You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example 1:**



**Input:** l1 = [2,4,3], l2 = [5,6,4]

**Output:** [7,0,8]

**Explanation:** 342 + 465 = 807.

**Example 2:**

**Input:** l1 = [0], l2 = [0]

**Output:** [0]

**Example 3:**

**Input:** l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

**Output:** [8,9,9,9,0,0,0,1]

**Constraints:**

* The number of nodes in each linked list is in the range [1, 100].
* 0 <= Node.val <= 9
* It is guaranteed that the list represents a number that does not have leading zeros.

/\*\*

\* Definition for singly-linked list.

\* function ListNode(val, next) {

\* this.val = (val===undefined ? 0 : val)

\* this.next = (next===undefined ? null : next)

\* }

\*/

/\*\*

\* @param {ListNode} l1

\* @param {ListNode} l2

\* @return {ListNode}

\*/

var addTwoNumbers = function(l1, l2) {

};

Approach 1: Elementary Math

**Intuition**

Keep track of the carry using a variable and simulate digits-by-digits sum starting from the head of list, which contains the least-significant digit.

*Figure 1. Visualization of the addition of two numbers: 342 + 465 = 807342+465=807.  
Each node contains a single digit and the digits are stored in reverse order.*

**Algorithm**

Just like how you would sum two numbers on a piece of paper, we begin by summing the least-significant digits, which is the head of l1*l*1 and l2*l*2. Since each digit is in the range of 0 \ldots 90…9, summing two digits may "overflow". For example 5 + 7 = 125+7=12. In this case, we set the current digit to 22 and bring over the carry = 1*carry*=1 to the next iteration. carry*carry* must be either 00 or 11 because the largest possible sum of two digits (including the carry) is 9 + 9 + 1 = 199+9+1=19.

The pseudocode is as following:

* Initialize current node to dummy head of the returning list.
* Initialize carry to 00.
* Loop through lists l1*l*1 and l2*l*2 until you reach both ends and crarry is 00.
  + Set x*x* to node l1*l*1's value. If l1*l*1 has reached the end of l1*l*1, set to 00.
  + Set y*y* to node l2*l*2's value. If l2*l*2 has reached the end of l2*l*2, set to 00.
  + Set sum = x + y + carry*sum*=*x*+*y*+*carry*.
  + Update carry = sum / 10*carry*=*sum*/10.
  + Create a new node with the digit value of (sum \bmod 10)(*sum*mod10) and set it to current node's next, then advance current node to next.
  + Advance both l1*l*1 and l2*l*2.
* Return dummy head's next node.

Note that we use a dummy head to simplify the code. Without a dummy head, you would have to write extra conditional statements to initialize the head's value.

Take extra caution of the following cases:

| **Test case** | **Explanation** |
| --- | --- |
| l1=[0,1]*l*1=[0,1] l2=[0,1,2]*l*2=[0,1,2] | When one list is longer than the other. |
| l1=[]*l*1=[] l2=[0,1]*l*2=[0,1] | When one list is null, which means an empty list. |
| l1=[9,9]*l*1=[9,9] l2=[1]*l*2=[1] | The sum could have an extra carry of one at the end, which is easy to forget. |

**Implementation**

class Solution {

// Add Two Numbers (Java improved)

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

ListNode dummyHead = new ListNode(0);

ListNode curr = dummyHead;

int carry = 0;

while (l1 != null || l2 != null || carry != 0) {

int x = (l1 != null) ? l1.val : 0;

int y = (l2 != null) ? l2.val : 0;

int sum = carry + x + y;

carry = sum / 10;

curr.next = new ListNode(sum % 10);

curr = curr.next;

if (l1 != null)

l1 = l1.next;

if (l2 != null)

l2 = l2.next;

}

return dummyHead.next;

}

}

**Complexity Analysis**

* Time complexity : O(\max(m, n))*O*(max(*m*,*n*)). Assume that m*m* and n*n* represents the length of l1*l*1 and l2*l*2 respectively, the algorithm above iterates at most \max(m, n)max(*m*,*n*) times.
* Space complexity : O(\max(m, n))*O*(max(*m*,*n*)). The length of the new list is at most \max(m,n) + 1max(*m*,*n*)+1.

**Follow up**

What if the the digits in the linked list are stored in non-reversed order? For example:

(3 \to 4 \to 2) + (4 \to 6 \to 5) = 8 \to 0 \to 7(3→4→2)+(4→6→5)=8→0→7