Sort Linked List (LeetCode 148 - Medium)

Problem Description

Sort a linked list.

Analysis

Merge sort for linked list.

1. Use two pointers to divide the linked list.
2. Sort each sub linked list.
3. Merge two sorted linked list.

Time Complexity: O(nlogn)

Merge Two Sorted Linked List (LeetCode 21 - Easy)

Problem Description

Merge two sorted linked list.

Analysis

1. Iterate through both linked list.
2. Compare and move the pointers until one of the pointer points to null.
3. Append the remaining nodes.

Time Complexity: O(n)

Solution

```
/**
 * LeetCode 148
 * Merge sort for linked list.
 * -> Use two pointers to divide the linked list.
 * -> Sort each sub linked list.
 * -> Merge two sorted linked list.
 * @param head
 * @return sorted linked list
 * @timecomplexity - F(n) = 2*F(n/2) + O(n) = O(nlogn)
 */
public static ListNode sortList(ListNode head) {
      if (head == null || head.next == null) {
            return head;
      }

      ListNode pre_slow = head;
      ListNode slow = head;
      ListNode fast = head;
      while (fast != null && fast.next != null) {
            pre_slow = slow;
            slow = slow.next;
            fast = fast.next.next;
      }

      pre_slow.next = null;
      ListNode list1 = head;
      ListNode list2 = slow;

      list1 = sortList(list1);
      list2 = sortList(list2);

      ListNode rv = mergeLists(list1, list2);

      return rv;
}
```

```java
/**
 * LeetCode 21
 * Merge two sorted linked list.
 * @param list1
 * @param list2
 * @return sorted merged list
 * @timecomplexity - O(n)
 */
private static ListNode mergeLists(ListNode list1, ListNode list2) {
	ListNode dummy = new ListNode(0);
	ListNode temp = dummy;
	while (list1 != null && list2 != null) {
		if (list1.val <= list2.val) {
			temp.next = list1;
			list1 = list1.next;
		} else {
			temp.next = list2;
			list2 = list2.next;
		}
		temp = temp.next;
	}
	if (list1 != null) {
		temp.next = list1;
	}
	if (list2 != null) {
		temp.next = list2;
	}
	return dummy.next;
}
```