Technique: Reverse a Linked List

Level: Medium

Is Palindrome: Given a Linked List, determine if it is a Palindrome.

For example, the following lists are palindromes:
A -> B -> C -> B -> A
A -> B -> B -> A
K -> A -> Y -> A -> K

Note: Do it with O(N) time and O(1) space? (Hint: Reverse a part of the list)

Questions to Clarify:
Q. Is this a singly linked list?
A. Yes, there is no previous pointer.

Q. Can we modify the input list?
A. Yes

Q. Do we need to restore the list after modifying it?
A. No need, you can leave it as you wish.

Q. What if the list is empty?
A. Return false

Solution:
We need to check if the first and last elements are equal, and then move inwards. Currently, we cannot do that, because there is no previous pointer.

To do that, we need to reverse the second half of the list. We have already implemented the reverse function in the lecture solution. If we reverse the second half of the list, It will look as follows:

```
K -> A -> Y <- A <- K
```

Now, we can make 2 pointers at either end of the list and move them inwards to verify.

In order to reverse the second half, we will need to find the middle element of the list.

There are two ways to do this:
1. Count the number of nodes in the list. Then move to the middle node.
2. Make a *slow* and *fast* pointer. Advance the *fast* pointer 2 nodes at a time, and the *slow* pointer 1 node at a time. When *fast* reaches the end of the list, *slow* should be at the middle element.

Both these approaches are fine. We'll implement the second approach here, but you can stick to the first approach for simplicity.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a few
lines quickly to plan out your solution. Your actual solution should be in a real
language and use good syntax.)

isPalindrome(Node head):
     Node middle = findMedian(head)
     last = reverse(middle)

     start = head, end = last
     while (start != null && end != null) {
           if (start.data != end.data)
               return false

           start = start.next;
           end = end.next;
     }
     return true // reached end


// Note: for lists of even size, our median element will be floor(n/2),
//       which is ok.
findMedian(head):
    fast = head, slow = head

    while fast.next != null
          fast = fast.next
          if fast.next == null:
              break
          fast = fast.next
          slow = slow.next

    return slow
```

Test Cases:
Edge Cases: empty list
Base Cases: single element, 2 elements (palindrome/not palindrome)
Regular Cases: even and odd elements (palindrome/not palindrome)

Time Complexity: O(n)

Space Complexity: O(1)

```
public static boolean isPalindrome(Node head) {
    if (head == null)
        return false;

    Node median = findMedian(head);
    Node last = reverse(median);
    Node start = head, end = last;
    while (start != null && end != null) {
        if (start.data != end.data)
            return false;

        start = start.next;
        end = end.next;
    }
    return true;
}

public static Node findMedian(Node head) {
    Node fast = head, slow = head;
    while (fast.next != null) {
        fast = fast.next;
        if (fast.next == null)
            break;
        fast = fast.next;
        slow = slow.next;
    }
    return slow;
}

/* This function is taken from the lecture solution */
public static Node reverse(Node head) {
    Node prev = null, curr = head;
    while (curr != null) {
        Node next = curr.getNext();
        curr.setNext(prev);
        prev = curr;
        curr = next;
    }
    return prev;
}
```