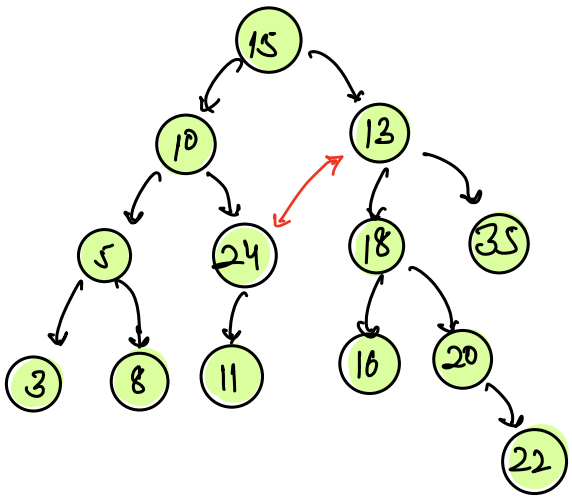
Q) Given root of BST. Two nodes are swapped in given B.S.T, find those two nodes.



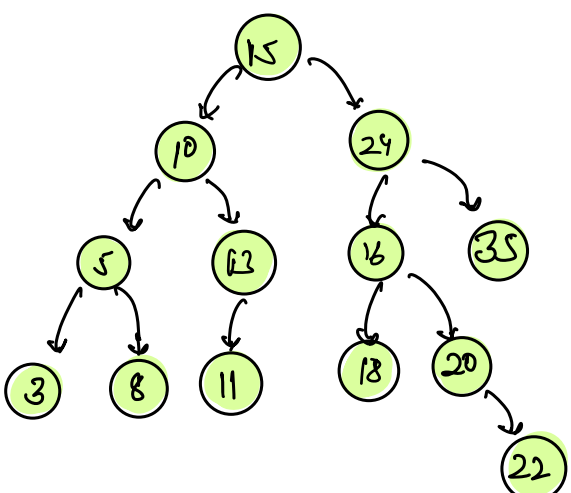
24, 13.

arr → [3, 5, 8, 10, 11, 24, 15, 16, 18, 20, 22, 13, 35]

first · second ·

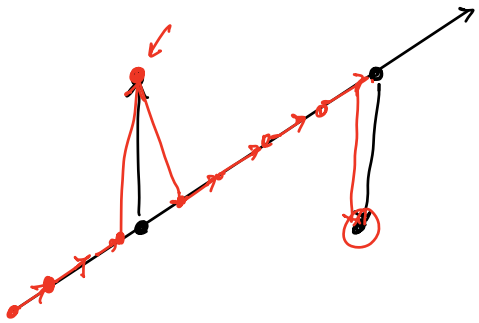
3, 5, 8, 10, 11, 24, 15, 16, 18, 20, 22, 13, 35

24, 13.



T.C → O(N)
S.C → O(N)

in → [3, 5, 8, 10, 11, 13, 15, 16, 18, 20, 22, 24, 35]

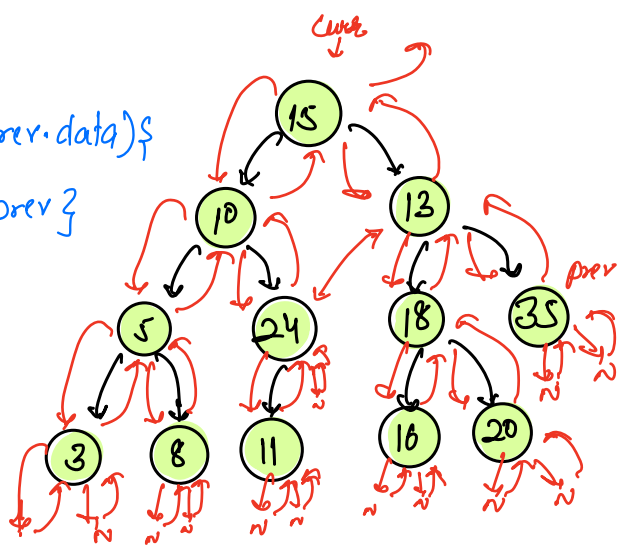


Idea-2. → Since you are looking for adjacent nodes only.
Use prev & curr strategy

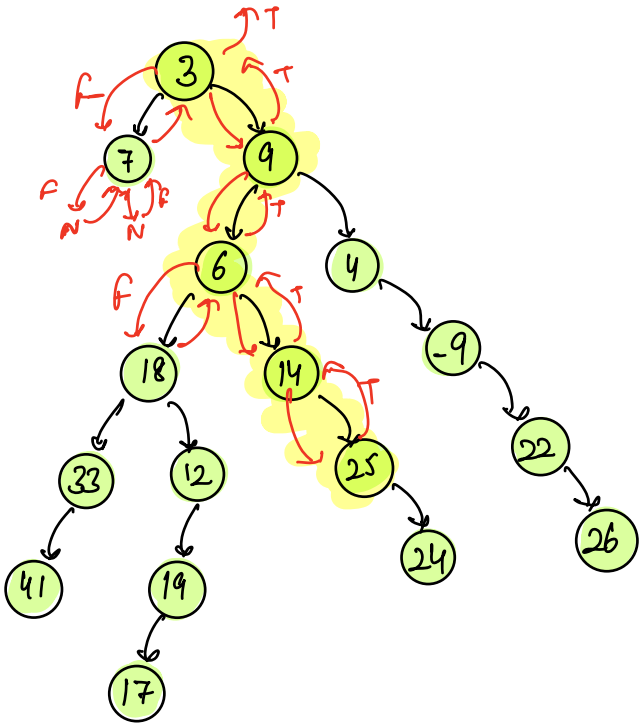
prev = NULL , Node first = NULL, second = NULL

```
find ( Node curr) {
    if (curr == NULL) {return}
    find ( curr.left);
    if (prev != NULL && curr.data < prev.data){
        if (first == NULL) { first = prev}
        second = curr;
    }
    prev = curr;
    find ( curr.right) ;
}
```

prev, first, second → NULL
prev → 24.
first → 13
second → 15



[T.C → O(N) , S.C → O(H)]
↓
Recursive Stack ht.



Binary Tree.

Search (25).

⇓

pre-order. traversal

[3, 9, 6, 14, 25].

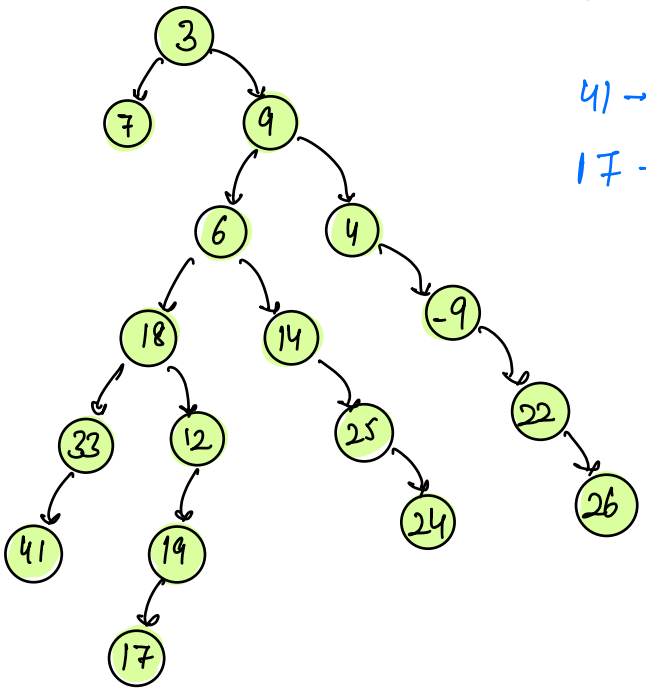
```
boolean search (Node root, int K) {
    if (root == NULL) { return false }
    if (root.data == K) { return true }
    return search(root.left, K) || search (root.right, K)
}
```

path from root to node.

```
boolean search (Node root, int k, list<int> path){
    if (root == NULL) { return false }
    if (root.data == k) { path.insert(root.data), return true }
    if ( search(root.left, k) || search (root.right, k) ){
        path.insert(root.data)
        return true
    }
    return false;
}
```

// reverse the list to get path from root to node.

Lowest Common Ancestor (L.C.A)



41 and 17

41 → [3, 9, 6, 18, 32, 41]

17 → [3, 9, 6, 18, 12, 19, 17]

lca(41, 17) → 18.

[T.C → O(N)
S.C → O(N+H)]

O(1) H todo.

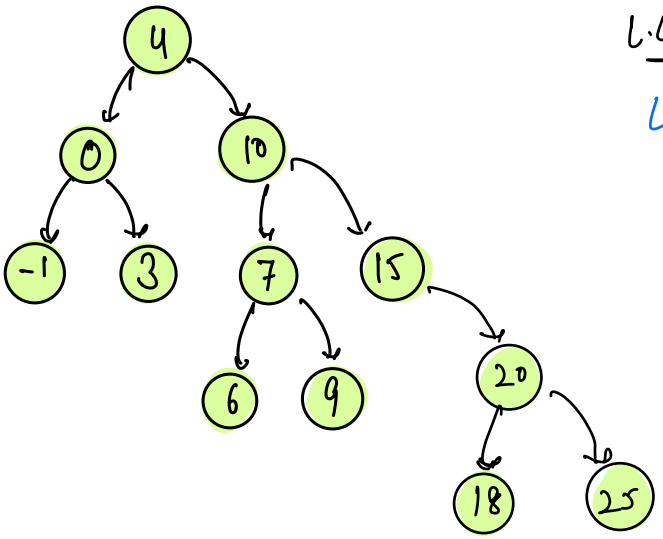
L.C.A (x, y).

① → root to node path (x)

② → root to node path (y)

③ → traverse the path until first non-common ancestor.

L.C.A in B.S.T.



L.C.A(6, 20)

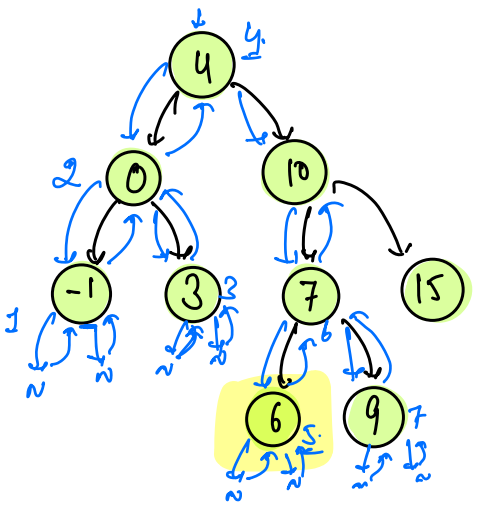
LCA(x, y)

```
Node 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;
    }
}
```

B.B.L.I → O(log₂N)

{ T.C → O(H)
S.C → O(1) }

Q: Find k^{th} smallest element in B.S.T.



K = 3 ⇒ (3)

K = 5 ⇒ (6)

K = 8 ⇒ (10)

Count → 0.

idea-1. in-order & store elements in arr[].

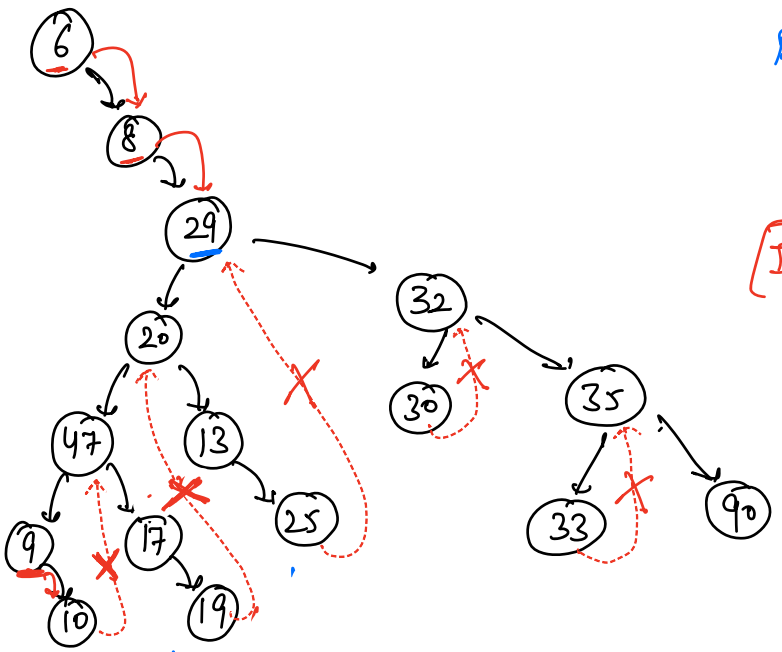
return arr[K-1];

T.C → O(N)
S.C → O(N)

idea-2. On the fly, we can maintain a count. { T.C → O(N), S.C → O(H) }

In-order Traversal of Binary Tree in SC → O(1)

(Morris In-Order Traversal)



Recursion → ✗

iteration → ✓

[In-order predecessor.]

curr

6, 8, 9, 10, 47, 17, 19, 20, 13, 25, 29, 30, 32, 33, 35, 90.

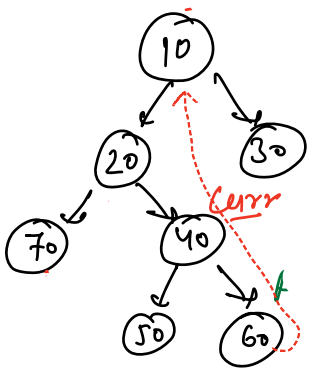
Curr.

```
Node temp = curr.left
while( temp.right != NULL){
    temp = temp.right
}
temp.right = curr.
```

pseudo-code.

```
Node curr = root;
while (curr != NULL) {
    if (curr.left == NULL) {
        print(curr.data);
        curr = curr.right
    }
    else {
        pred = curr.left
        while (pred.right != NULL && pred.right != curr) {
            pred = pred.right
        }
        if (pred.right == NULL) {
            // create the connection
            pred.right = curr, curr = curr.left
        } else {
            // break the connection
            pred.right = null
            print(curr.data);
            curr = curr.right
        }
    }
}
```

T.C → O(N)
S.C → O(1)



70, 20,

→ in-order
→ pre-order in O(1) space (#todo)