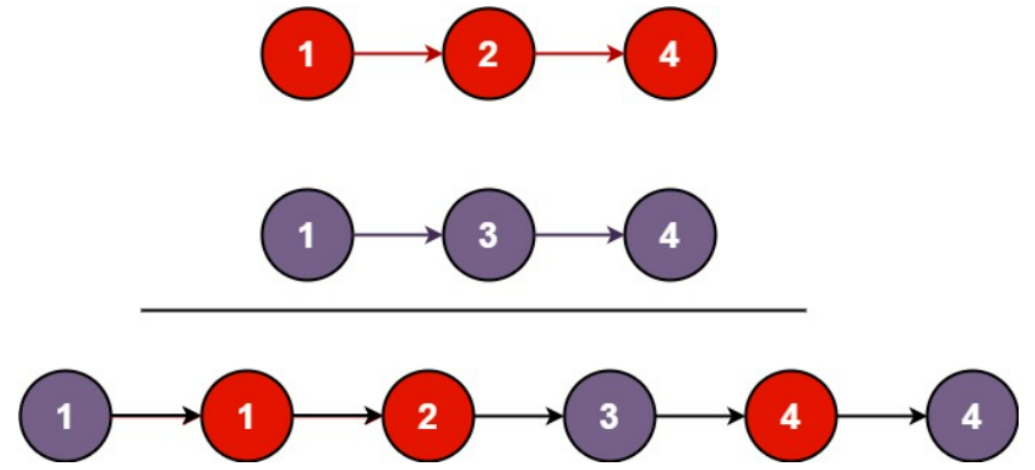
<https://leetcode.com/problems/merge-two-sorted-lists/>

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*.

**Example 1:**



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

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

**Example 2:**

Input: list1 = [], list2 = []

Output: []

**Example 3:**

Input: list1 = [], list2 = [0]

Output: [0]

**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.

**Attempt 1: 2023-02-10**

**Solution 1: Iterative Solution (10 min)**

/\*\*

\* Definition for singly-linked list.

\* 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; }

\* }

\*/

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

ListNode dummy = new ListNode();

ListNode itr = dummy;

ListNode itr1 = list1;

ListNode itr2 = list2;

while(itr1 != null || itr2 != null) {

if(itr1 != null && itr2 != null) {

if(itr1.val < itr2.val) {

itr.next = itr1;

itr1 = itr1.next;

} else {

itr.next = itr2;

itr2 = itr2.next;

}

} else if(itr1 == null) {

itr.next = itr2;

itr2 = itr2.next;

} else {

itr.next = itr1;

itr1 = itr1.next;

}

itr = itr.next;

}

return dummy.next;

}

}

Time Complexity: O(n + m)

Space Complexity: O(1)

**Solution 2: Recursive Solution (10 min)**

/\*\*

\* Definition for singly-linked list.

\* 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; }

\* }

\*/

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

// Base case

if(list1 == null) {

return list2;

}

if(list2 == null) {

return list1;

}

// Divide and Conquer

ListNode head;

// If value pointend by l1 pointer is less than value

// pointed by l2 pointer, we pick l1 pointer as current head,

// then for remain nodes, we wall call recursively l1.next

// and whole l2 list, otherwise the similar way

if(list1.val < list2.val) {

head = list1;

head.next = mergeTwoLists(list1.next, list2);

} else {

head = list2;

head.next = mergeTwoLists(list1, list2.next);

}

return head;

}

}

Time Complexity: O(n + m)

Space Complexity: O(n + m)

**Refer to**

<https://leetcode.com/problems/merge-two-sorted-lists/solutions/9772/java-solution-with-real-world-concerns-real-world-concerns/>

My first solution is an iterative one. One thing deserves discussion is whether we should create a new ListNode as a convenient way to hold the list. Sometimes, in industrial projects, sometimes it's not trivial to create a ListNode which might require many resource allocations or inaccessible dependencies (we need to mock them).

So ideally, we should pick up either the head of l1 or l2 as the head other than creating a new one, which however makes the initialization step tedious.

But during an interview, I would rather create a new ListNode as list holder, but communicate with the interviewer that I'm aware of the potential issue, and would improve it if he/she thinks this is a big deal.

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode head = new ListNode(0);

ListNode handler = head;

while(l1 != null && l2 != null) {

if (l1.val <= l2.val) {

handler.next = l1;

l1 = l1.next;

} else {

handler.next = l2;

l2 = l2.next;

}

handler = handler.next;

}

if (l1 != null) {

handler.next = l1;

} else if (l2 != null) {

handler.next = l2;

}

return head.next;

}

}

My second solution is to use recursion. Personally, I don't like this approach. Because in real life, the length of a linked list could be much longer than we expected, in which case the recursive approach is likely to introduce a stack overflow. (Imagine a file system)

But anyway, as long as we communicate this concerning properly with the interviewer, I don't think it's a big deal here.

public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if (l1 == null) return l2;

if (l2 == null) return l1;

ListNode handler;

if(l1.val < l2.val) {

handler = l1;

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

} else {

handler = l2;

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

}

return handler;

}

}