Technique: Slow Pointer, Fast Pointer

Level: Easy

Find if a given Linked List has a cycle.

Questions to Clarify:
Q. How do you want the output?
A. Return a boolean.

Solution:
One thing to note: If the list has a cycle, there will be no tail node. Try it yourself.

We use a fast pointer and a slow pointer. If the two pointers meet, the list has a cycle.
If the fast pointer reaches the end of the list, there is no cycle.

This technique may not be intuitive, please watch the video for a detailed explanation.

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.)

```
init 2 pointers - fast and slow
while (end of list is not reached)
     advance fast by 1 nodes
     return true if fast = slow
     advance fast by 1 node (check for fast = null)
     return true if fast = slow
     advance slow by 1 node
end of list reached, return false
```

Test Cases:
Edge Cases: Empty list, one node list
Base Cases: cycle of size 1, 2
Regular Cases: no cycle, whole list is cycle/circular, part of list is cycle

Time Complexity: O(n)

Space Complexity: O(1)

```
public static boolean hasCycle(Node head) {
    Node fast = head, slow = head;
    while (fast != null) {
        fast = fast.getNext();
        if (fast == slow)
```

```
            return true;
        if (fast != null) {
            fast = fast.getNext();
            if (fast == slow)
                return true;
        }
        slow = slow.getNext();
    }
    return false;
}
```

Level: Medium

Given a linked list that has a cycle, find the length of the cycle. The length is in number of nodes.

Questions to Clarify:
Q. How do you want the output
A. Return the number of nodes in the cycle

Q. Can we assume the input has a cycle?
A. No

Solution:
This problem is an extension of finding cycles. The solution for this is simple.

Once the fast and slow pointers meet, we know that they point to a node in the cycle.
Now take the fast pointer. Keep advancing it until it meets the slow pointer again. By doing this, it made a full circle around the cycle. The number of nodes it passed is the length of the cycle

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.)

```
Advance fast and slow pointers until they meet
fast = fast.next;
nodes_passed = 1;
while fast != slow
     fast = fast.next
     nodes_passed += 1
return nodes_passed
```

Test Cases:
Edge Cases: No cycle
Base Cases: Cycle of 1 node, Cycle of 2 nodes, 3 Nodes
Regular Cases: Odd cycle, even cycle, whole list is cycle

Time Complexity: O(n)

Space Complexity: O(1)

```
public static int findCycleLength(Node head) {
    Node fast = head, slow = head;
    while (fast != null) {
        fast = fast.getNext();
        if (fast == slow)
```

```
            break;
        if (fast != null) {
            fast = fast.getNext();
            if (fast == slow)
                break;
        }
        slow = slow.getNext();
    }

    if (fast == null) // no cycle found
        return -1;

    fast = fast.getNext();
    int nodesPassed = 1;
    while (fast != slow) {
        fast = fast.getNext();
        nodesPassed += 1;
    }
    return nodesPassed;
}
```

Level: Medium

Find the median node of a linked list. For example:

1 -> 2 -> 3 -> 4 -> 5 Median node is 3.

Questions to Clarify:
Q. If there are even number of nodes, which node to return?
A. Return either one of the 2 middle nodes.

Solution:
We use a slow pointer and a fast pointer until the fast pointer reaches the end of the list. If the slow pointer is moving at half the pace of the fast pointer, it will be pointing to the median.

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.)

```
// for even numbered lists, this will point to the first of the 2 middle nodes
median(head, tail)
    fast = head, slow = head
    while (fast.next != null)
         fast = fast.next
         if (fast.next != null)
             fast = fast.next
             slow = slow.next
    return slow
```

Test Cases:
Edge Cases: empty list
Base Cases: single node list
Regular Cases: 2 nodes, 3 nodes, odd, even

Time Complexity: O(n)

Space Complexity: O(1)

```
/*
 * Note: We passed head and tail as arguments here, but be careful with this
 * approach. If your function modifies the linked list, this will not work.
 *
 * For example, if you need to change the head and tail pointers, simply reassigning
 * them in this function call will not change the original variables. In most cases,
 * we recommend making list modification functions a part of the LinkedList class.
 */
public static Node findMedian(Node head, Node tail) {
    if (head == null || tail == null)
        return null;
    Node fast = head, slow = head;
    while(fast.getNext() != null) {
        fast = fast.getNext();
        if (fast.getNext() != null) {
            fast = fast.getNext();
            slow = slow.getNext();
        }
    }
    return slow;
}
```