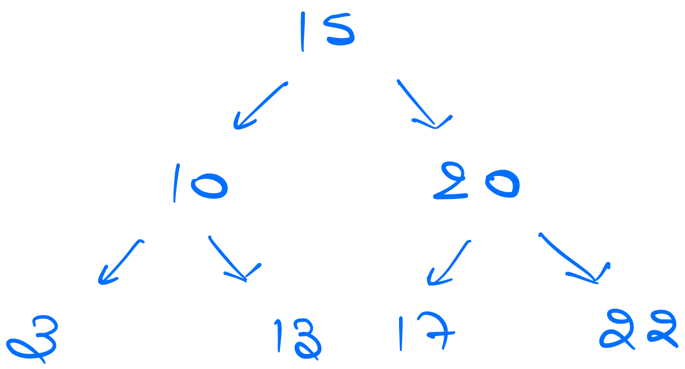
Agenda:

1) Check if a Binary Tree is a BST.
2) LCA of 2 Nodes.
3) LCA in BST
4) InTime / Out Time.

Q Check if a given Binary Tree is a BST.



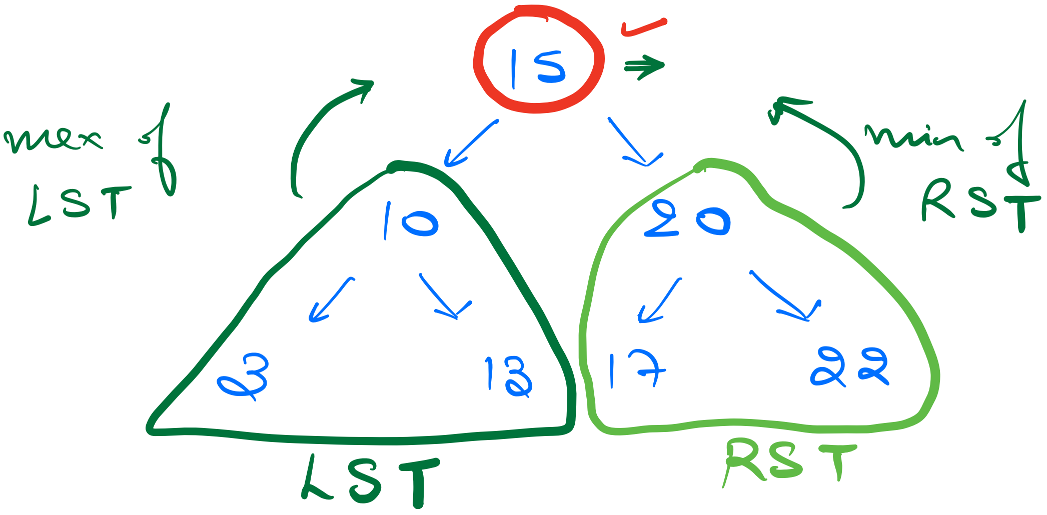
Approach 1 : (In Case of Unique Elements)

If the inorder traversal of the Tree is sorted it is a BST.

Approach 2 :

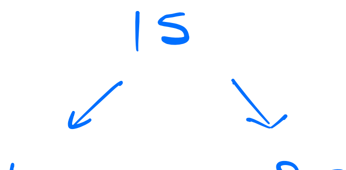
∀ nodes check : 1) node.data ≥ x.data
∀ nodes x in the LST

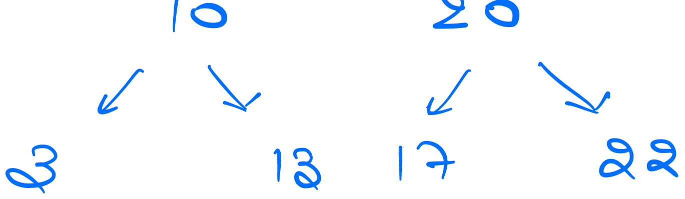
2) node.data < y.data
∀ nodes y in the RST



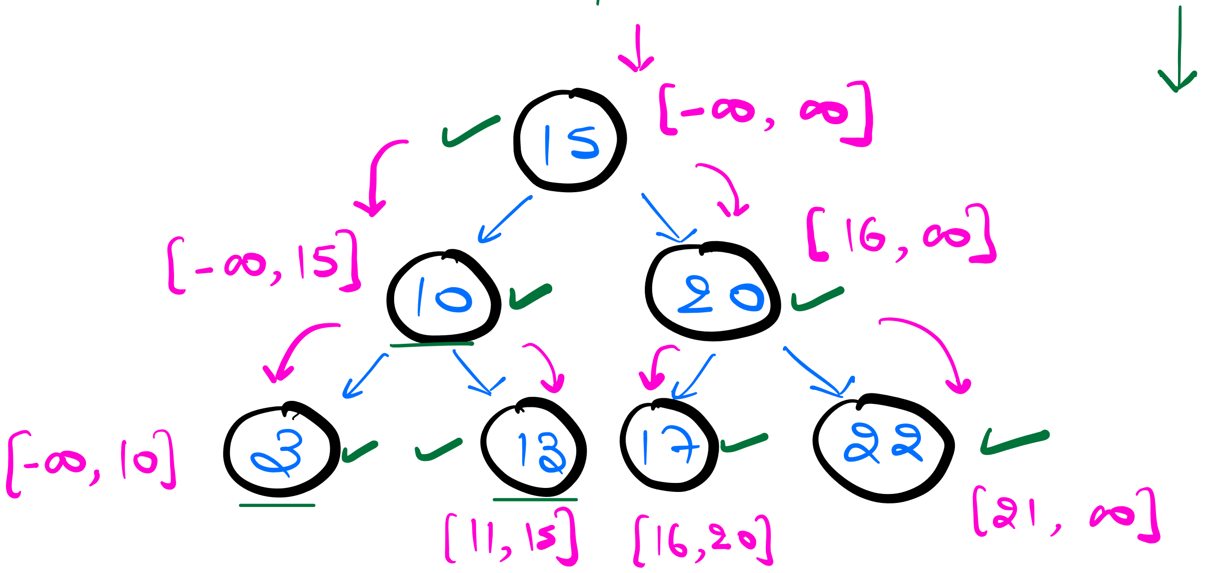
Pre Order not optimal as the flow of info. (req. info) is coming from bottom (LST & RST) for all nodes.

Approach 3: Post Order $(\bar{L}\ \bar{R}\ \underset{\uparrow}{\underline{N}})$





Approach 4: Optimal Pre-Order



Pre Order : information is passed as method arguement

Post Order : information is passed a method return value.

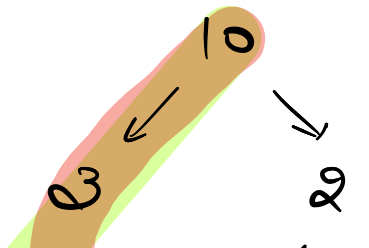
Code

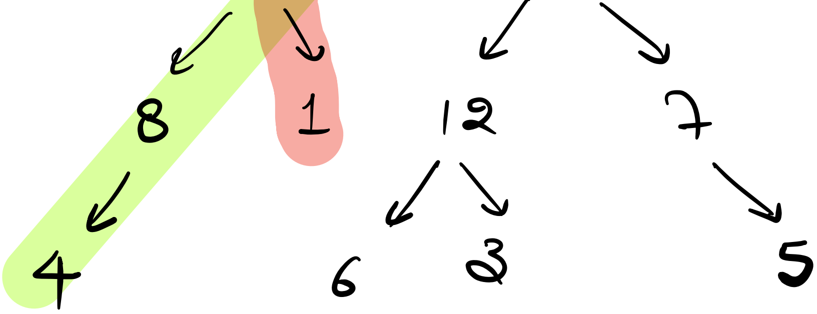
INT_MAX INT_MIN.

↑

```
bool isBST( root, maxValue, minValue){
        if (root == NULL){
                return True;
        }
        if ( root.data ≥ minValue &&
                root.data ≤ maxValue){

            if ( isBST( root.left, root.data,
                    minValue) && isBST(
            root.right, maxValue, root.data+1){
                    return True;
            } else {
                return false;
            }
    }
    return false;
}
```

LCA: Lowest Common Ancestor.





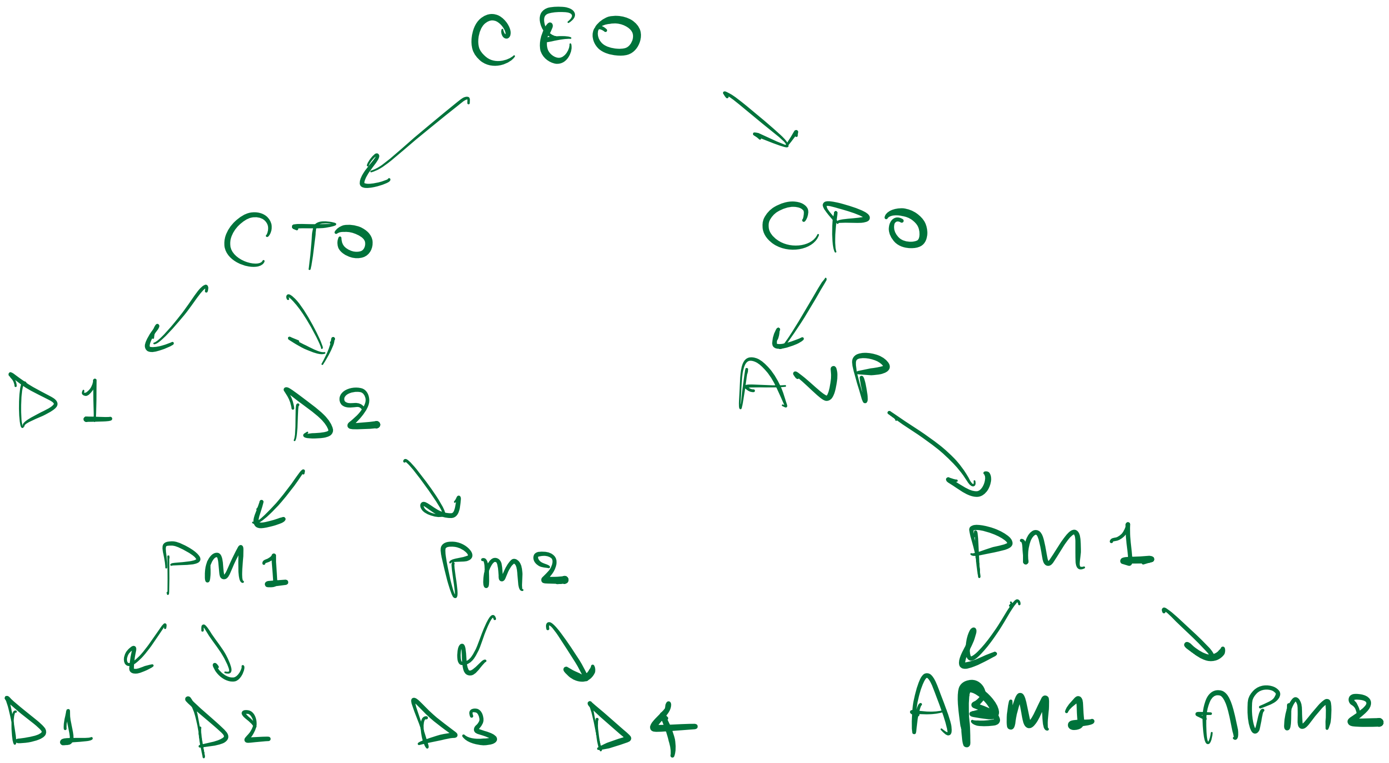
LCA of 6 & 5 = 2

Ancestors

1) 4 : 10, 3, 8, 4

2) 1 : 10, 3, 1

LCA of 4 & 1



Sol^n

Given 2 nodes u & v.

find the path

1) root to u
2) root to v

⇒ find the last common node b/w both the paths.

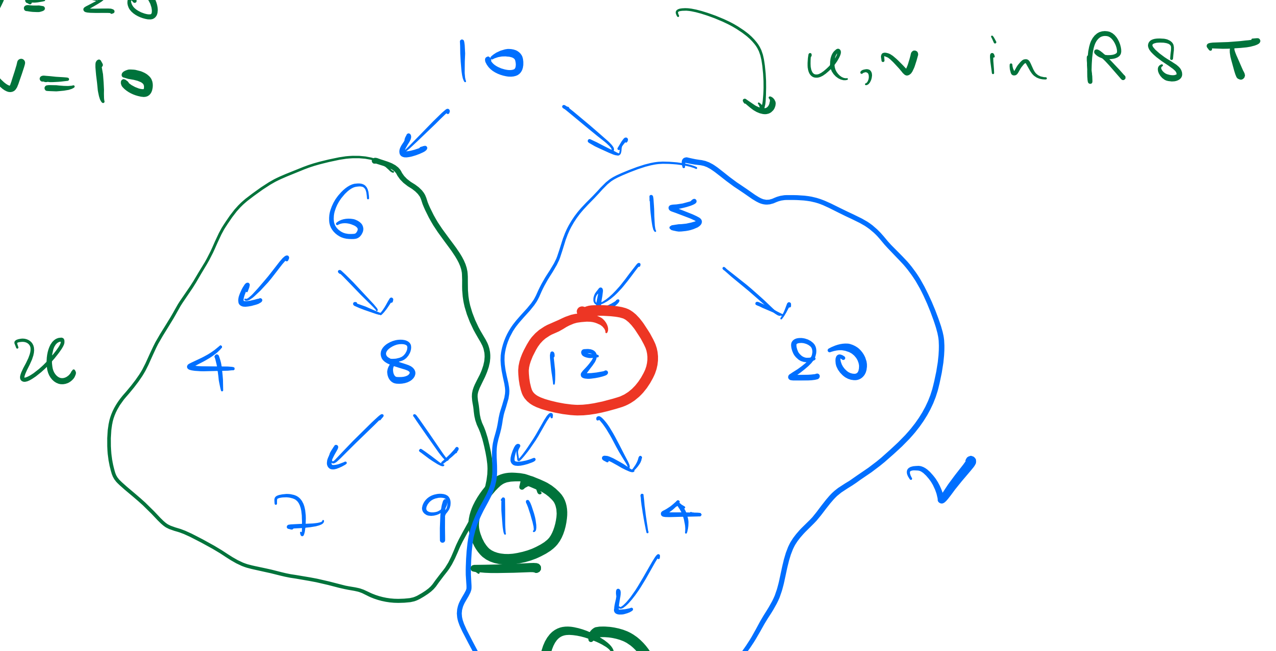
Q

Given a BST & 2 nodes u & v.

find the LCA of u & v. (Unique no's)

$u = 10, v = 20$

$u = 8, v = 10$



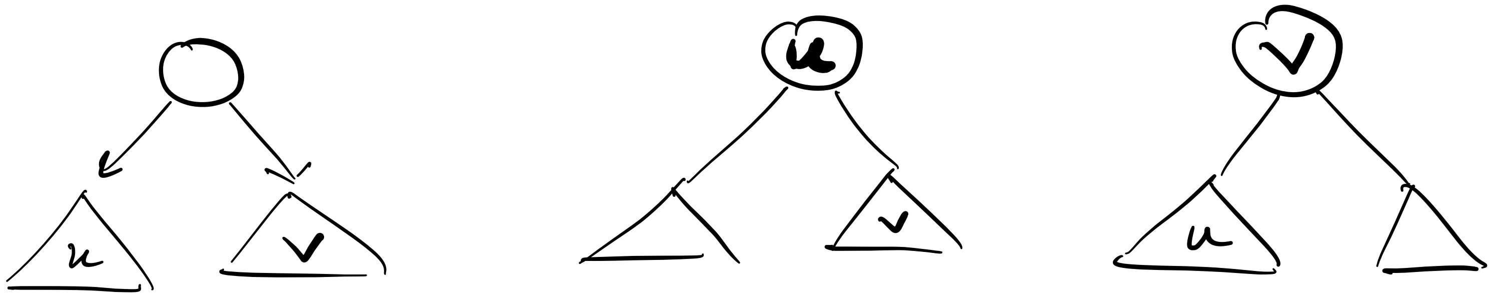
LCA of 11 & 13 : 12

Q For 2 nodes u & v where u.data < v.data

When can a root node be the LCA of u & v.

Sol^n Assume u.data < v.data.

Case I ⇒ When root is LCA



Case II When both u & v present in LST

⇒ LCA is also present in LST

Case III when both u & v present in RST

LCA is also present in RST

Code

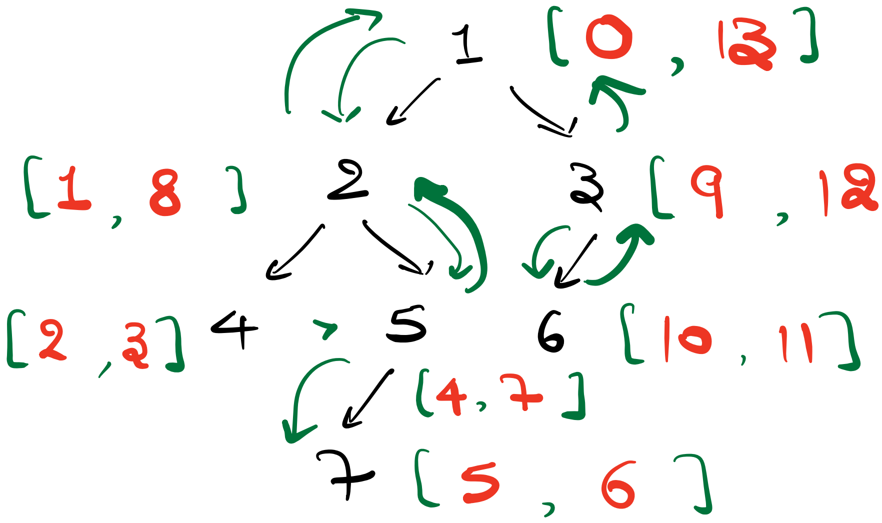
```
Node findLCA (root, u, v) {
    if (root == NULL) {
            return NULL;
    }
    Node curr = root;

    while (curr != NULL) {
        if (u.data < curr.data && v.data <
                curr.data ) {
                curr = curr.left;
        } else if(u.data > curr.data && v.data > curr.data){
                curr = curr.right;
        } else {
            return curr;
        }
    }
    return NULL;
}
```

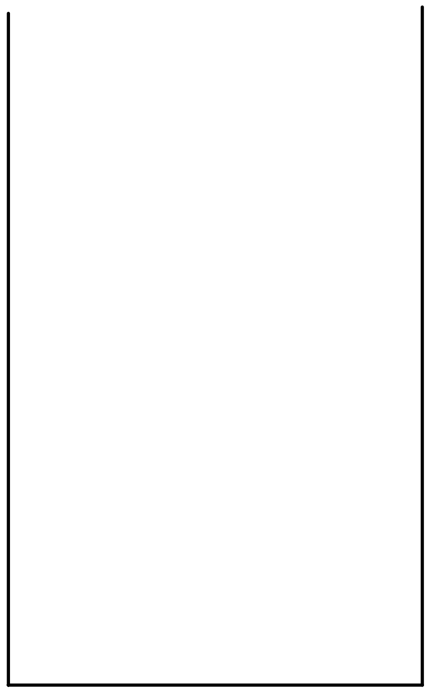
T.C. = O(Height) = O(N)

S.C. = $\underline{O(1)}$

In Time & Out Time

T = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ ~~12~~ 13





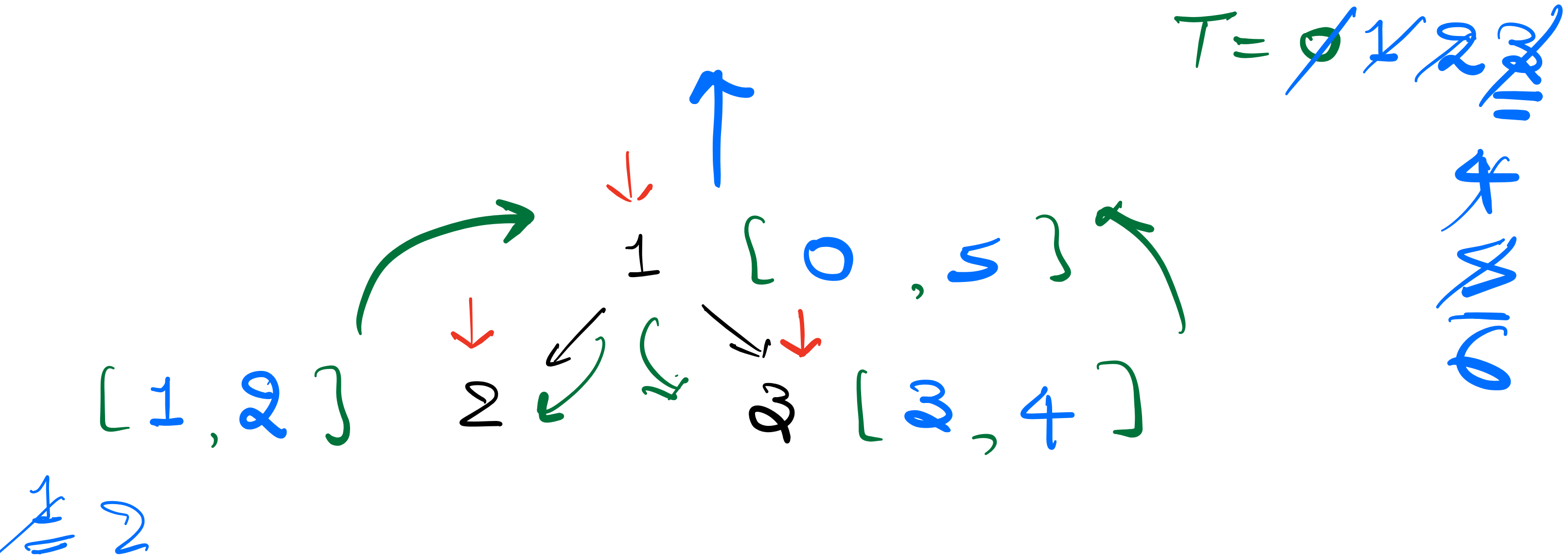
```
void  preOrder ( root) {
        if ( null) {
          print (root);
          preorder (left);
          preOrder (right);
}
```

```
class Node {

        int data;
        int  inTime
        int OutTime;
        Node left;
        Node   right;

        Node ( int x) {
            data = x
            inTime = OutTime = -1;
            left = right = NULL;
      }
}
```

Q Given a Binary Tree. Update the Intime & Outtime for all nodes for any traversal.

```
global Time = 0;
void calculateInOutTime ( root) {

        if (root == NULL) {
                    return;
        }
         dfs (root);
}


void dfs( root) {
Node    root. intime  =  Time;
         Time ++;
        if ( root. left != NULL) {
Left {
                        dfs (root. left);
        }
```

```
Right {  if (root. right != NULL) {
                 dfs (root. right);
         }
Node  root. outTime = Time;
      Time++;
}
```

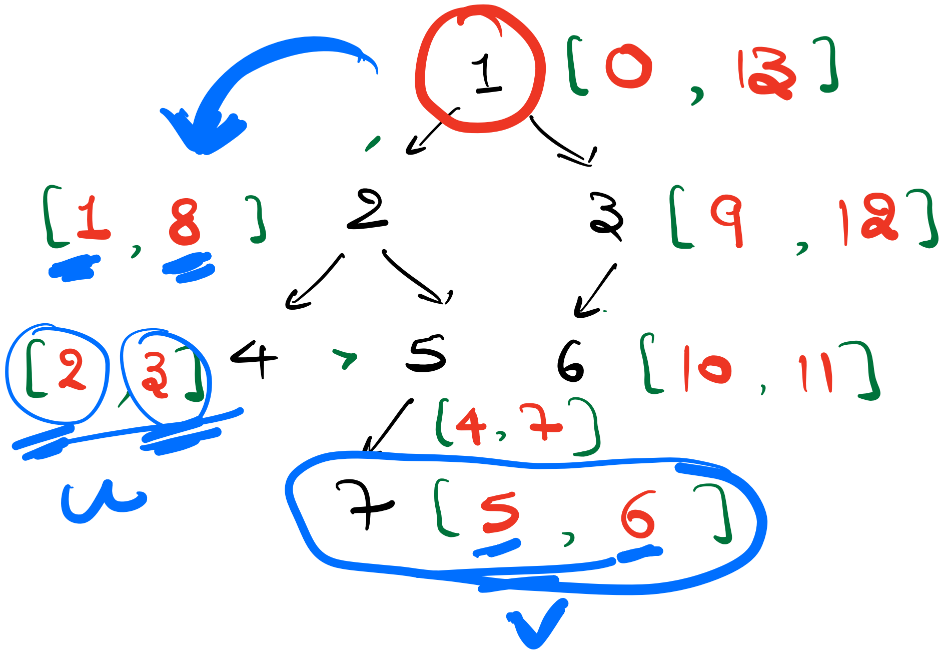


[0, 1]

1

T.C. = O (N)

S.C. = O (Height) = O (N)



Find the lca of 2 nodes

⇑

LCA(4,7) = 2

T.C. = O(Height)

= O(N)

Wednesday ⇒ Complete all Ace Assignment

Doubts in doubt session

u, v.
root

```
if ((u.intime >= root.left.intime) &&
        (u.outtime <= root.left.outtime))
    &&
```