

# Solution Review: Return the Nth Node from End

This review provides a detailed analysis of the different ways to return the nth node from the end of a linked list

## We'll cover the following



- Solution #1: Double Iteration
  - Time Complexity
- Solution #2: Two Pointers
  - Time Complexity

## Solution #1: Double Iteration #

main.py

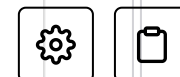
LinkedList.py

Node.py

```
18         return -1
19
20     count = 0
21
22     while count is not position:
23         current_node = current_node.next_element
24         count += 1
25
26     if current_node:
27         return current_node.data
28     return -1
```



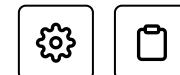
```
29
30
31 lst = LinkedList()
32 lst.insert_at_head(21)
33 lst.insert_at_head(14)
34 lst.insert_at_head(7)
35 lst.insert_at_head(8)
36 lst.insert_at_head(22)
37 lst.insert_at_head(15)
38
39
40 lst.print_list()
41 print(find_nth(lst, 5))
42 print(find_nth(lst, 1))
43 print(find_nth(lst, 10))
44
45
```



In this approach, our main goal is to figure out the index of the node we need to reach. The algorithm follows these simple steps:

1. Calculate the length of the linked list
2. Check if  $N$  is within the length
3. Find the position of the node using  $\text{length} - n + 1$  (We start from the last node since we can't start from `None`)
4. Iterate over to the node and return its value

# Time Complexity #



It performs two iterations on the list so the complexity is  $O(n)$ .

## Solution #2: Two Pointers #

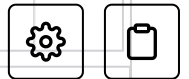
main.py

LinkedList.py

Node.py

```
1 from LinkedList import LinkedList
2 from Node import Node
3
4
5 def find_nth(lst, n):
6
7     if lst.is_empty():
8         return -1
9
10    nth_node = lst.get_head() # This iterator will reach the Nth no
11    end_node = lst.get_head() # This iterator will reach the end of
12
13    count = 0
14    while count < n:
15        if end_node is None:
16            return -1
17        end_node = end_node.next_element
18        count += 1
19
20    while end_node is not None:
21        end_node = end_node.next_element
22        nth_node = nth_node.next_element
23
24    return nth_node.data
25
26
```

```
27 lst = LinkedList()  
28 lst.insert_at_head(21)
```



This is the more efficient approach, although it is not an unfamiliar one. Here's the flow of the algorithm:

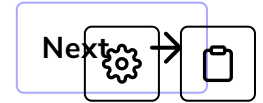
1. Move `end_node` forward `n` times, while `nth_node` stays at the head
2. If `end_node` becomes `None`, `n` was out of bounds of the array. Return `-1` to indicate that the node is not found.
3. One `end_node` is at `nth` position from the start, move both `end_node` and `nth_node` pointers simultaneously.
4. When `end_node` reaches the end, `nth_node` is at the `Nth` position from the end
5. Return the node's value

This algorithm also works in  $O(n)$  time complexity, but it still adopts the policy of one iteration over the whole list. We do not need to keep track of the length of the list.

## Time Complexity #

A single iteration is performed, which means that time complexity is  $O(n)$ .

And there you have it, you've passed all the coding challenges for linked lists. Congratulations! The next section will deal with stacks and queues, two very useful data structures. Before that, try your hand at the quiz in the next lesson. It'll be a good way to reinforce your concepts on linked lists.



Challenge 10: Return the Nth node fro...

Intersection Point of Two Lists

☒ Mark as Completed



Report an  
Issue



Ask a Question

([https://discuss.educative.io/tag/solution-review-return-the-nth-node-from-end\\_\\_introduction-to-linked-lists\\_\\_data-structures-for-coding-interviews-in-python](https://discuss.educative.io/tag/solution-review-return-the-nth-node-from-end__introduction-to-linked-lists__data-structures-for-coding-interviews-in-python))