

# Solution Review: Problem Challenge 1

We'll cover the following ^

- Palindrome LinkedList (medium)
- Solution
  - Code
  - Time complexity
  - Space complexity

## Palindrome LinkedList (medium) #

Given the head of a **Singly LinkedList**, write a method to check if the **LinkedList is a palindrome** or not.

Your algorithm should use **constant space** and the input LinkedList should be in the original form once the algorithm is finished. The algorithm should have  $O(N)$  time complexity where 'N' is the number of nodes in the LinkedList.

### Example 1:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> null
Output: true
```

### Example 2:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> 2 -> null
Output: false
```

## Solution #


As we know, a palindrome LinkedList will have nodes values that read the same backward or forward. This means that if we divide the LinkedList into two halves, the node values of the first half in the forward direction should be similar to the node values of the second half in the backward direction. As we have been given a Singly LinkedList, we can't move in the backward direction. To handle this, we will perform the following steps:


1. We can use the **Fast & Slow pointers** method similar to Middle of the LinkedList (<https://www.educative.io/collection/page/5668639101419520/5671464854355968/6033606055034880/>) to find the middle node of the LinkedList.
2. Once we have the middle of the LinkedList, we will reverse the second half.
3. Then, we will compare the first half with the reversed second half to see if the LinkedList represents a palindrome.


4. Finally, we will reverse the second half of the LinkedList again to revert and bring the LinkedList back to its original form.


Code #

Here is what our algorithm will look like:





 Java

 Python3

 C++

 JS

```
1 class Node:
2     def __init__(self, value, next=None):
3         self.value = value
4         self.next = next
5
6
7 def is_palindromic_linked_list(head):
8     if head is None or head.next is None:
9         return True
10
11     # find middle of the LinkedList
12     slow, fast = head, head
13     while (fast is not None and fast.next is not None):
14         slow = slow.next
15         fast = fast.next.next
16
17     head_second_half = reverse(slow) # reverse the second half
18     # store the head of reversed part to revert back later
19     copy_head_second_half = head_second_half
20
21     # compare the first and the second half
22     while (head is not None and head_second_half is not None):
23         if head.value != head_second_half.value:
24             break # not a palindrome
25
26         head = head.next
27         head_second_half = head_second_half.next
28
```



Time complexity #

The above algorithm will have a time complexity of  $O(N)$  where 'N' is the number of nodes in the LinkedList.

Space complexity #

The algorithm runs in constant space  $O(1)$ .

 Back

Problem Challenge 1

Next 

Problem Challenge 2

 Mark as Completed

Issue

(https://leetcode.com/tag/leetcode-problem-exchange/2-pattern-recognition-engineering-the-coding-interview-patterns-for-coding-questions)

