



# Solution Review: Detect Loop in a Linked List

This review provides an analysis of the solution for the Detect a Loop in a Linked List challenge.

## We'll cover the following ^

- Solution: Using a Set
- Time Complexity

## Solution: Using a Set #

main.py

LinkedList.py

Node.py

```
1 from LinkedList import LinkedList
2 from Node import Node
3
4
5 def detect_loop(lst):
6     # Used to store nodes which we already visited
7     visited_nodes = set()
8     current_node = lst.get_head()
9
10    # Traverse the set and put each node in the visitedNodes set
11    # and if a node appears twice in the map
12    # then it means there is a loop in the set
13    while current_node:
```

```

14         if current_node in visited_nodes:
15             return True
16         visited_nodes.add(current_node) # Insert node in visited node
17         current_node = current_node.next_element
18     return False
19
20 # -----
21
22
23 lst = LinkedList()
24
25 lst.insert_at_head(21)
26 lst.insert_at_head(14)
27 lst.insert_at_head(7)
28

```



This is the primitive approach, but it works nonetheless.

We iterate over the whole linked list and add each visited node to a **visited\_nodes** set. At every node, we check whether it has been visited or not.

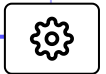
By principle, if a node is revisited, a cycle exists!

## Time Complexity #

We iterate the list once. On average, lookup in a set takes  $O(1)$  time. Hence, the average runtime of this algorithm is  $O(n)$ . However, in the worst case, lookup can increase up to  $O(n)$ , which would cause the algorithm to work in  $O(n^2)$ .

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ



[← Back](#)

Challenge 10: Detect Loop in a Linked ...

Challenge 11: Remove Duplicates fro...

[Mark as Completed](#)[Report an Issue](#)