# Question

Given a non-empty, singly linked list with head node head, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

**Example 1:**

**Input:** [1,2,3,4,5]

**Output:** Node 3 from this list (Serialization: [3,4,5])

The returned node has value 3. (The judge's serialization of this node is [3,4,5]).

Note that we returned a ListNode object ans, such that:

ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, and ans.next.next.next = NULL.

**Example 2:**

**Input:** [1,2,3,4,5,6]

**Output:** Node 4 from this list (Serialization: [4,5,6])

Since the list has two middle nodes with values 3 and 4, we return the second one.

**Note:**

* The number of nodes in the given list will be between 1 and 100.

# Solution

#### **Approach 1: Output to Array**

**Intuition and Algorithm**

Put every node into an array A in order. Then the middle node is just A[A.length // 2], since we can retrieve each node by index.

We can initialize the array to be of length 100, as we're told in the problem description that the input contains between 1 and 100 nodes.

|  |
| --- |
| class Solution {  public ListNode middleNode(ListNode head) {  ListNode[] A = new ListNode[100];  int t = 0;  while (head != null) {  A[t++] = head;  head = head.next;  }  return A[t / 2];  }  } |

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*), where N*N* is the number of nodes in the given list.
* Space Complexity: O(N)*O*(*N*), the space used by A.

#### **Approach 2: Fast and Slow Pointer**

**Intuition and Algorithm**

When traversing the list with a pointer slow, make another pointer fast that traverses twice as fast. When fast reaches the end of the list, slow must be in the middle.

|  |
| --- |
| class Solution {  public ListNode middleNode(ListNode head) {  ListNode slow = head, fast = head;  while (fast != null && fast.next != null) {  slow = slow.next;  fast = fast.next.next;  }  return slow;  }  } |

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*), where N*N* is the number of nodes in the given list.
* Space Complexity: O(1)*O*(1), the space used by slow and fast.