Linked List: Fast and Slow Pointer — Coding Interview Notes (Light Theme)

General Pattern Template

```
def fn(head):
    slow = head
    fast = head
    ans = 0

    while fast and fast.next:
        # do logic
        slow = slow.next
        fast = fast.next.next

    return ans
```

Concept:
The **Fast and Slow Pointer** (also called *Tortoise and Hare*) pattern uses two pointers that traverse a linked list at different speeds — typically one moves one node at a time (slow), and the other two nodes at a time (fast).

Purpose: Detect cycles, find middle nodes, measure lengths, or identify intersections efficiently.
Time Complexity: O(n), Space Complexity: O(1).

Key Ideas

1. Use two pointers moving at different speeds through the list.
2. When the fast pointer reaches the end, the slow pointer is at the midpoint.
3. If a cycle exists, fast and slow will eventually meet inside it.
4. Efficient for O(1) space solutions to linked list problems.

Example 1: Find Middle of Linked List

Goal: Return the middle node of a singly linked list.
Approach: Move slow by one and fast by two steps. When fast reaches the end, slow is at the middle.

```
def find_middle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    return slow
```

Example 2: Detect Cycle in Linked List (Floyd’s Algorithm)

Goal: Detect whether a cycle exists in a linked list.
Approach: If fast and slow pointers ever meet, a cycle exists.

```
def has_cycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

Example 3: Find Start of Cycle

Goal: If a cycle exists, find the node where the cycle begins.
Approach: After detecting the meeting point, move one pointer to head. Move both at 1x speed — they meet at cycle start.

```
def detect_cycle_start(head):
    slow = fast = head

    # Step 1: Detect cycle
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break
    else:
        return None  # No cycle

    # Step 2: Find cycle start
    slow = head
    while slow != fast:
        slow = slow.next
        fast = fast.next

    return slow
```

Summary Table

ProblemApproachComplexity Find middle of listFast 2x, slow 1xO(n) Detect cycleFloyd's algorithm (fast meets slow)O(n) Find cycle startReset one pointer, move both 1xO(n)