Description

148. Sort List

Sort a linked list in O(n log n) time using constant space complexity.

Example 1:

Input: 4->2->1->3
Output: 1->2->3->4
Example 2:

Input: -1->5->3->4->0
Output: -1->0->3->4->5

Idea

The problem specifically asked O(nlogn), so we can try quick sort or merge sort. But for quick sort, we are not able to swap two linked list node in O(1) time, and we can't traverse from end to beginning because it's singly linked list. So we use merge sort. We split the linked list into two halves using O(n) time. And merge them in O(n) time. And we can split logn times to get the base case where each linked list node is a single node or null. So the time complexity is O(2nlogn) which is essentially O(nlogn). For the space, we only use a dummy head to merge instead of extra array to merge is it's an array, so we only use constant space.

```
public class Solution {
    public ListNode sortList(ListNode head) {
        return mergeSort(head);
    }

    private ListNode mergeSort(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode mid = findMid(head);
        ListNode right = mergeSort(mid.next);
        mid.next = null;
        ListNode left = mergeSort(head);
        return merge(left, right);
    }

    private ListNode merge(ListNode left, ListNode right) {
        ListNode dummy = new ListNode(0);
        ListNode head = dummy;
        while (left != null && right != null) {
            if (left.val < right.val) {
                head.next = left;
                left = left.next;
            } else {
                head.next = right;
                right = right.next;
            }
            head = head.next;
        }

        if (left != null) {
            head.next = left;
        } else {
            head.next = right;
        }

        return dummy.next;
    }

    private ListNode findMid(ListNode head) {
        ListNode slow = head;
        ListNode fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

Summary

- Slow fast two pointers to get mid of the linked list.
- Dummy node for merge.
- Reduce one big problem into two same problems of smaller size.
- Merge sort.
- Modulize different functions.

C++ implementation

```cpp
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        return mergeSort(head);
    }

private:
    ListNode* mergeSort(ListNode* head) {
        if (head == NULL || head->next == NULL) {
            return head;
        }

        ListNode* mid = findMid(head);
        ListNode* right = mergeSort(mid->next);
        mid->next = NULL;
        ListNode* left = mergeSort(head);
        return merge(left, right);
    }

    ListNode* merge(ListNode* left, ListNode* right) {
        ListNode* dummy = new ListNode(0);
        ListNode* head = dummy;
        while (left != NULL && right != NULL) {
            if (left->val < right->val) {
                head->next = left;
                left = left->next;
            } else {
                head->next = right;
                right = right->next;
            }
            head = head->next;
        }
        if (left != NULL) {
            head->next = left;
        } else {
            head->next = right;
        }
        return dummy->next;
    }

    ListNode* findMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```