Topic Introduction

Welcome back to another PrepLetter!
Today, we're diving into a classic interview topic: **Arithmetic on Number Representations**. Picture this—you're handed a number, but not in its standard integer form. Instead, you get its digits, one after another, packed in an array or linked list. Your job? Perform arithmetic, just as you would on paper, digit by digit.

Why do interviewers love these problems?
Because they test your understanding of **carry-over logic**, traversal of data structures, and how you translate basic math into code. These skills are essential for handling real-world data, which is often chunked, streamed, or not conveniently packed as integers.

When should you use these techniques?

- When numbers are too large for built-in types.
- When digits are stored in lists or arrays.
- When the problem asks for digit-wise manipulation (think adding, multiplying, or incrementing).

How does it work?
Let's say you have two numbers:
`[2, 4, 3]` and `[5, 6, 4]` (representing 342 and 465, in reverse order).
You add them digit by digit from the least significant, carry when required, and build your answer.

Quick Example:
Add [2, 4, 3] and [5, 6, 4]:

- 2 + 5 = 7
- 4 + 6 = 10 (write 0, carry 1)
- 3 + 4 + 1 = 8

Result: [7, 0, 8]

Simple, but the devil is in the details!
Ready? Let's explore three classic problems that build on this theme, each with its own twist.

Problem 1: Plus One

LeetCode 66: Plus One

Problem Statement (in plain English):

Given an array of digits representing a non-negative integer, add 1 to the integer and return the resulting digits as an array. The most significant digit is at the front, and each element contains a single digit.
You may assume the integer does not contain any leading zeros (except zero itself).

Example:
Input: `[1, 2, 3]`

Output: `[1, 2, 4]`

(123 + 1 = 124)

What's the catch?

If the addition causes a carry past the most significant digit, you need to add a new digit at the start!

Example:

Input: `[9, 9, 9]`

Output: `[1, 0, 0, 0]`

(999 + 1 = 1000)

Let's walk through a key example:

Input: `[4, 3, 9]`

Step by step:

- Add 1 to the last digit: 9 + 1 = 10, so write 0, carry 1.
- Next digit: 3 + 1 = 4 (no carry), so write 4.
- Next digit: 4 (no carry), so write 4.

Result: `[4, 4, 0]`

Try this one on paper:

Input: `[2, 9, 9]`

What should the output be?

Take a moment to try solving this on your own before reading the solution.

Solution Logic

The main idea:

- Traverse the digits from the end (least significant) to the start.
- Add 1 to the last digit.
- If a digit becomes 10, set it to 0 and carry over 1 to the next digit.
- If there's still a carry after the first digit, insert 1 at the beginning.

It's essentially simulating how you do addition by hand.

Python Code:

```python
def plusOne(digits):
    n = len(digits)
    for i in range(n - 1, -1, -1):
        if digits[i] < 9:
            digits[i] += 1
            return digits  # No carry, done!
            digits[i] = 0  # Set to 0, carry will go to next digit
    # If we finished loop, it means all digits were 9
    return [1] + digits
```

Time Complexity: O(n)

Space Complexity: O(1) (ignoring input/output size; modifies in place except, at most, adding one digit at front)

How does this work for `[2, 9, 9]`?

- 9 + 1 = 10 => 0, carry 1
- 9 + 1 = 10 => 0, carry 1
- 2 + 1 = 3

Result: `[3, 0, 0]`

Try out `[9, 8, 9]` on your own and see if you get `[9, 9, 0]`.

Did you know this could also be solved by converting the digits to a number, adding one, and splitting back?
Try that approach for fun after reading this article—but beware of overflow for very large numbers!

Let's level up and see what happens when our digits are stored in a linked list instead of an array.

Problem 2: Add Two Numbers

LeetCode 2: Add Two Numbers

Problem Statement (rephrased):

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

Example:
Input:
l1 = [2, 4, 3] (represents 342)
l2 = [5, 6, 4] (represents 465)
Output: [7, 0, 8] (342 + 465 = 807)

What's the difference from Plus One?

- Now, you're adding two numbers (not just one).
- The digits are stored in reverse order in linked lists, not arrays.

Let's step through an example:

l1 = [9, 9, 9] (999)
l2 = [1] (1)

- 9 + 1 = 10 (write 0, carry 1)
- 9 + 0 + 1 = 10 (write 0, carry 1)
- 9 + 0 + 1 = 10 (write 0, carry 1)
- No more digits, but we have a carry: write 1

Result: [0, 0, 0, 1]

Try this one on paper:

l1 = [2, 4, 9]

l2 = [5, 6, 4]

What will be the output?

Take a moment to try solving this on your own before reading the solution.

Solution Logic

It's almost the same as the manual addition you did as a kid, but digit by digit using linked lists:

- Initialize a dummy node to build the result list.
- Traverse both linked lists node by node.
- Add corresponding digits and carry.
- If a list is shorter, use 0 for its digit.
- Continue until both lists are done and there is no carry.

Python Code:

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def addTwoNumbers(l1, l2):
    dummy = ListNode()
    current = dummy
    carry = 0

    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0  # Use 0 if l1 is None
        val2 = l2.val if l2 else 0  # Use 0 if l2 is None
        total = val1 + val2 + carry

        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next
        if l1: l1 = l1.next
        if l2: l2 = l2.next

    return dummy.next
```

Time Complexity: O(max(m, n)), where m and n are the lengths of the two lists.
Space Complexity: O(max(m, n)), for the output list.

How does this work for l1 = [2, 4, 9], l2 = [5, 6, 4]?

- 2 + 5 = 7 (no carry)
- 4 + 6 = 10 (write 0, carry 1)
- 9 + 4 + 1 = 14 (write 4, carry 1)
- No more digits, but carry is 1: write 1

Result: [7, 0, 4, 1]

Try dry-running the code with l1 = [1, 8], l2 = [0]. What do you get?

Did you know this could also be solved by first converting the lists to numbers, adding, and then converting back?
Give that a try after reading this article!
But for large numbers, the approach we used (digit-wise) is safer and avoids integer overflow.

Now, let's see what happens when the digits are stored **in forward order**.

Problem 3: Add Two Numbers II

LeetCode 445: Add Two Numbers II

Problem Statement (in simple terms):

You're given two non-empty linked lists representing two non-negative integers. The most significant digit comes first (forward order), and each node contains a single digit.
Add the two numbers and return the sum as a linked list.

Example:
l1 = [7, 2, 4, 3] (represents 7243)
l2 = [5, 6, 4] (represents 564)
Output: [7, 8, 0, 7] (7243 + 564 = 7807)

What's new compared to Add Two Numbers?

- The digits are now in **forward order**—the first node is the most significant digit.

This makes things trickier, because you can't just add from least significant to most significant without reversing the lists.

Walkthrough Example:

l1 = [9, 9, 9] (999)

l2 = [1] (1)

- 999 + 1 = 1000
- Output should be [1, 0, 0, 0]

Try this one on paper:

l1 = [2, 4, 9]

l2 = [5, 6, 4]

What will be the output?

Take a moment to try solving this on your own before reading the solution.

Solution Logic

Since we need to add from the least significant digit, but our lists are in forward order, our main technique is to **reverse the lists first**. Once reversed, we can process them just like in Problem 2.

Steps:

- Reverse both input lists.
- Use the same logic as Add Two Numbers.
- Reverse the result list to restore forward order.

Python Code:

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverseList(head):
    prev = None
    while head:
        next_node = head.next
        head.next = prev
        prev = head
        head = next_node
    return prev

def addTwoNumbersII(l1, l2):
    # Step 1: Reverse both lists
    l1 = reverseList(l1)
    l2 = reverseList(l2)

    # Step 2: Add as in Problem 2
    dummy = ListNode()
    current = dummy
    carry = 0
    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        total = val1 + val2 + carry
        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next
        if l1: l1 = l1.next
        if l2: l2 = l2.next

    # Step 3: Reverse the result
    return reverseList(dummy.next)
```

Time Complexity: O(m + n), where m and n are the lengths of the input lists.

Space Complexity: O(m + n), for the output.

How does this work for l1 = [2, 4, 9], l2 = [5, 6, 4]?

- Reverse l1: [9, 4, 2]
- Reverse l2: [4, 6, 5]
- Add:
  - 9 + 4 = 13 (write 3, carry 1)
  - 4 + 6 + 1 = 11 (write 1, carry 1)
  - 2 + 5 + 1 = 8 (write 8)
- Result (reversed): [3, 1, 8]
- Reverse: [8, 1, 3]

So, the output is [8, 1, 3]

Try dry-running with l1 = [9, 9, 9], l2 = [1]. What should the output be?

While you could use stacks to avoid reversing the lists (by simulating the addition from the end), for this article, reversing is more relevant to understanding the core pattern. Want to try the stack approach? Give it a shot after reading!

Summary and Next Steps

These three problems are grouped together because they're all about **digit-wise arithmetic on number representations**—whether in arrays or linked lists, forward or reverse order. The key is carefully simulating the addition process, handling carries, and being mindful of the data structure's traversal direction.

Key Patterns:

- Traversing from least to most significant digit (may require reversing or using stacks).

- Handling carries correctly at each step.
- Creating new nodes or array elements for overflows.
- Being comfortable with both array and linked list manipulations.

Common Mistakes:

- Forgetting to handle a carry after the last digit.
- Mixing up digit order (forward vs reverse).
- Not handling different-length numbers properly.
- Modifying input lists when not allowed.

Action List:

- Solve all three problems yourself, ideally with pen and paper before coding.
- Try alternative approaches—converting to integers, using stacks, or recursion.
- Read others' solutions to see different styles and edge case handling.
- Practice similar problems (like multiplying two numbers represented as arrays/lists).
- If you get stuck, that's fine! Learning comes from practice and persistence.

Final tip:

When faced with arithmetic on digits in data structures, think:

- How do I process these digits in the correct order?
- How do I handle carries?
- What structure do I need for my result?

You'll see these patterns pop up again and again.
Keep practicing, and see you in the next PrepLetter!