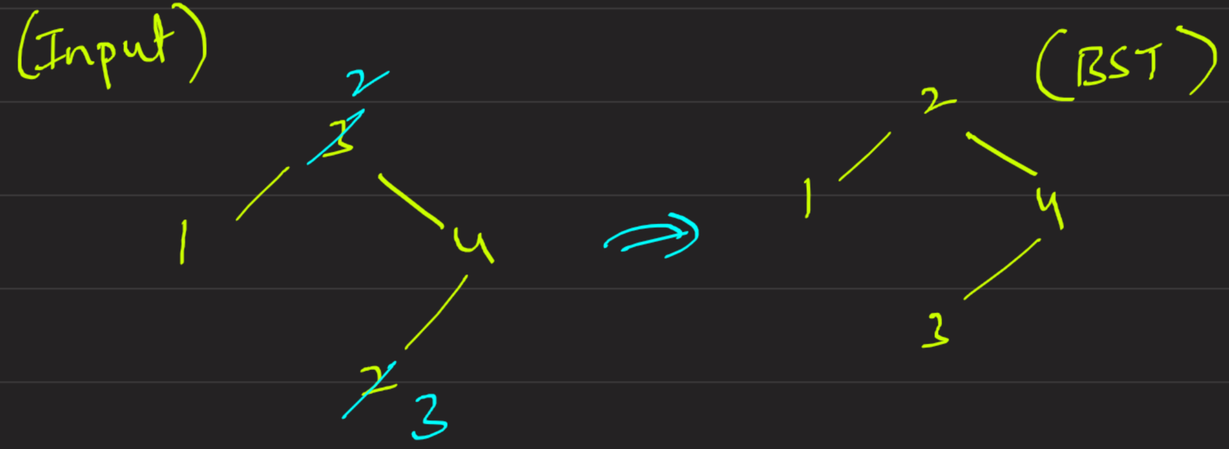
Recover BST

* Given a BST, but with 2 nodes swapped → recover it back to BST



Brute force :

① Inorder Traversal
② Sort → we get correct inOrder
③ Do, inOrder again → and keep check on correct inorder we get from Sort, (is it in the same position)

T.C → O(n) + O(nlogn) + O(n) ≈ O(nlogn)
S.C → O(n)

Swap can have 2 cases

① Swapped nodes are not adjacent

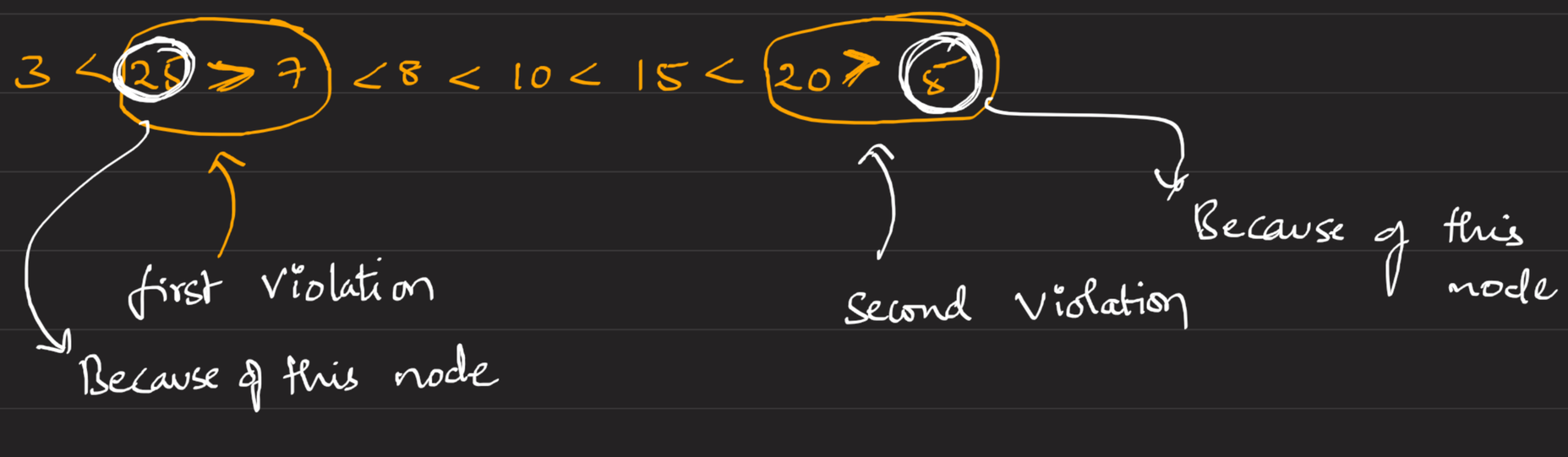
3 (25) 7 8 10 15 20 (5) { these are not adjacent }

② Swapped nodes are adjacent

3 5 (8) (7) 10 15 20 25 { these are adjacent }

① Swapped nodes are not adjacent

3 (25) 7 8 10 15 20 (5) { these are not adjacent }

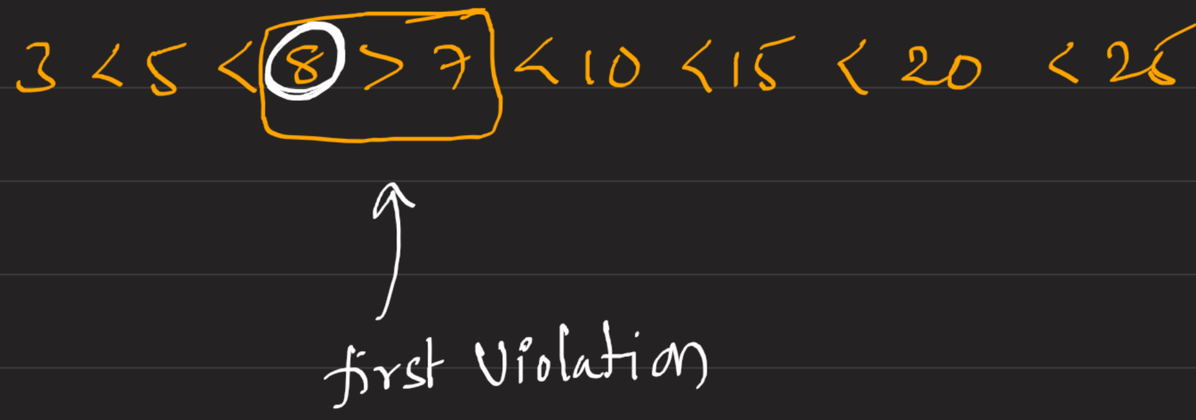


* Swap these 2 nodes

→ what if there is no 2nd violation ?

② Swapped nodes are adjacent

3 5 (8) (7) 10 15 20 25 { these are adjacent }



* But there is no second Violation ...

To Overcome this,

Simply → When first violation is Occured

Store, first node as (first)

Second node as (middle) → name it whatever you want

* if at all, if you don't find 2nd violation, then Simply Swap (first, middle)

first violation node

second violation node

to compare with node

first, second, prev = null;

```
void inOrder (Node root)
{
    if (root == null) return ;

    inOrder (root.left);                    (1)

    // Do stuff

    inOrder(root.right);
}
```

Pseudo Code

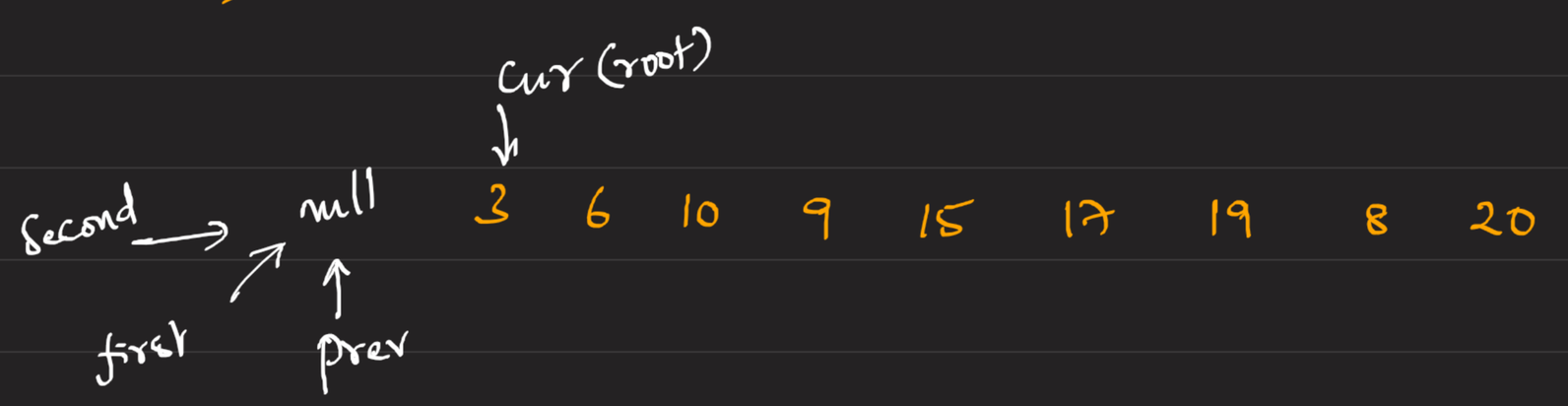
```
// Do Stuff

if (prev != null && (root.val < prev.val && first==null))
{
        first = prev ;
        second = root ;
}

(2) else if(prev != null && root.val < prev.val && first != null)
{
        second = root ;
}

(3)  prev = curr ;
```

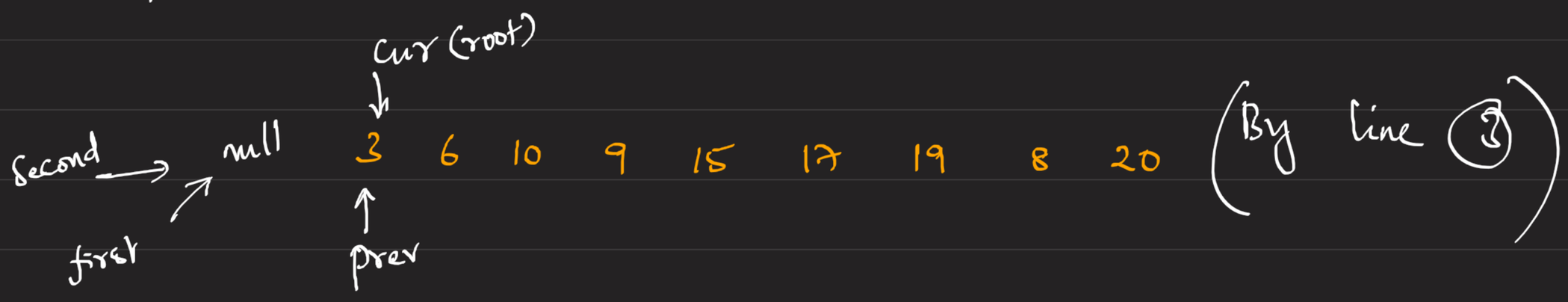
* if 2 violations are occured → (first) named variable points to first violated node

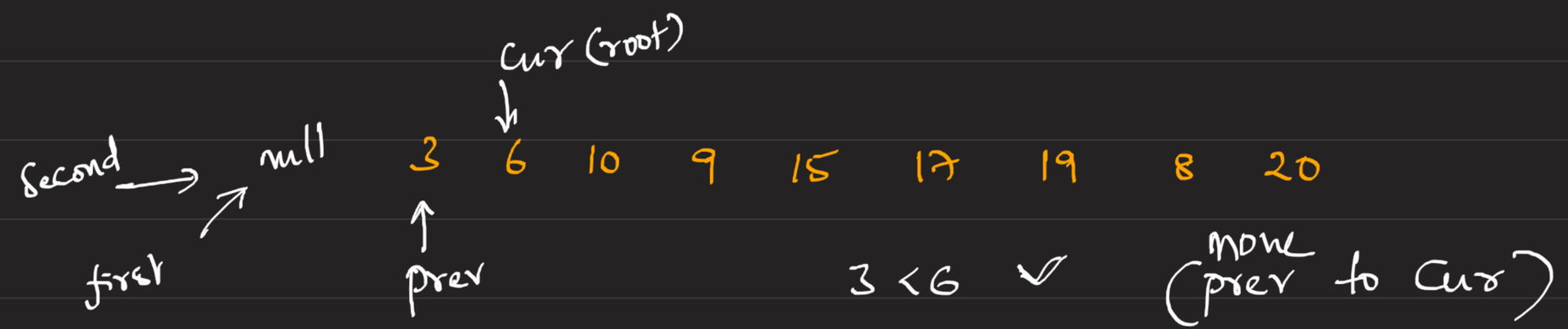
(second) named variable points to second violated node

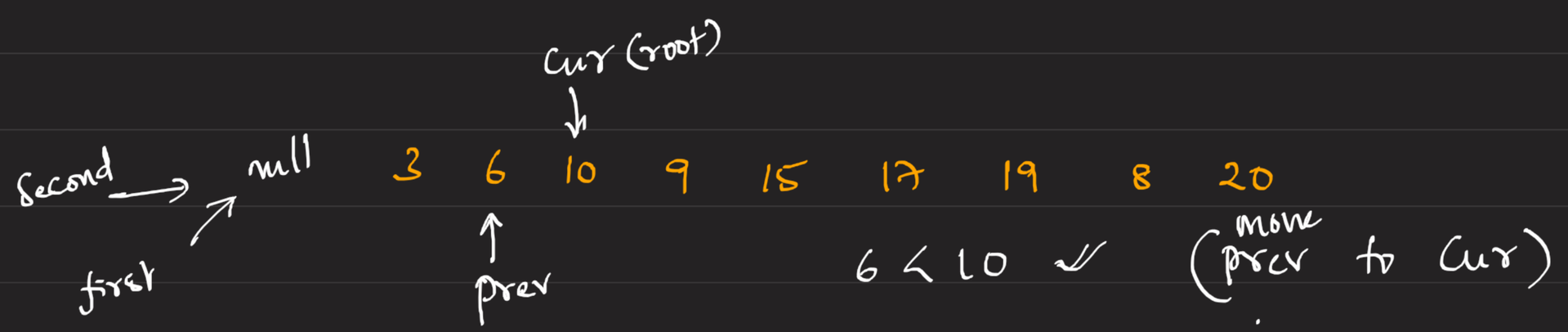
* if 2nd violation is not occured → {first, second} are consecutive nodes of first violation

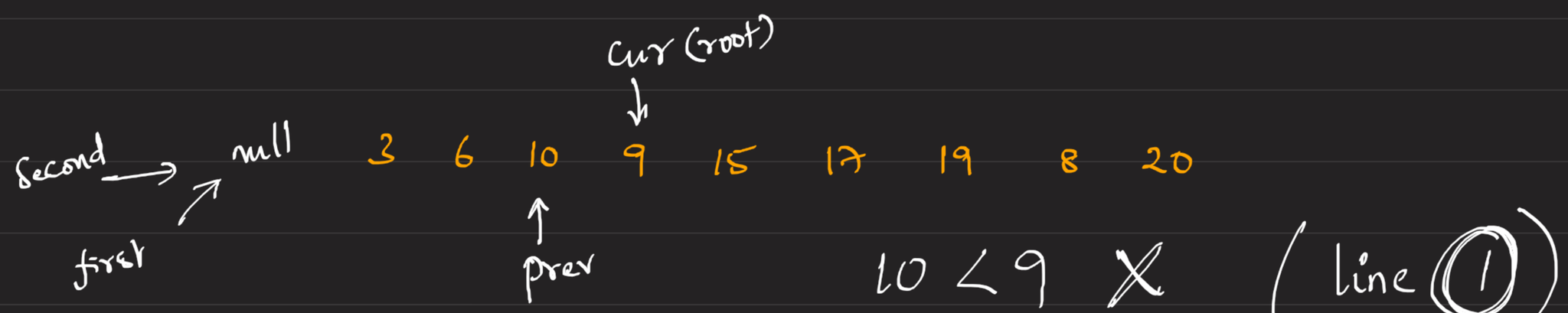
Ex:- 3 6 10 9 15 17 19 8 20



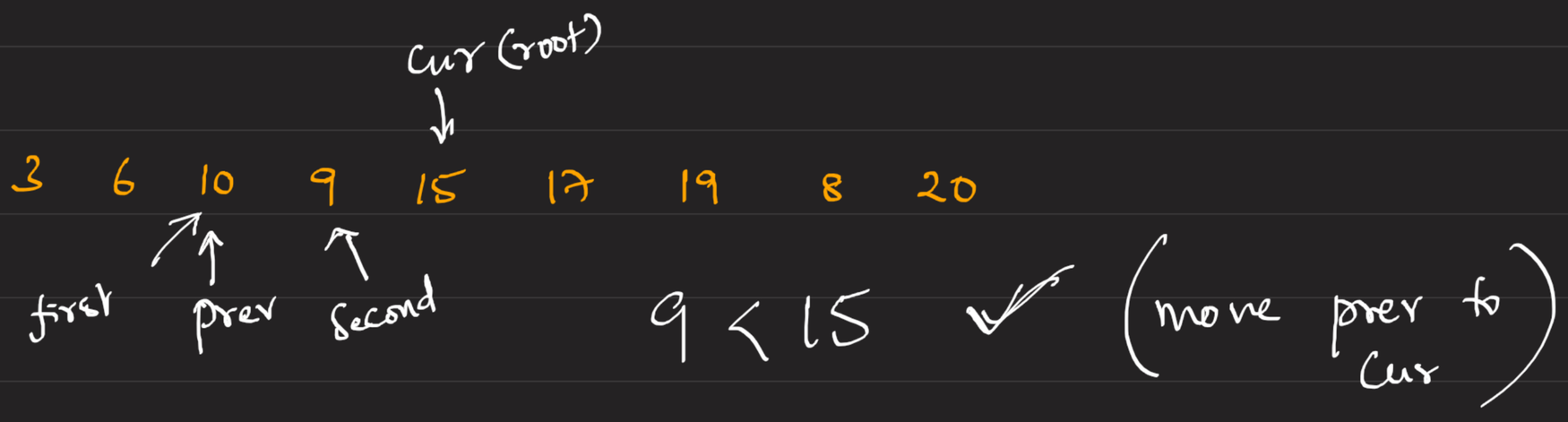


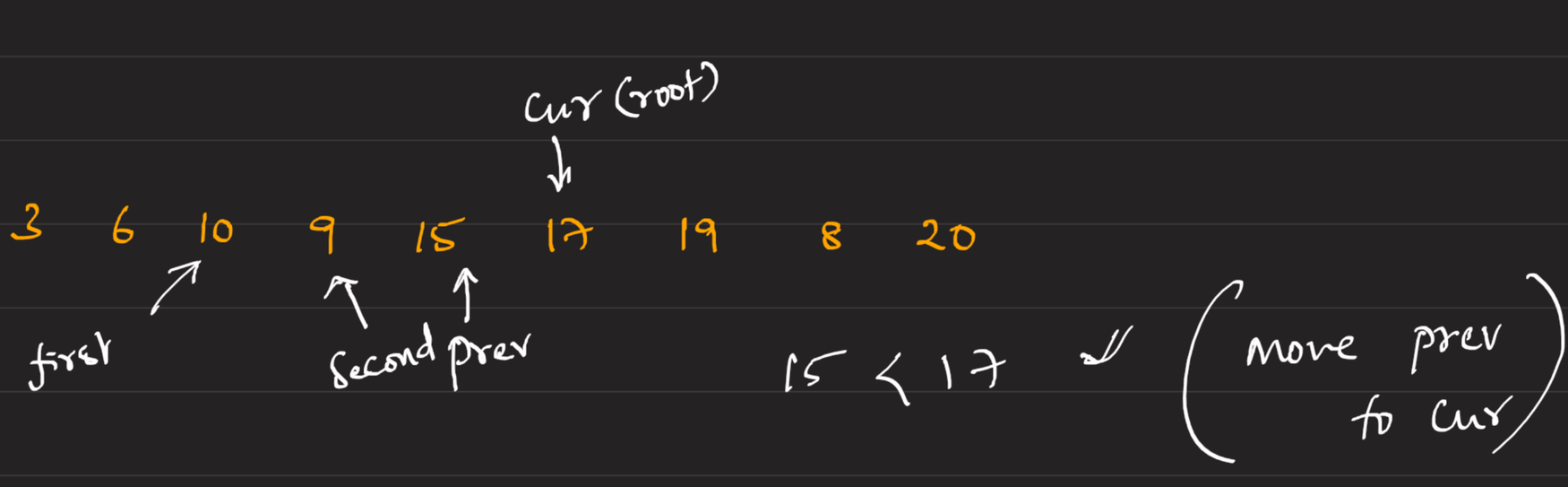

Cur (root)
second
null
first
3 6 10 9 15 17 19 8 20
prev
3 < 6 ✓
(move prev to Cur)

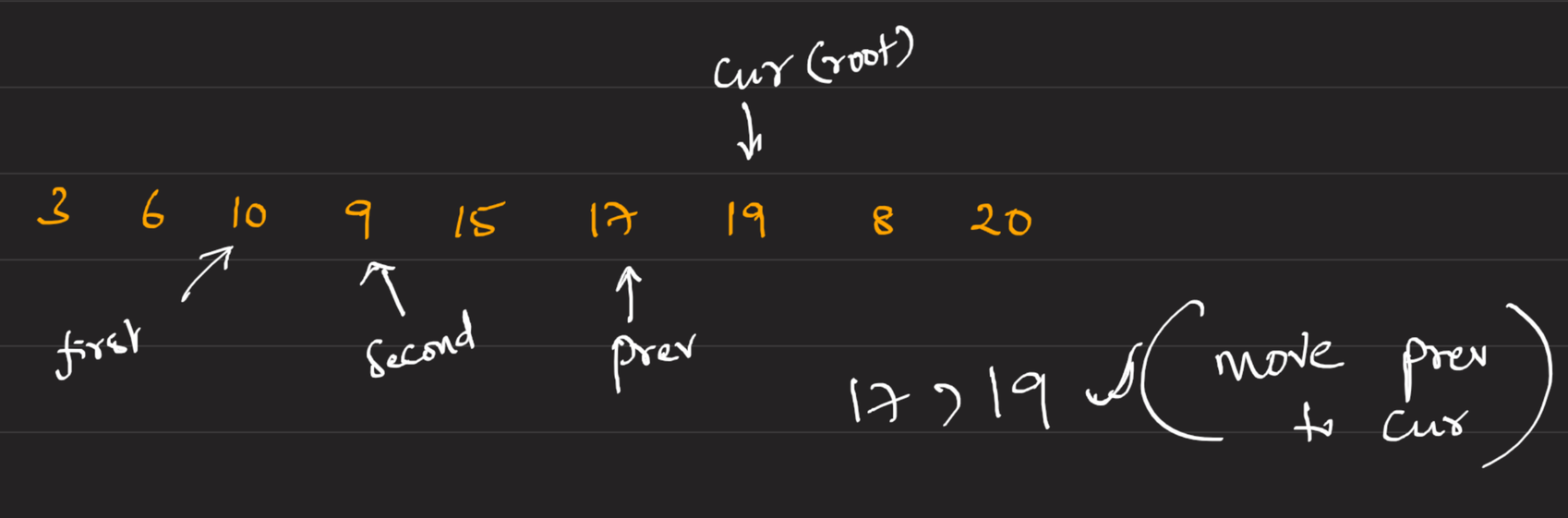

Cur (root)
second
null
first
3 6 10 9 15 17 19 8 20
prev
6 < 10 ✓
(move prev to Cur)

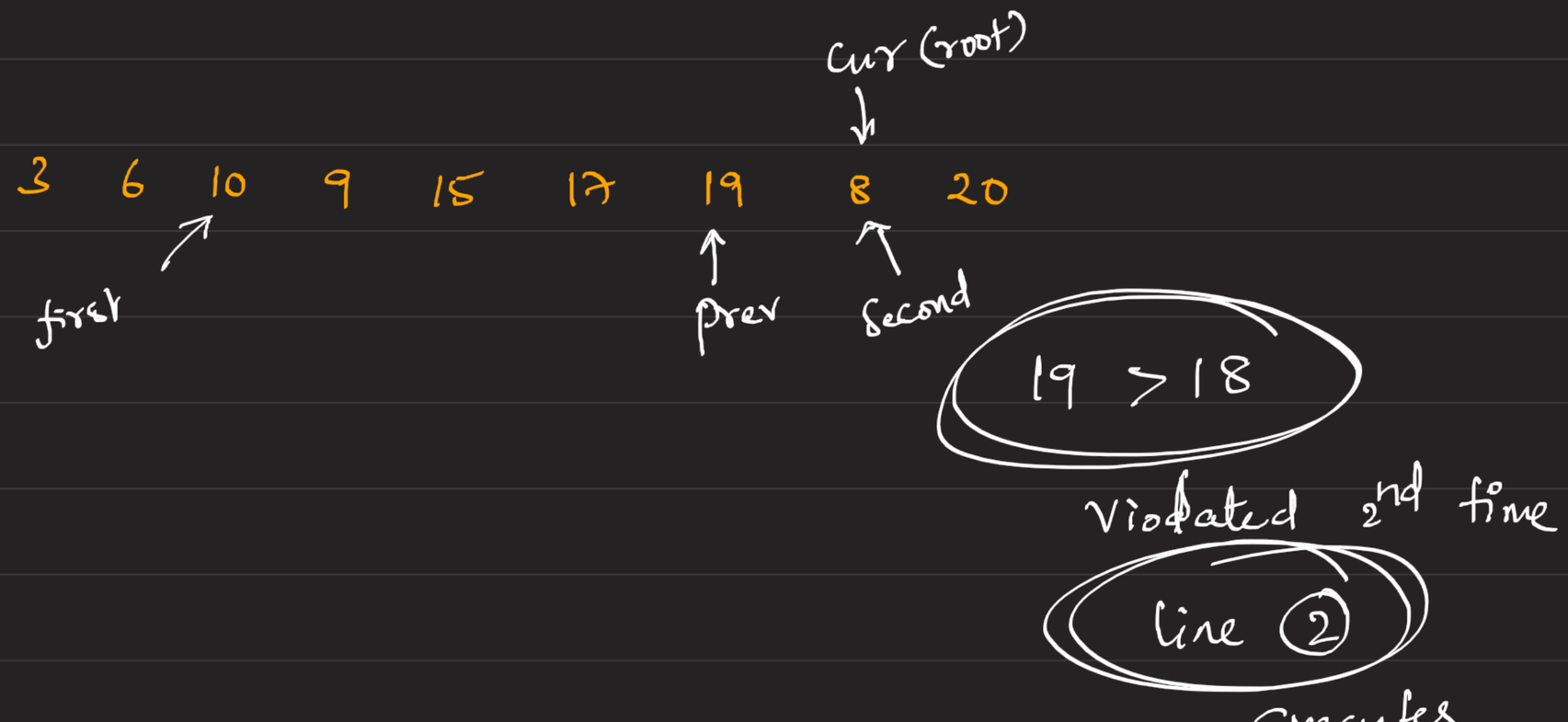

Cur (root)
second
null
first
3 6 10 9 15 17 19 8 20
prev
10 < 9 ✗
(line ① Executes


Cur (root)
3 6 10 9 15 17 19 8 20
first
prev
second
10 < 9 ✗


Cur (root)
3 6 10 9 15 17 19 8 20
first
prev
second
9 < 15 ✓
(move prev to Cur


Cur (root)
3 6 10 9 15 17 19 8 20
first
second
prev
15 < 17 ✓
(move prev to Cur)


Cur (root)
3 6 10 9 15 17 19 8 20
first
second
prev
17 > 19 ✗
(move prev to Cur)



* The moment your inOrder traversal finishes

you will have (first, second) nodes with you

~~***~~ Swap their values

T.C : O(n)

SC : O(1)