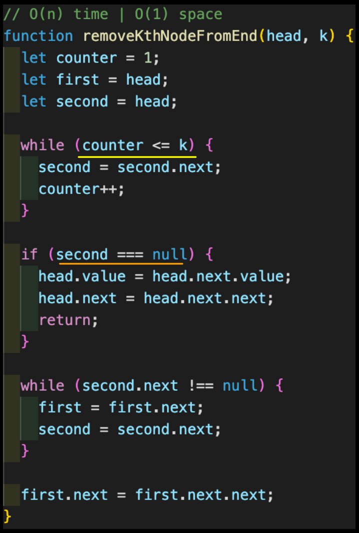
Input: The head of a Singly Linked List
An integer K

head = 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9
K = 4

Output: None The 4th node (from the end) is removed (value 6)

0 → 1 → 2 → 3 → 4 → 5 → 7 → 8 → 9



```
// O(n) time | O(1) space
function removeKthNodeFromEnd(head, k) {
  let counter = 1;
  let first = head;
  let second = head;

  while (counter <= k) {
    second = second.next;
    counter++;
  }

  if (second === null) {
    head.value = head.next.value;
    head.next = head.next.next;
    return;
  }

  while (second.next !== null) {
    first = first.next;
    second = second.next;
  }

  first.next = first.next.next;
}
```

Time: O(n) (where n is the total # of nodes in the linked list) since we iterate through the entire linked list.
We move our second pointer simutaineously so it does not add to the time complexity
Space: O(1) since we are just manipulating pointers

Idea: Start two pointers at the head node. Increment the second pointer until it is <=K (or K nodes AHEAD) of the first pointer
If the second pointer points to null, we know that we have to remove the head (since we've reached the end of the Linked List)
Otherwise, we increment the first and Second pointer until the second pointer is at the last node
We then set the first pointer node to its next, next value and the garbage collector deletes the first.next node we prev. had as no pointers point to it.

Note: The reason we go to <=K is bc we want to move to Second pointer K nodes AHEAD of the first pointer not just at the Kth node

0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9   K = 4
F   S