Click on the logo to find Problem Statement

Intuition

Imagine you have two sorted lists, `list1` and `list2`. The task is to merge these two lists into a single sorted list. To do this, you start with two pointers, one for each list, and compare the values at these pointers.

1. **Initialization:**
    - Create a new empty list, represented by the `head` pointer. Also, have a `current` pointer pointing to the current node in the merged list.
2. **Comparison and Merging:**
    - Compare the values at the pointers of `list1` and `list2`. Take the smaller value and add it to the merged list. Move the pointer in the respective list to the next node. Repeat this process until either `list1` or `list2` becomes empty.
3. **Handling Remaining Nodes:**
    - After one of the lists becomes empty, there might be some nodes left in the other list. Since both `list1` and `list2` are already sorted, you can directly attach the remaining nodes to the merged list, without the need for further comparison.
4. **Return:**
    - Finally, return the merged list starting from the node after `head`.

Approach

1. **Initialization:**
    - Create a dummy node called `head` and a pointer `current` initialized to `head`.
    - Traverse both `list1` and `list2` simultaneously using two pointers.
2. **Comparison and Merging:**
    - While both `list1` and `list2` are not empty, compare the values at the pointers of `list1` and `list2`.
    - If the value in `list1` is smaller, add it to the merged list. Move the `list1` pointer to the next node.
    - If the value in `list2` is smaller or equal, add it to the merged list. Move the `list2` pointer to the next node.
    - Move the `current` pointer to the last added node in the merged list.
3. **Handling Remaining Nodes:**
    - After one of the input lists becomes empty, attach the remaining nodes of the non-empty list to the merged list. Since both `list1` and `list2` are already sorted, you can directly attach the remaining nodes.
4. **Return:**
    - Return the merged list starting from the node after `head`.

Complexity

- Time complexity: O(n)
- Space complexity: O(1)

Code

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[
        head = ListNode()
        current = head
```

```
while list1 and list2:
    if list1.val < list2.val:
        current.next = list1
        list1 = list1.next
    else:
        current.next = list2
        list2 = list2.next
    current = current.next

current.next = list1 or list2
return head.next
```