

## PROBLEM 1

The screenshot shows the LeetCode interface for problem 141. The top navigation bar includes 'Problem List', 'Submit', and user profile information. The left sidebar shows 'Description', 'Editorial', 'Solutions', and 'Submissions'. The main area displays the problem title '141. Linked List Cycle' and its status as 'Solved'. Below the title is a snippet of code with annotations: 'Given head, the head of a linked list, determine if the linked list has a cycle in it.' A note explains that there is a cycle if some node can be reached again by continuously following the `next` pointer. It specifies that `pos` is used to denote the index of the node that tail's `next` pointer is connected to. A note states that `pos` is not passed as a parameter. The code returns `true` if there is a cycle and `false` otherwise.

**Example 1:**

**Input:** head = [3, 2, 0, -4], pos = 1  
**Output:** true  
**Explanation:** There is a cycle in the linked list, where the tail connects to the second node.

Runtime: 9 ms | Beats 50.60%  
Memory: 11.88 MB | Beats 52.60%

Time (ms)	Percentage (%)
0ms	0%
2ms	~1%
4ms	~2%
6ms	~3%
8ms	~25%
10ms	~15%
12ms	~20%
14ms	~5%
16ms	~2%
18ms	~1%

```
1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9 class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         if (head == NULL || head->next == NULL)
13             return false;
14
15         ListNode* slow = head;
16         ListNode* fast = head;
17
18         while (fast != NULL && fast->next != NULL) {
19             slow = slow->next;
20             fast = fast->next->next;
21
22             if (slow == fast)
23                 return true;
24         }
25
26         return false;
27     }
28 };
29 }
```

## PROBLEM 2

Problem List < > Solved

**142. Linked List Cycle II**

Description | Editorial | Solutions | Submissions

Solved

Medium | Topics | Companies

Given the `head` of a linked list, return *the node where the cycle begins*. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that `tail`'s `next` pointer is connected to (`0-indexed`). It is `-1` if there is no cycle. Note that `pos` is not passed as a parameter.

Do not modify the linked list.

**Example 1:**

`Input: head = [3,2,0,-4], pos = 1`

Code | Accepted | All Submissions | Editorial | Solution

Accepted 18 / 18 testcases passed  
OxShark submitted at Jan 20, 2026 13:10

Runtime: 10 ms | Beats 47.28% | Analyze Complexity

Memory: 11.45 MB | Beats 23.63%

Runtime distribution chart showing a peak at 10ms.

Code | Accepted | All Submissions |

```
9 class Solution {
10 public:
11     ListNode *detectCycle(ListNode *head) {
12         if(head == NULL || head->next == NULL) return NULL;
13
14         ListNode *slow = head;
15         ListNode *fast = head;
16
17         while(fast && fast->next){
18             slow = slow->next;
19             fast = fast->next->next;
20
21             if(slow == fast) break;
22         }
23
24
25         if(fast == NULL || fast->next == NULL) return NULL;
26
27
28         slow = head;
29         while(slow != fast){
30             slow = slow->next;
31             fast = fast->next;
32         }
33
34
35         return slow;
36     }
37 };
```

# PROBLEM 3

The screenshot shows a LeetCode problem page for "206. Reverse Linked List". The problem description asks to reverse a singly linked list given its head. Example 1 shows a list [1, 2, 3, 4, 5] being reversed to [5, 4, 3, 2, 1]. Example 2 is not shown. The submission statistics indicate the code was accepted with 100% runtime beats and 70.83% memory beats. The runtime chart shows a single bar at 0 ms. The memory chart shows a single bar at 13.39 MB. The code itself is a C++ implementation of the iterative reversal algorithm.

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* reverseList(ListNode* head) {
14         ListNode* prev = NULL;
15         ListNode* curr = head;
16
17         while (curr != NULL) {
18             ListNode* nextNode = curr->next;
19             curr->next = prev;
20             prev = curr;
21             curr = nextNode;
22         }
23
24         return prev;
25     }
26 };
27 }
```

## PROBLEM 4

The screenshot shows a coding environment with the following details:

- Problem Details:** Problem 876. Middle of the Linked List. Description: Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.
- Example 1:** A linked list with nodes 1, 2, 3, 4, 5. Node 3 is highlighted in red.
- Example 2:** A linked list with nodes 1, 2, 3, 4, 5, 6. Node 4 is highlighted in red.
- Runtime Analysis:** Accepted, 0 ms | Beats 100.00% | 9.98 MB | Beats 62.09%.
- Memory Analysis:** 9.98 MB | Beats 62.09%.

Code | C++

```
1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     ListNode *next;
6   *     ListNode() : val(0), next(nullptr) {}
7   *     ListNode(int x) : val(x), next(nullptr) {}
8   *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9   * };
10  */
11 class Solution {
12 public:
13     ListNode* middleNode(ListNode* head) {
14         ListNode* slow = head;
15         ListNode* fast = head;
16
17         while (fast != NULL && fast->next != NULL) {
18             slow = slow->next;
19             fast = fast->next->next;
20         }
21
22         return slow;
23     }
24 };
25 }
```