1、Add Two Numbers

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 contain a single digit. Add the two numbers and return it as a linked list.

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

```
Input:  (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output:  7 -> 0 -> 8
```

2. Add Two Numbers II

You are given two **non-empty** linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

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

Follow up:

What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:

```
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 8 -> 0 -> 7
```

3. Remove Nth Node From End of List

Given a linked list, remove the n^{th} node from the end of list and return its head.

For example,

```
   Given linked list: 1->2->3->4->5, and n = 2.

   After removing the second node from the end, the linked list becomes 1->2->3->5.
```

Note:

Given *n* will always be valid.

Try to do this in one pass.

4. Merge Two Sorted Lists

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

5. Merge k Sorted Lists

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

6. Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

For example,

Given `1->2->3->4`, you should return the list as `2->1->4->3`.

Your algorithm should use only constant space. You may **not** modify the values in the list, only nodes itself can be changed.