Middle of the LinkedList (easy)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time complexity
  - Space complexity

Problem Statement

Given the head of a **Singly LinkedList**, write a method to return the **middle node** of the LinkedList.

If the total number of nodes in the LinkedList is even, return the second middle node.

Example 1:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> null
Output: 3
```

Example 2:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
Output: 4
```

Example 3:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> null
Output: 4
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```python
class Node:
  def __init__(self, value, next=None):
    self.value = value
    self.next = next


def find_middle_of_linked_list(head):
  # TODO: Write your code here
  return head


def main():
  head = Node(1)
  head.next = Node(2)
  head.next.next = Node(3)
  head.next.next.next = Node(4)
  head.next.next.next.next = Node(5)

  print("Middle Node: " + str(find_middle_of_linked_list(head).value))

  head.next.next.next.next.next = Node(6)
  print("Middle Node: " + str(find_middle_of_linked_list(head).value))

  head.next.next.next.next.next.next = Node(7)
  print("Middle Node: " + str(find_middle_of_linked_list(head).value))
```

```
main()
```

Run Save Reset

Solution

One brute force strategy could be to first count the number of nodes in the LinkedList and then find the middle node in the second iteration. Can we do this in one iteration?

We can use the **Fast & Slow pointers** method such that the fast pointer is always twice the nodes ahead of the slow pointer. This way, when the fast pointer reaches the end of the LinkedList, the slow pointer will be pointing at the middle node.

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```python
class Node:
  def __init__(self, value, next=None):
    self.value = value
    self.next = next


def find_middle_of_linked_list(head):
  slow = head
  fast = head
  while (fast is not None and fast.next is not None):
    slow = slow.next
    fast = fast.next.next
  return slow


def main():
  head = Node(1)
  head.next = Node(2)
  head.next.next = Node(3)
  head.next.next.next = Node(4)
  head.next.next.next.next = Node(5)

  print("Middle Node: " + str(find_middle_of_linked_list(head).value))

  head.next.next.next.next.next = Node(6)
  print("Middle Node: " + str(find_middle_of_linked_list(head).value))

  head.next.next.next.next.next.next = Node(7)
  print("Middle Node: " + str(find_middle_of_linked_list(head).value))


main()
```

Run Save Reset

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 $O(1)$.