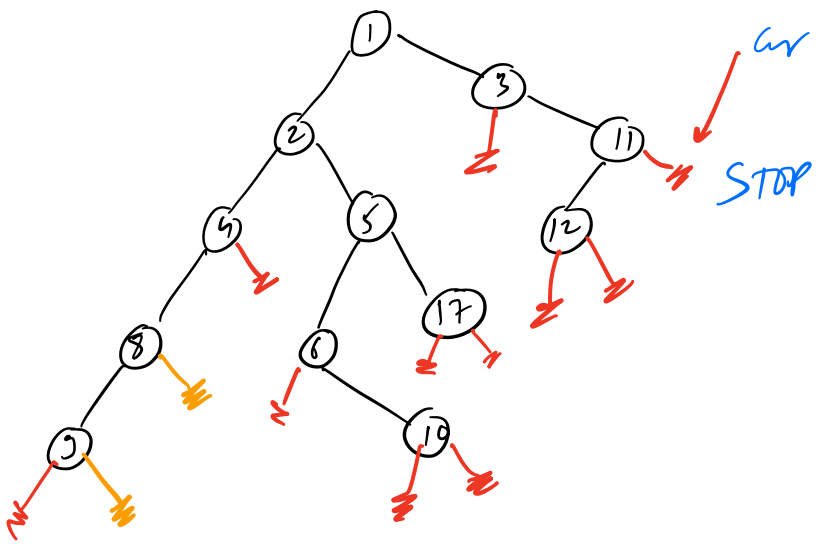
Inorder Traversal

TC → O(N)

SC → O(H)

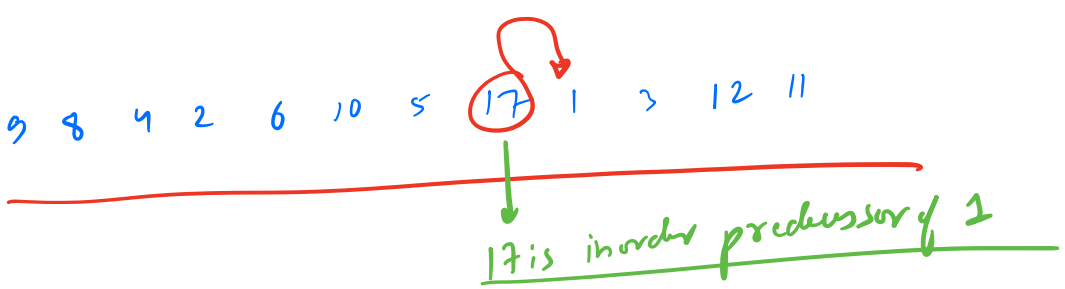
---

⑧ MORRIS INORDER TRAVERSAL [LDR]

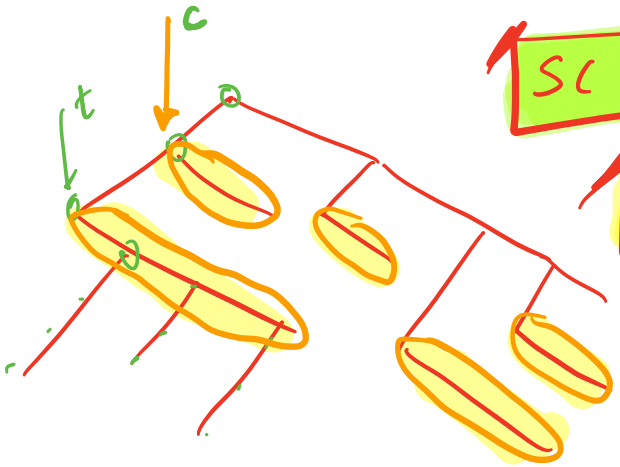


INORDER : 9 8 4 2 6 10 5 17 1 3 12 11

MORRIS



```
void morrisIOT (Node root) {
    Node cur = root;
    while (cur != NULL) {
        if (cur.left != NULL) {
            Node temp = cur.left;
            while (temp.right != NULL && temp.right != cur) {
                temp = temp.right;
            }
            if (temp.right == NULL) {
                temp.right = cur;
                cur = cur.left
            }
            else {
                temp.right = NULL;
                print(cur.data);
                cur = cur.right;
            }
        }
        else {
            print(cur.data);
            cur = cur.right;
        }
    }
```

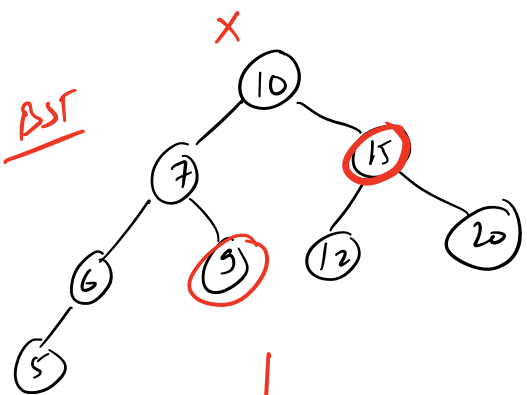


SC = O(1)

TC = O(N)

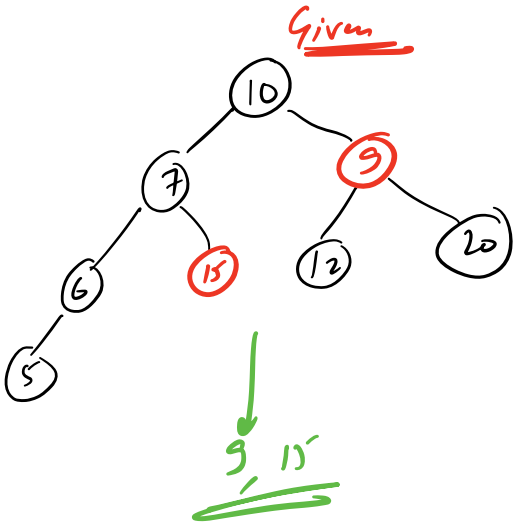
Q Given a BST after 2 of it's nodes have been SWAPPED.

Find those nodes!



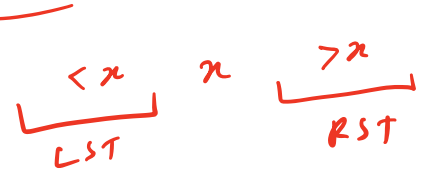
INORDER

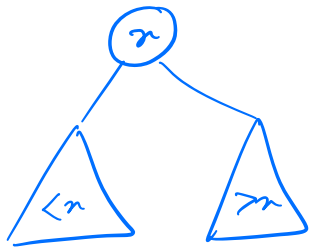
5 6 7 9 10 12 15 20

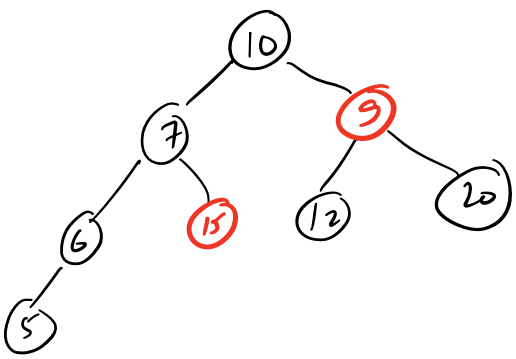


FACT: INORDER of a BST is SORTED in INC order!

IN:



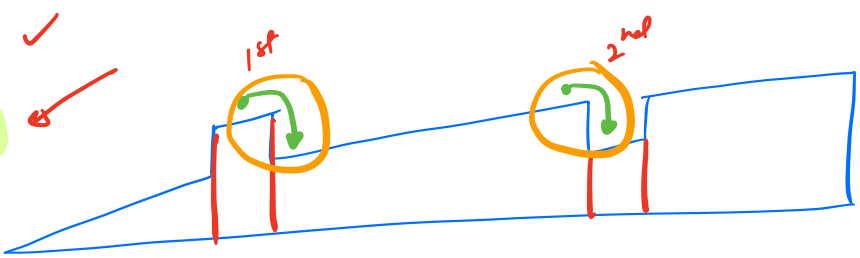


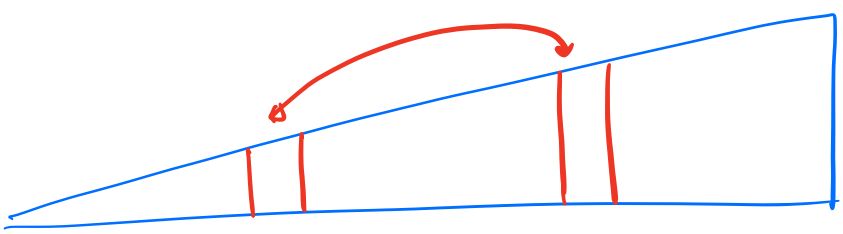


INORDER: 5 6 7 15 10 12 9 20

TC = O(N) ✓

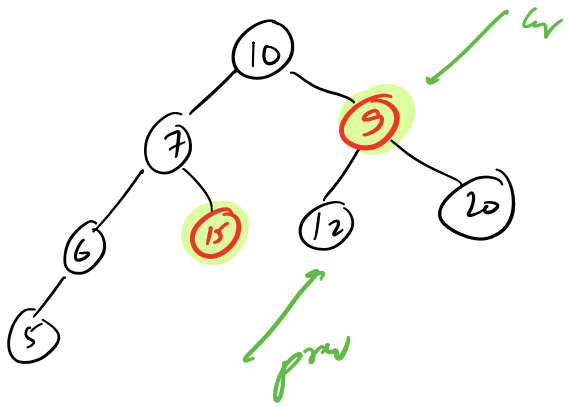
SC = O(N)





II

```
prev = -∞;
void inorder(Node cur) {
    if(cur == NULL) ret;
    inorder(cur.left);
    if(prev > cur.data) → problem
    prev = cur.data;
    inorder(cur.right);
}
```



TC = O(N)

SC = O(H)

III Morris

```
void morrisIOT(Node root) {
    Node cur = root;
    while(cur != NULL) {
        if(cur.left != NULL) {
            Node temp = cur.left;
            while(temp.right != NULL && temp.right != cur) {
                temp = temp.right;
            }
            if(temp.right == NULL) {
                temp.right = cur;
```

```
        cur = cur.left
    }
    else {
        temp.right = NULL;
        print(cur.data);
        cur = cur.right;
    }
}
else {
    print(cur.data);
    cur = cur.right;
}
}
```

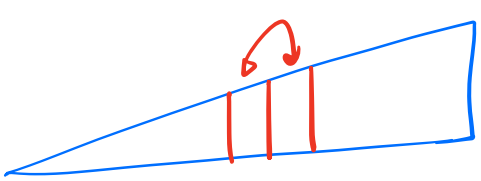
```
if(prev > cur.data){
    → problem
}
prev = cur.data;
```

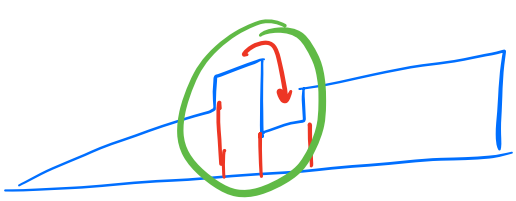
TC = O(N)

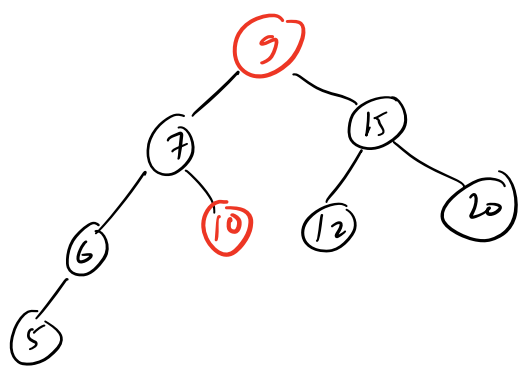
SC = O(1)

CORNER CASE

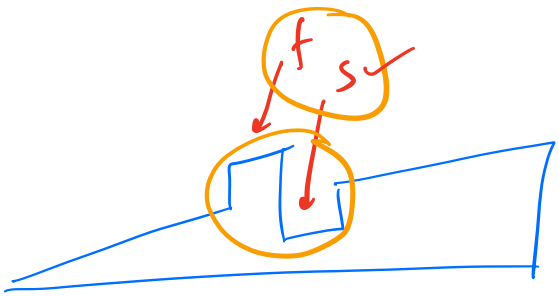
IN: 5 6 7 10 9 12 15 20

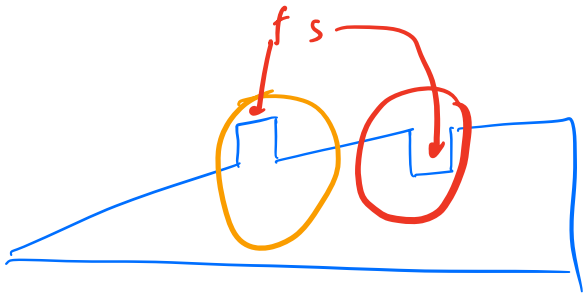






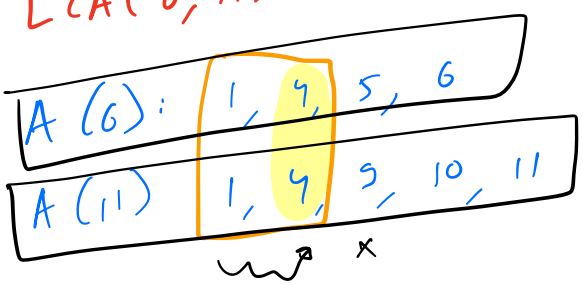
1 DIP

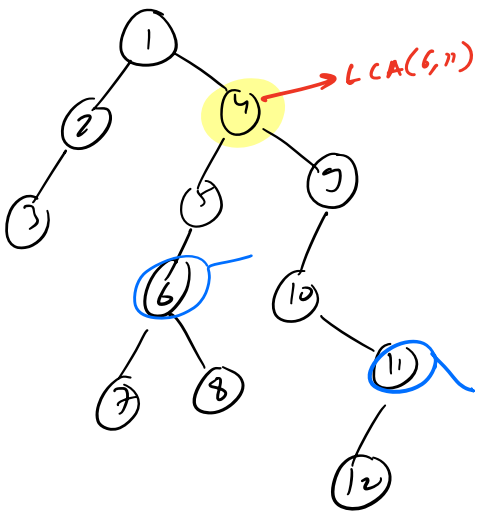




LCA [Lowest | Common | Ancestor] →

LCA(6, 11) →





LCA(3, 7) → 1

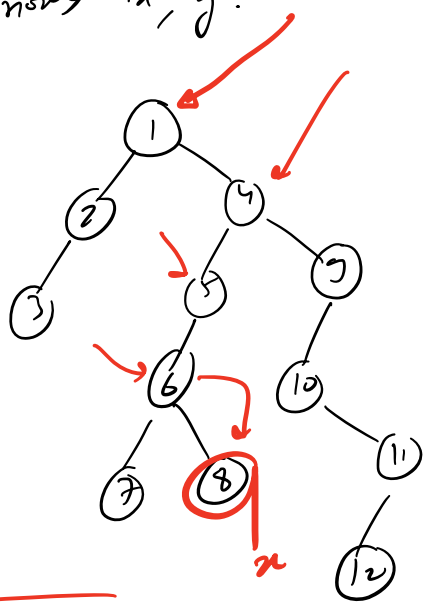
LCA(8, 12) → 4

LCA(7, 8) → 6

LCA(6, 8) → 6

LCA(6, 6) → 6

Q Given a BT. Given 2 nodes x, y.
Find their LCA!



Idea: find the Ancestor list of x & y. Find the last common node!

→ root to node path.

Anc(8) 1 2 4 5 6 8

path[]

```
void getpath(Node root, int x) {
    if (root == NULL) ret;
    path.push_back(root);
    if (root.data == x) { path[] --copy--> ANC[]; }
    getpath(root.left, x);
    getpath(root.right, x);
    path.pop_back();
}
```

N

path x[]
y[]

for (i:0 → n) { → H

find the last common value in path X[] & path Y[].

}

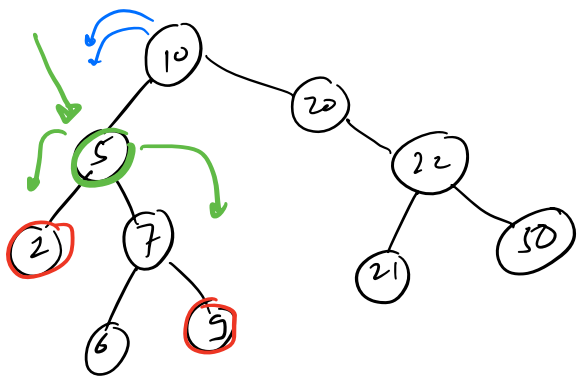
LCA(x, y).

TC : O(N)

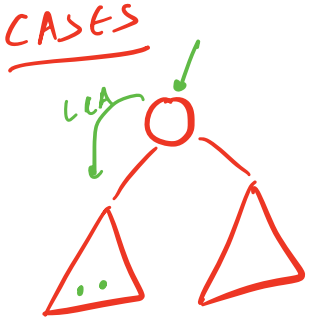
SC = O(H)

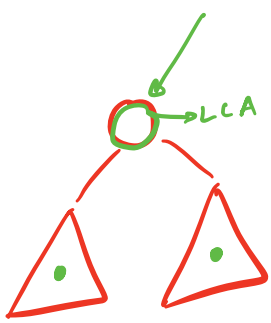
Q Given a BST. find LCA(x, y).

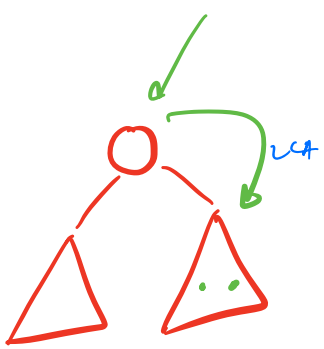
LCA(2, 5)

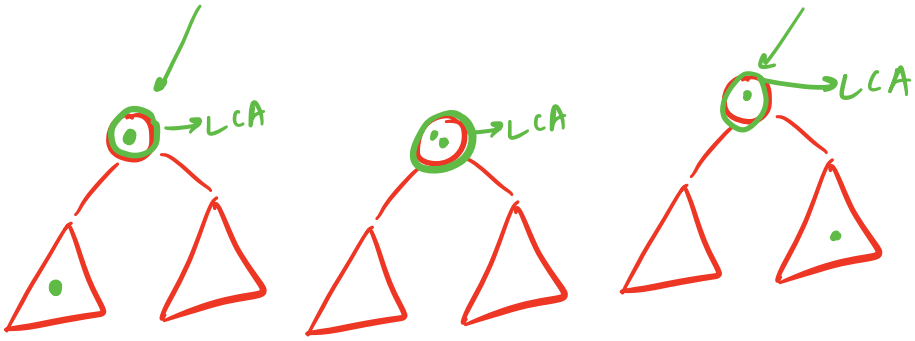


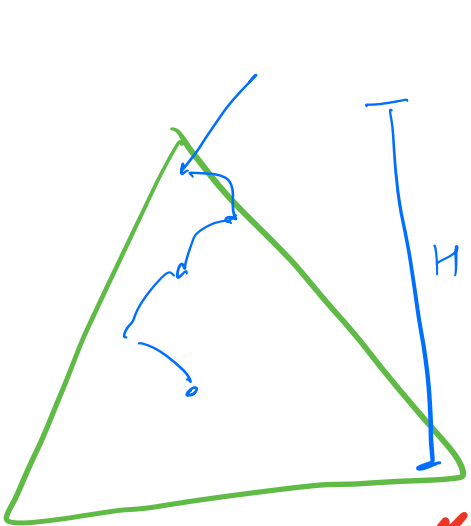
CASES

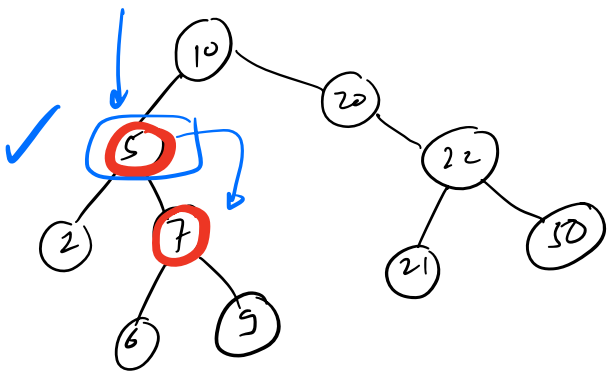








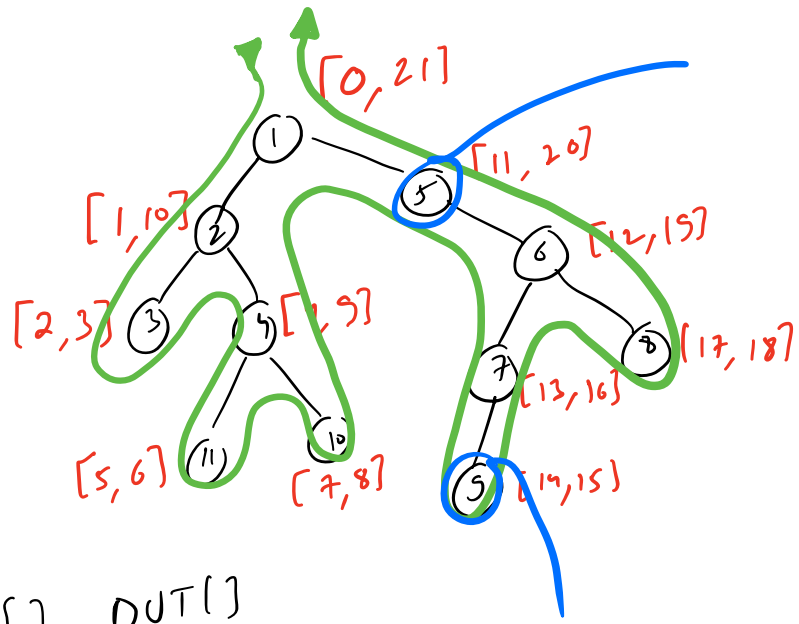




TC = O(H)

SC = O(1)

IN/OUT TIME CONCEPT



T=0; IN[], OUT[]

preorder (—) {

IN[root] = T;
T++;

→ L
→ R

OUT[root] = T;
T++;

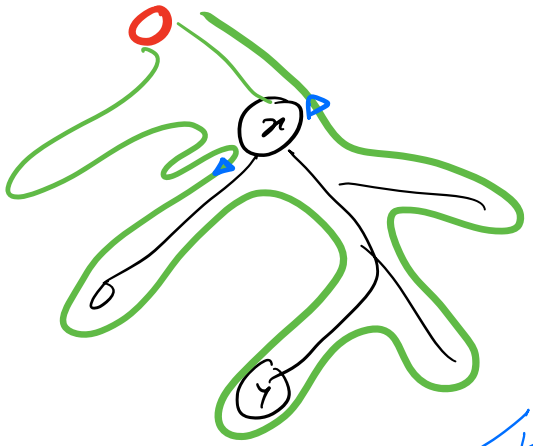
}

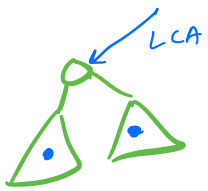
Root

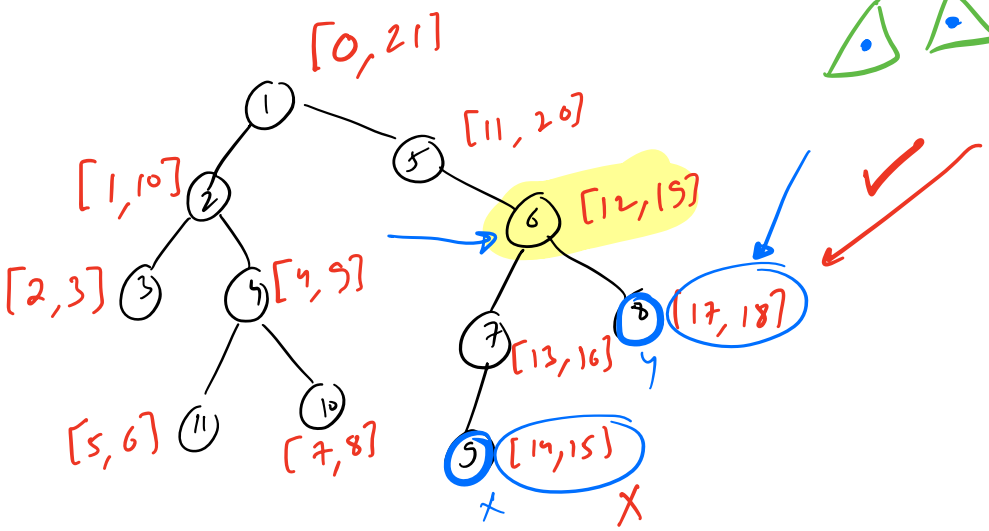
x is an ANC (y)

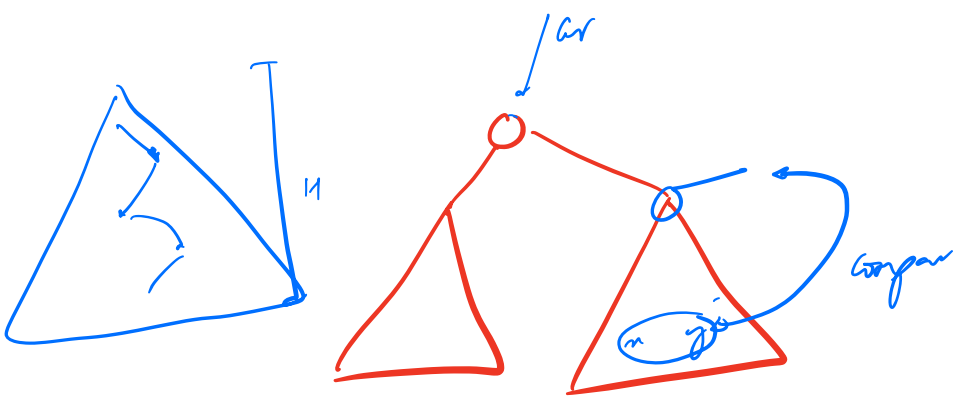
IN(x) ≤ IN(y) &&

OUT(x) ≥ OUT(y)









IN[], OUT[] ⟶

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

1 query ⟶ O(H)
Q ⟶ O(Q · H)

TC = O(N + Q · H)

SC = O(N)