Reversing a Linked List — Coding Interview Notes (Light Theme)

General Pattern Template

```
def fn(head):
    curr = head
    prev = None
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node

    return prev
```

Concept:
The **Linked List Reversal** pattern is one of the most fundamental operations in linked list manipulation. It involves reversing the direction of the `next` pointers in place, producing a list where traversal order is reversed.

Applications: reversing entire lists, partial segments, or preparing lists for palindrome checks.
Time Complexity: O(n) **Space Complexity:** O(1)

Key Ideas

1. Maintain three pointers: prev, curr, and next_node.
2. Reverse the link direction at each step (`curr.next = prev`).
3. Move pointers forward: prev ← curr, curr ← next_node.
4. At the end, `prev` points to the new head of the reversed list.

Example 1: Reverse Entire Linked List

Goal: Reverse all nodes in a linked list.
Approach: Iteratively change each node's next pointer to point to its previous node.

```
def reverse_list(head):
    prev = None
    curr = head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    return prev
```

Example 2: Reverse First N Nodes

Goal: Reverse only the first N nodes of a linked list.
Approach: Use the same logic as full reversal, but stop after N steps. Keep track of the next node to reconnect.

```
def reverse_first_n(head, n):
    prev = None
    curr = head
    count = 0
    while curr and count < n:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
        count += 1
    # head is now the tail of the reversed section; connect it to remaining nodes
    head.next = curr
    return prev
```

Example 3: Reverse Sublist (m to n)

Goal: Reverse nodes between positions m and n in a singly linked list.
Approach: Move pointers to m-1 position, reverse sublist of length n–m+1, and reconnect ends.

```
def reverse_between(head, m, n):
    if not head or m == n:
        return head

    dummy = ListNode(0)
    dummy.next = head
    prev = dummy

    # Step 1: move prev to node before m
    for _ in range(m - 1):
        prev = prev.next

    # Step 2: reverse sublist
    curr = prev.next
    next_node = None
    last_unreversed = prev
    sublist_tail = curr

    for _ in range(n - m + 1):
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node

    # Step 3: reconnect
```

```
        last_unreversed.next = prev
        sublist_tail.next = curr

        return dummy.next
```

Summary Table

ProblemApproachComplexity Reverse entire listIterative with 3 pointersO(n) Reverse first N nodesPartial iteration with reconnectO(n) Reverse sublist (m to n)Reverse only middle segmentO(n)