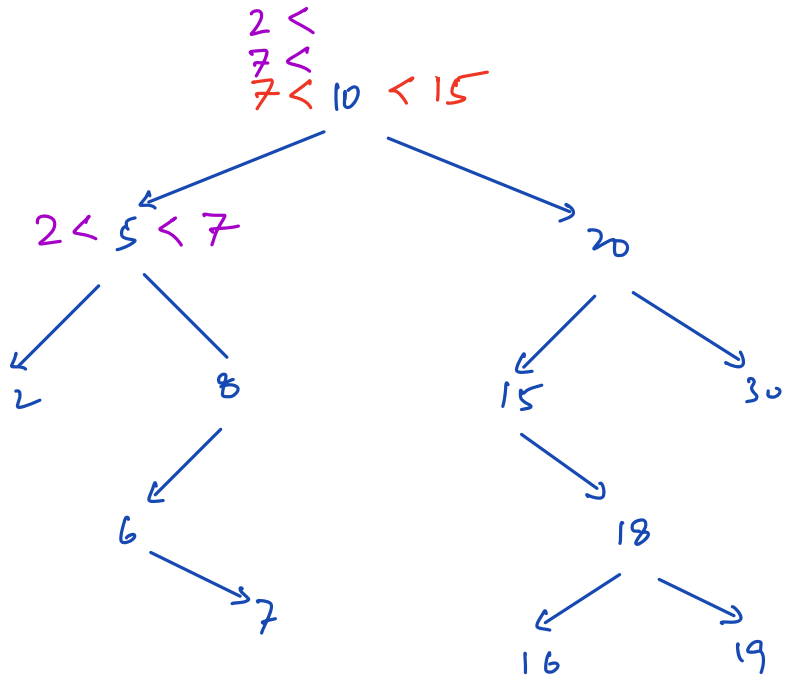
TC = O(N)

SC = O(H)

find LCA in BST

LCA(7, 15) = 10

LCA(2, 7) = 5



Code

```
curr = root
while ( curr != null) {
    if( x < curr.data && y < curr.data) {
        curr = curr.left
    }
    else if( x > curr.data && y > curr.data){
        curr = curr.right
    }
    else {
        return curr;
    }
```

TC = O(H)

SC = O(1)

}

return null

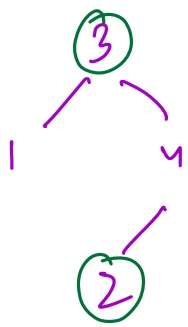
Question

In BST, inorder traversal is sorted.

If 2 nodes are swapped by mistake, BST property is violated.

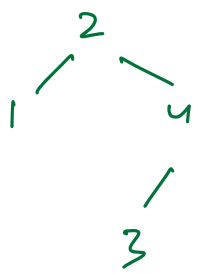
The goal is to recover the BST by swapping 2 misplaced nodes back to their correct position.

eg



inorder : 1 3 2 4

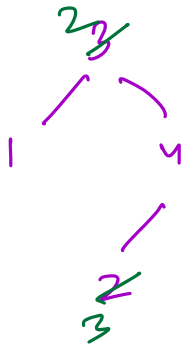
o/p :



→ 1 2 3 4

Idea 1:

1. Do inorder traversal & store the nodes in list
2. Sort the list
3. Perform inorder again & replace node values with the corresponding value in the sorted list.



inorder : 1 3 2 4

↓ sort

1 2 3 4

Code

```
ArrayList<int> vals;

void inorder (root) {
    if(!root) return
    inorder(root.left)
    vals.push(root.data)
    inorder(root.right)
}
```

Sort(vals)

```
void recover( root, index ) {
    if(!root)  return
    recover( root.left, index)
     root.data = vals[index]
     index++
     recover( root.right, index)
}
```

TC = O(N + NlogN + N) = O(NlogN)

- N → 1st inorder
- NlogN → sort
- N → 2nd inorder

SC = O(H + N) : O(N)

- N → storing data

Optimization

Since there is only 1 swapping happened, there will be one or two violations in the inorder sorted order.

Code

```
Node first = NULL
Node second = NULL
Node prev = NULL

void inorder( root ){
    if(!root) return
    inorder( root.left)
    if( prev && prev.data > root.data ){
        if( !first) first = prev
        second = root
    }
    prev = root
    inorder( root.right)
}

swap( first.data, second.data)
```

bool a = false

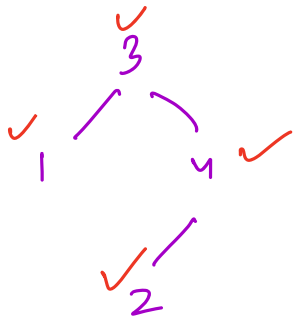
if (a == false) ⇒ if (!a)

TC = O(N)

SC = O(N)

↓

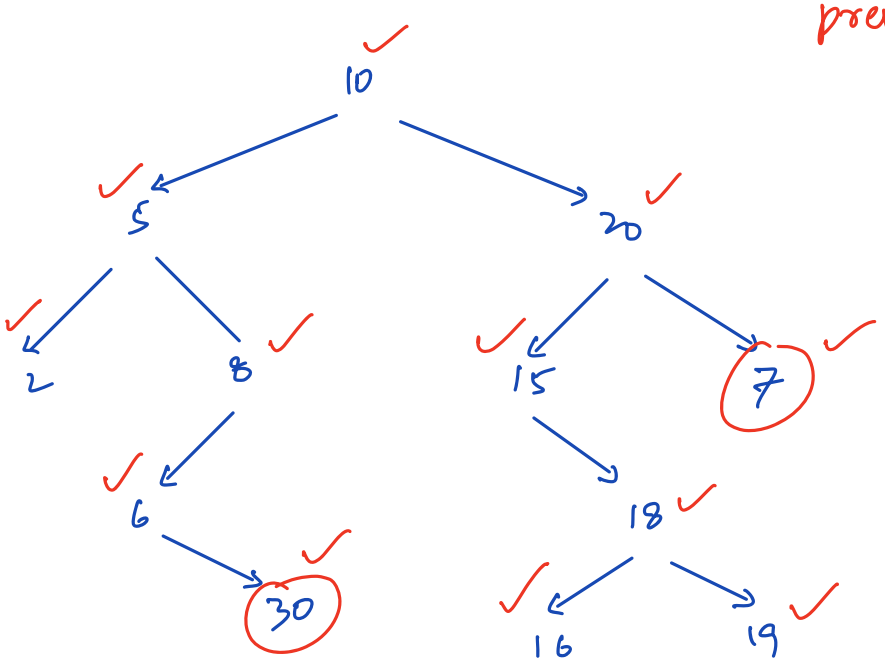
O(1) using morris inorder



prev = ~~X~~ 2

first = 3

second = 2



prev: null 2 5 6 30 8 10 15 16 18 19 20

first = 30

second = 8 7