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.
* Initialize p*p* and q*q* to head of l1*l*1 and l2*l*2 respectively.
* Loop through lists l1*l*1 and l2*l*2 until you reach both ends.
  + Set x*x* to node p*p*'s value. If p*p* has reached the end of l1*l*1, set to 00.
  + Set y*y* to node q*q*'s value. If q*q* 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 p*p* and q*q*.
* Check if carry = 1*carry*=1, if so append a new node with digit 11 to the returning list.
* 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. |

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

ListNode dummyHead = new ListNode(0);

ListNode p = l1, q = l2, curr = dummyHead;

int carry = 0;

while (p != null || q != null) {

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

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

int sum = carry + x + y;

carry = sum / 10;

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

curr = curr.next;

if (p != null) p = p.next;

if (q != null) q = q.next;

}

if (carry > 0) {

curr.next = new ListNode(carry);

}

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.