LeetCode 148 — Sort List

1. Problem Title & Link

- **Title:** LeetCode 148 — Sort List
- **Link:** https://leetcode.com/problems/sort-list/

2. Problem Statement (Short Summary)

You are given the head of a **singly linked list**.
Return the list **sorted in ascending order**.

⚠ Must run in **O(n log n)** time.

⚠ Must use **constant space** → No arrays, no copying.

3. Examples (Input → Output)

Example 1

Input: head = [4,2,1,3]

Output: [1,2,3,4]

Example 2

Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Example 3

Input: head = []

Output: []

4. Constraints

- 0 <= nodes <= 5 * 10^4
- -10^5 <= Node.val <= 10^5
- Must use **linked list merge sort**
- Can't use array sorting

5. Core Concept (Pattern / Topic)

Linked List Merge Sort

- Use **fast & slow pointers** to split list into halves
- Recursively sort left and right halves
- Merge two sorted linked lists (like LC 21)

This is the only O(n log n), O(1) extra space sorting method for linked lists.

6. Thought Process (Step-by-Step Explanation)

Why merge sort?

- QuickSort is BAD for linked lists (requires random access).
- Merge sort works very well because merging linked lists is O(n).
- Uses divide and conquer.

Steps:

1. Base case: if list has 0 or 1 node → sorted
2. Use **slow & fast pointers** to find the middle
3. Split list into two halves
4. Recursively sort left half
5. Recursively sort right half
6. Merge two sorted halves
7. Return sorted head

7. Visual / Intuition Diagram

```
List:
4 → 2 → 1 → 3
Split:
4 → 2       |      1 → 3
Sort recursively:
2 → 4       |      1 → 3
Merge:
1 → 2 → 3 → 4
This is classical linked-list merge sort.
```

8. Pseudocode

```
if head == null or head.next == null:
    return head

mid = findMiddle(head)
right = mid.next
mid.next = null

leftSorted = sortList(head)
rightSorted = sortList(right)

return merge(leftSorted, rightSorted)
```

9. Code Implementation

✅ **Python**

```python
class Solution:
    def sortList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head or not head.next:
            return head

        # find middle
        slow, fast = head, head.next
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        mid = slow.next
        slow.next = None

        left = self.sortList(head)
        right = self.sortList(mid)

        return self.merge(left, right)

    def merge(self, l1, l2):
        dummy = ListNode(0)
        curr = dummy

        while l1 and l2:
            if l1.val < l2.val:
                curr.next = l1
                l1 = l1.next
            else:
                curr.next = l2
                l2 = l2.next
            curr = curr.next

        curr.next = l1 if l1 else l2
        return dummy.next
```

✅ **Java**

```java
class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) return head;

        // find middle
        ListNode slow = head, fast = head.next;
```

```
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        ListNode mid = slow.next;
        slow.next = null;

        ListNode left = sortList(head);
        ListNode right = sortList(mid);

        return merge(left, right);
    }

    private ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode curr = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                curr.next = l1;
                l1 = l1.next;
            } else {
                curr.next = l2;
                l2 = l2.next;
            }
            curr = curr.next;
        }

        curr.next = (l1 != null) ? l1 : l2;
        return dummy.next;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(n log n)
Space	O(1) extra (recursion stack not counted)

11. Common Mistakes / Edge Cases

❌ Forgetting to break the list into two halves

❌ Wrong mid-splitting technique → infinite recursion

❌ Incorrect merge logic (not updating pointers)

❌ Handling 1-node lists incorrectly

Edge cases:

- Empty list
- Single node
- Already sorted list
- Reverse sorted list

12. Detailed Dry Run (Step-by-Step Table)

Input:

[4, 2, 1, 3]

Step 1: Find middle

slow stops at 2 → split:

left: 4 → 2

right: 1 → 3

Step 2: Recursively sort

Left side:

4 → 2 → sort → 2 → 4

Right side:

1 → 3 → sort → 1 → 3

Step 3: Merge left & right

Merge:

2,4 and 1,3

1 < 2 → 1

2 < 3 → 2

3 < 4 → 3

4 → 4

Final: 1 → 2 → 3 → 4

13. Common Use Cases

- Sorting linked list for stable ordering

- Cleaning up data pipelines
- Preprocessing for duplicates
- Scheduling tasks by priority

14. Common Traps

- Using quicksort (bad for linked list)
- Forgetting to break mid → infinite recursion
- Incorrect merge pointer update
- Not using fast-slow correctly

15. Builds To (Related Problems)

- **LC 21** — Merge Two Sorted Lists
- **LC 23** — Merge K Sorted Lists
- **LC 147** — Insertion Sort List
- **LC 2** — Add Two Numbers

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Merge Sort (recursive)	O(n log n)	O(1) extra	✔ Best
Merge Sort (bottom-up)	O(n log n)	O(1)	No recursion
Convert to array + sort	O(n log n)	O(n)	❌ violates constraints

17. Why This Solution Works (Short Intuition)

Linked lists merge efficiently, but splitting is slow.
Merge sort splits using slow-fast pointers and merges in O(n).
This gives overall O(n log n) time — perfect!

18. Variations / Follow-Up Questions

- How to make it iterative (bottom-up merge sort)?
- How to sort a doubly linked list?
- How to sort by custom comparator?
- How to sort K linked lists efficiently?