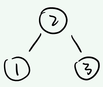
285_Inorder Successor in BST



inorder: left - root - right

for ①: it's inorder successor is in / ①'s right subtree
\ ①'s root

for ②: it's inorder successor is in / ②'s right subtree
\ null if ② has no right child

for ③: it's inorder successor is in / ③'s right subtree
\ null if ③ has no right child

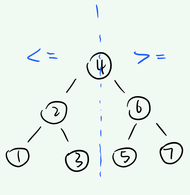
∴

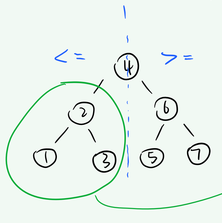
Node's Location	It's Inorder Successor in BST
left subtree	< It's right subtree It's root
root	< It's right subtree null if it has no right child
right subtree	< It's right subtree null if it has no right child

if we use divide and conquer.

```
// corner case
if ( root == null) {
    return root;
}
// p is the root or p is in the right subtree
// find p's inorder successor in the right subtree
  if ( root.val <= p.val) {
      return inorderSuccessor ( root.right, p);
  // p is in the left subtree
  // find p's inorder successor in the left subtree, if not found, return root;
  } else {
      TreeNode left = inorderSuccessor ( root.left, p);
      return (left != null) ?  left : root;
  }
```

230_Kth Smallest Element in a BST





left subtree has 3 nodes
if k <= 3, we find k in left subtree
if k > 3+1, we find k in right subtree
if k == 3+1, k is the root

∵ BST

∴ left tree's nodes' value <= root.val <= right tree's node's value

The smallest elements distribution should be: left tree → root → right tree

We need to count total # of nodes in left subtree, let's call it l,

if k <= l, we do not need to find kth smallest element in the root and right sub tree.

if k > l+1, that means kth smallest element is in the right sub-tree.

Otherwise, k == l+1, kth smallest element is the root.