**21. Merge Two Sorted Lists**

<https://leetcode.com/problems/merge-two-sorted-lists/>

1. **Listen**

**Problem Statement:**

You are given the **heads** of **two sorted linked lists** list1 and list2.

**Merge** the two lists in a **one** **sorted** **list**.

The list should be made by splicing together the nodes of the first two lists.

**Return** the *head of the merged linked list*.

**Goal**:

Merge two sorted singly **linked lists**.

**Input**:

list1 is head of a sorted singly linked list

list2 is head of a sorted singly linked list

**Return**:

head of merged list

1. **Example**

A picture containing text, clipart, clock

Description automatically generated

**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Constraints:**

* The number of nodes in both lists is in the range [0, 50].
* -100 <= Node.val <= 100
* Both list1 and list2 are sorted in non-decreasing order.

**Test Cases:**

* L1 is even length and L2 is even length
* L1 is odd length and L2 is even length
* L1 is even length and L2 is odd length
* L1 is odd length and L2 is odd length
* L1 is longer length (> 1 greater) than L2
* L2 is longer length (> 1 greater) than L1

**Edge Cases:**

* L1 is empty and L2 is non-empty
* L1 is non-empty and L2 is empty
* L1 is empty and L2 is empty

**Questions:**

Do we want to create all new nodes, or can we use existing nodes from the input lists?

* use existing nodes

1. **Brute Force**

Solution 1: Iterative

Create a third ListNode pointer that will be the head of the new list.

Create a dummy node for the algorithm to avoid edge cases.

Iterate over l1 and l2 at same time, only moving the pointer if we add their node’s value to l3.

if (l1.val <= l2.val)

add l1 to l3

move l1 forward 1

else

add l2 to l3

move l2 forward 1

If one of the lists is longer than the other, we can simply assign the rest of the longer list to the end of the sorted list l3.

Runtime of this algorithm is O(Math.max(n, m)) where n and m are the lengths of list1 and list2.

Space is O(1) since we are using the input nodes.

1. **Optimize**

We could shorten the code by making the algorithm recursive, but we would be increasing the space to O(N) while the runtime would stay the same.

1. **Implement**

Iterative Solution:

public class ListNode {

int val;

ListNode next;

ListNode() {}

ListNode(int val) { this.val = val; }

ListNode(int val, ListNode next) { this.val = val; this.next = next; }

}

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

// edge cases

if(list1 == null || list2 == null) return (list1 == null ? list2 : list1);

// create dummy node

ListNode sorted = new ListNode();

// merge two lists into one sorted list

ListNode curr1 = list1;

ListNode curr2 = list2;

ListNode currSorted = sorted;

while(curr1 != null && curr2 != null)

{

if(curr1.val <= curr2.val)

{

currSorted.next = curr1;

curr1 = curr1.next;

}

else

{

sorted.next = curr2;

curr2 = curr2.next;

}

currSorted = currSorted.next;

}

currSorted.next = (curr1 != null : curr2 : curr1);

return sorted.next;

}

Recursive Solution:

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

if(l1 == null) return l2;

else if(l2 == null) return l1;

if(l1.val < l2.val)

{

l1.next = mergeTwoLists(l1.next, l2);

return l1;

}

else

{

l2.next = mergeTwoLists(l1, l2.next);

return l2;

}

}

1. **Test**