21. Merge Two Sorted Lists

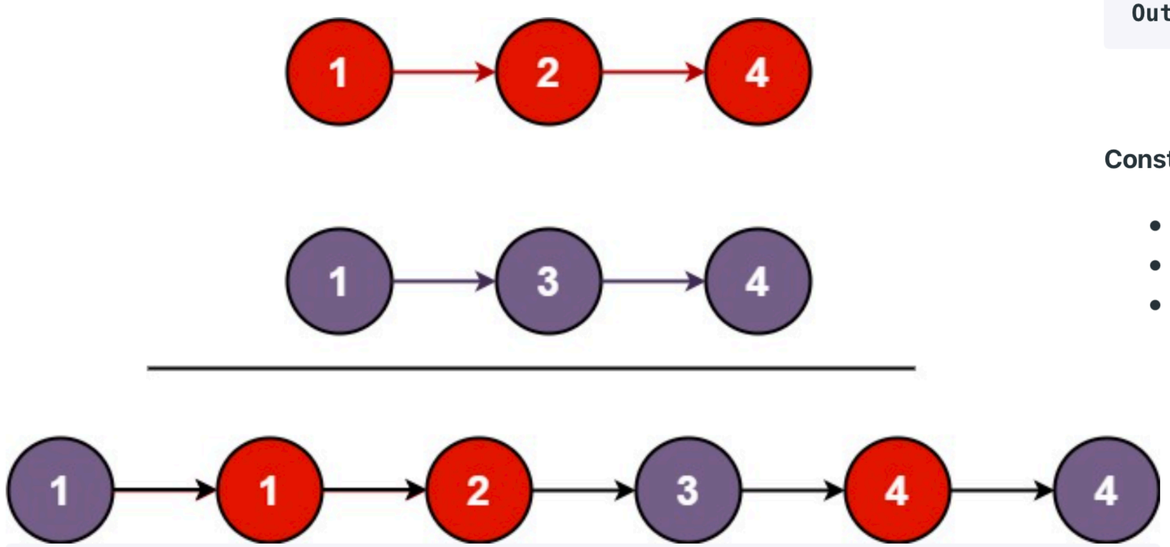
Easy 12211 1119 Add to List Share

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: []
```

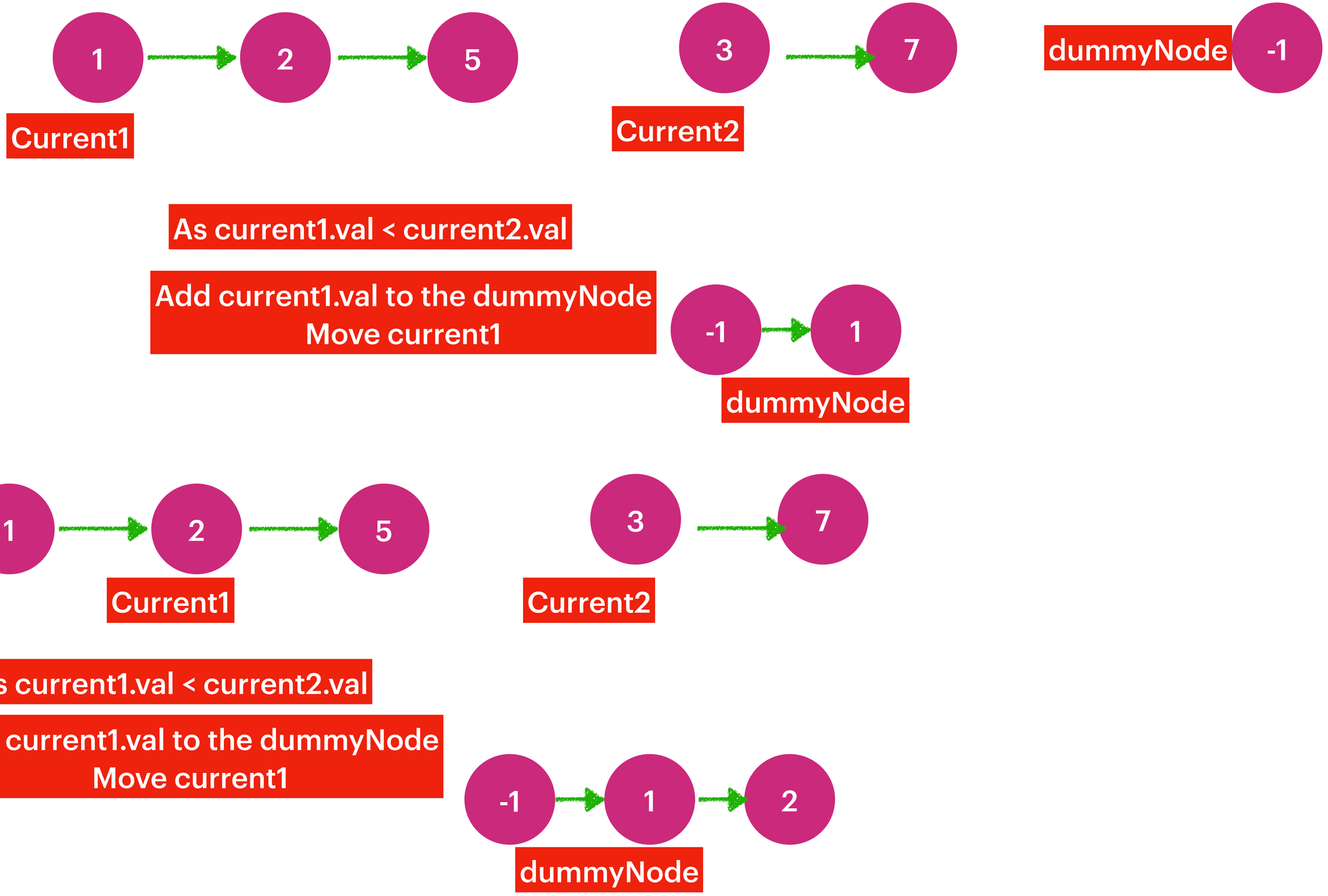
Example 3:

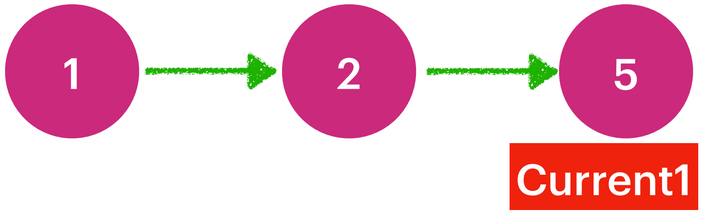
```
Input: list1 = [], list2 = [0]
Output: [0]
```

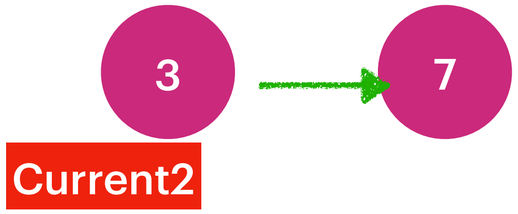
Constraints:

- 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.

After Merge

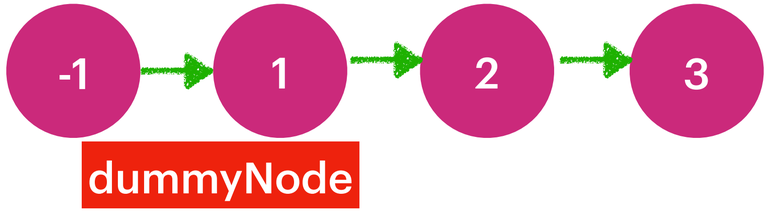

1
2
5
Current1
3
7
Current2
dummyNode
-1
As current1.val < current2.val
Add current1.val to the dummyNode
Move current1
-1
1
dummyNode
1
2
5
Current1
3
7
Current2
As current1.val < current2.val
Add current1.val to the dummyNode
Move current1
-1
1
2
dummyNode

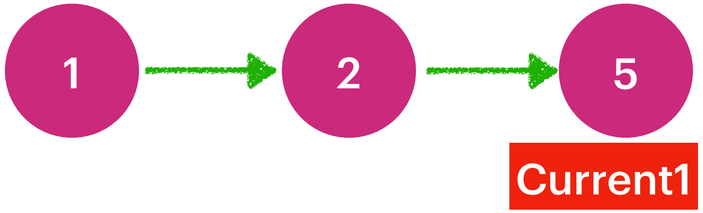


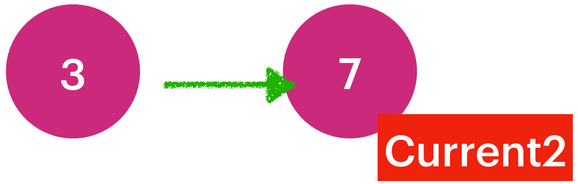


As current2.val < current1.val

Add current2.val to the dummyNode
Move current2

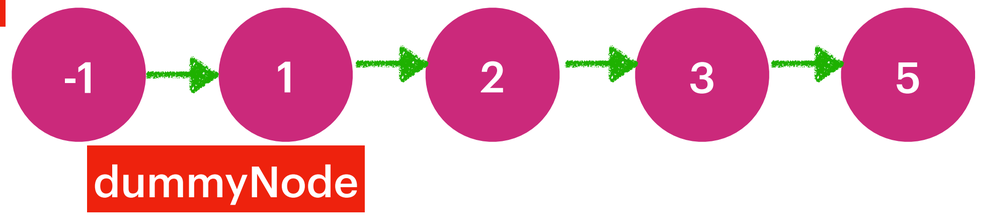


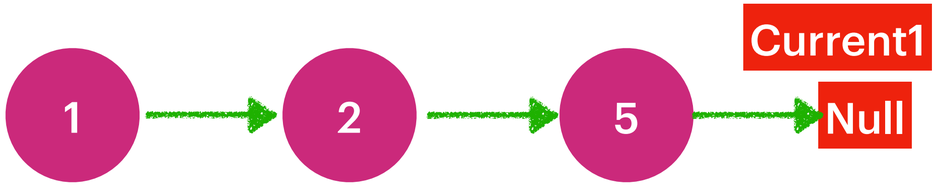


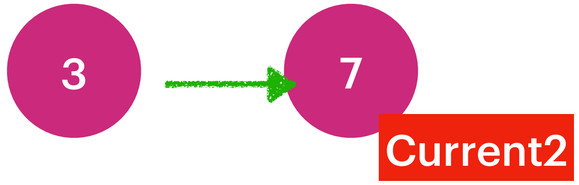


As current1.val < current2.val

Add current1.val to the dummyNode
Move current1

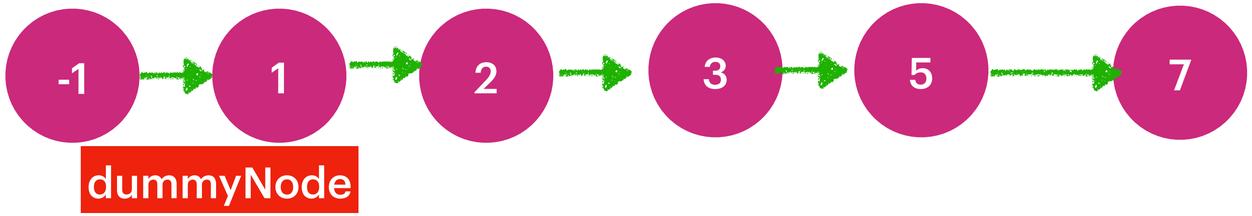




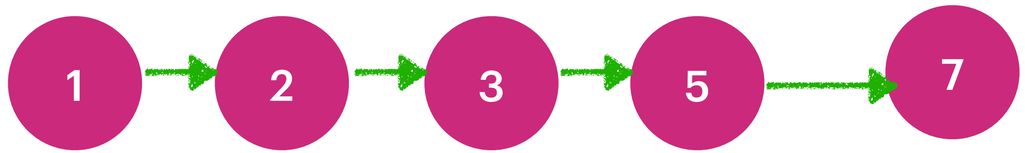


Time complexity : O(M+N)
Space Complexity : O(1)

As Current1 is null ,
Add current2 to the dummyNode



return dummyNodeHead.next;



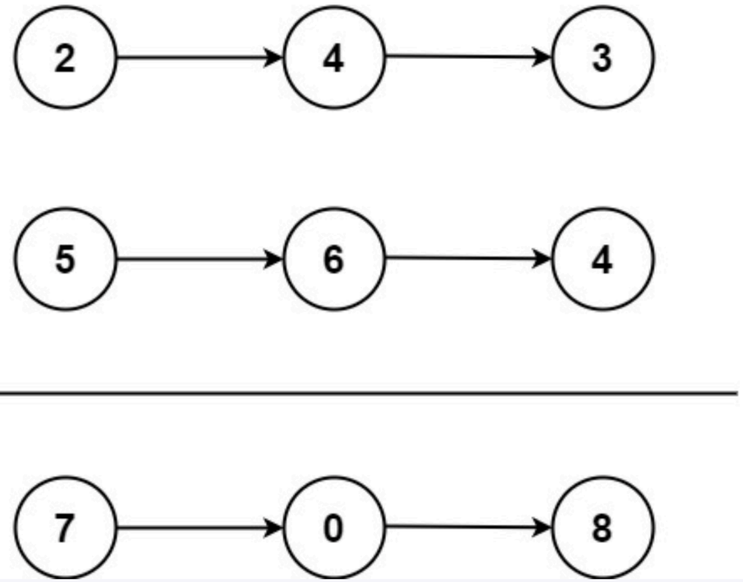
2. Add Two Numbers

Medium 18355 3755 Add to List Share

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:

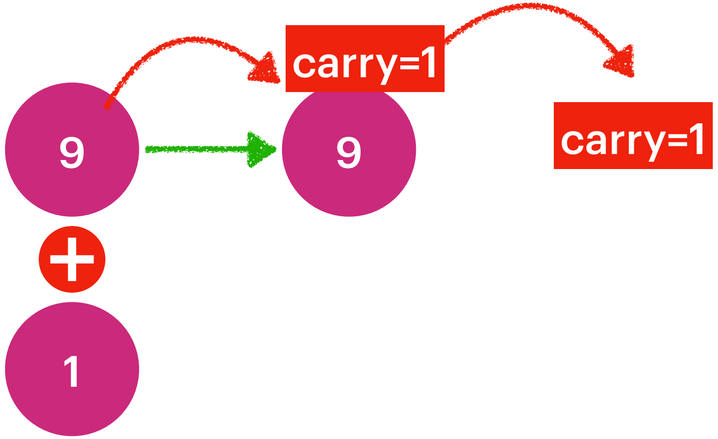
```
Input: l1 = [0], l2 = [0]
Output: [0]
```

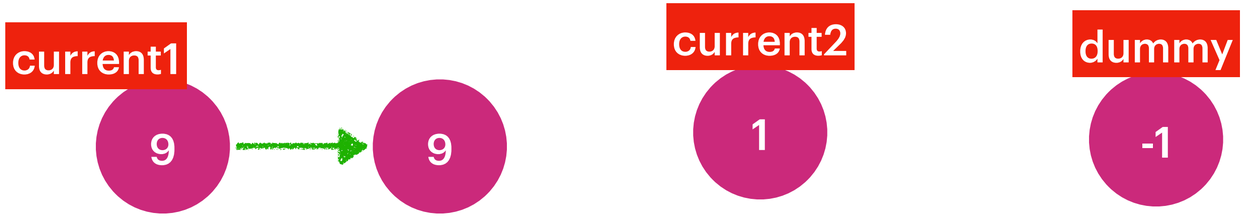
Example 3:

```
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]
```

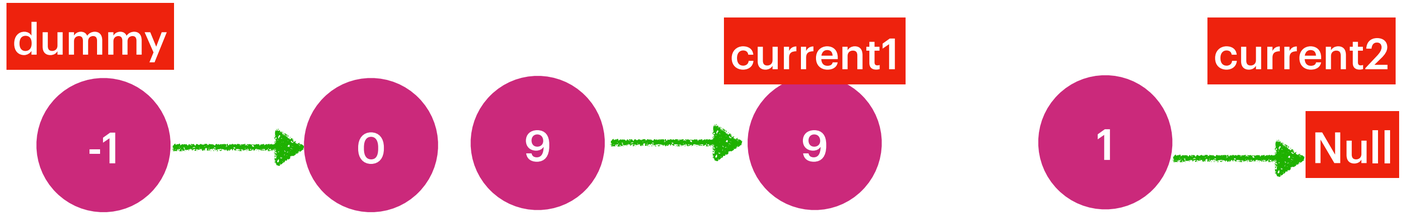
Constraints:

- 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.



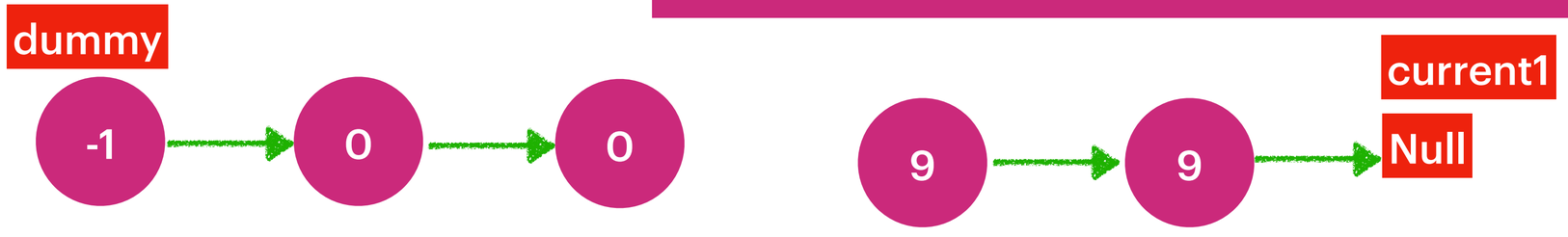


Carry = 0;
int currentSum = carry + current1.val + current2.val = 0 + 9 + 1 = 10
dummyCurrent.next = new ListNode(currentSum%10 = 0);
carry = currentSum/10 = 1;
Move current1, current2 & dummyCurrent to next

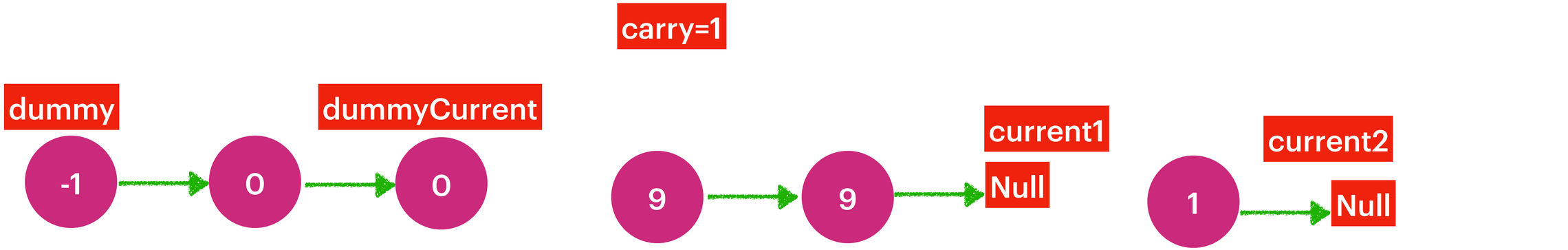


Carry = 1;
As Current2 is null then
currentSum = current1.val + carry; -> 9+1 = 10
dummyCurrent.next = new ListNode(currentSum%10 = 10%10 =0);
carry = currentSum/10 = 10/10 = 1

Move current1 & dummyCurrent to next



Time Complexity : O(max(M,N))
Space Complexity : O(1)



As current1 & current2 is null & the carry is != 0 so add carry to dummy current .

dummyCurrent.next = new ListNode(carry =1);

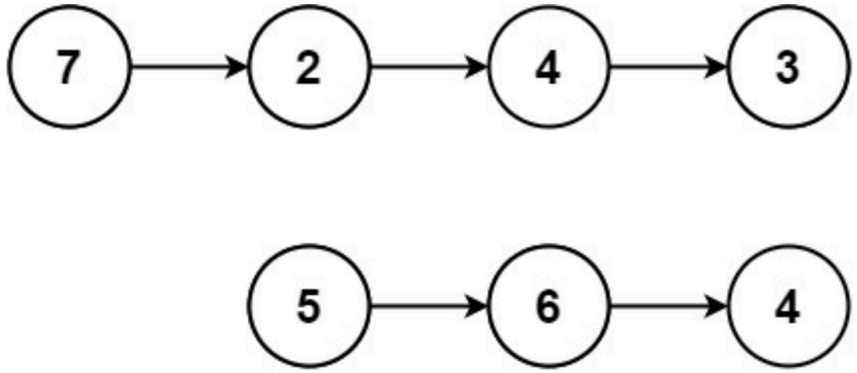
445. Add Two Numbers II

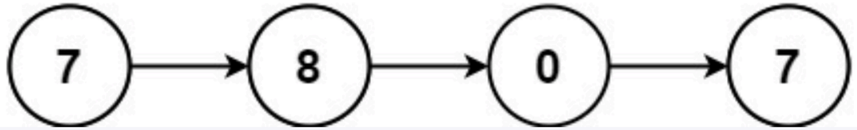
Medium 3502 225 Add to List Share

You are given two **non-empty** linked lists representing two non-negative integers. The most significant digit comes first 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

Example 1:






```
Input: l1 = [7,2,4,3], l2 = [5,6,4]
Output: [7,8,0,7]
```

Example 2:

```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [8,0,7]
```

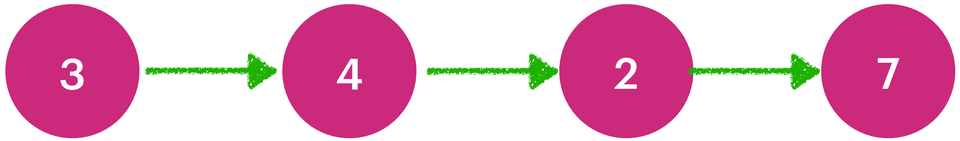
Example 3:

```
Input: l1 = [0], l2 = [0]
Output: [0]
```

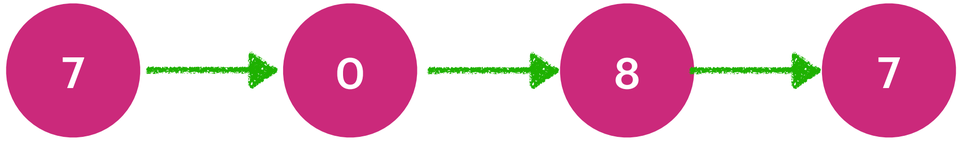
Constraints:

- 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.

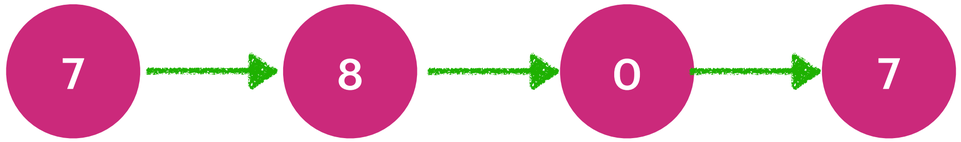
Reverse both the linked List



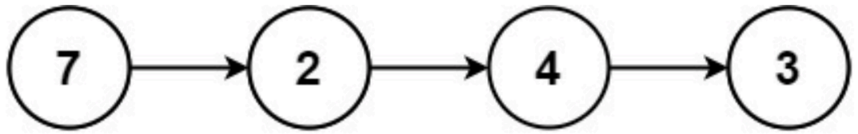
Then Do the addition.

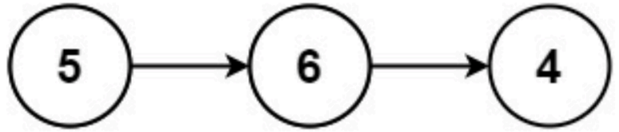


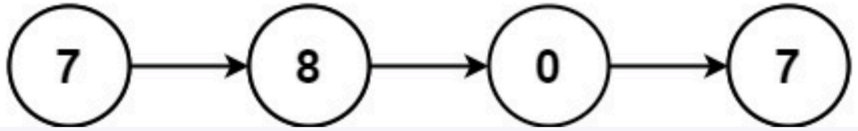
Reverse the Output



Example 1:








```
Input: l1 = [7,2,4,3], l2 = [5,6,4]
Output: [7,8,0,7]
```

Time Complexity : O(max(m,n))
Space Complexity : O(1)