Day 23 / 100 :

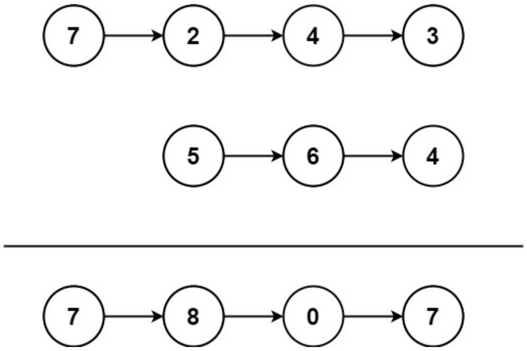
Topic - Linked List,

1️⃣ Problem statement: Add two number II (Medium)

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 0 itself.

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]

Solutions :
Approach 1 - Brute Force

1. The brute force solution starts by pushing the digits of l1 and l2 into two separate stacks (num1 and num2) in reverse order.
2. We then iterate through both stacks simultaneously, popping the digits and calculating their sum along with the carry value.
3. We create a new node for each digit of the sum and link it to the result linked list.

Finally, we return the result linked list.

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    stack<int> num1, num2;

    // Push digits of l1 into num1 stack
    while (l1) {
        num1.push(l1->val);
        l1 = l1->next;
    }

    // Push digits of l2 into num2 stack
    while (l2) {
        num2.push(l2->val);
        l2 = l2->next;
    }

    int carry = 0;
    ListNode* result = nullptr;

    // Calculate sum of digits and create new linked list
    while (!num1.empty() || !num2.empty() || carry > 0) {
        int sum = carry;
        if (!num1.empty()) {
            sum += num1.top();
            num1.pop();
        }
        if (!num2.empty()) {
            sum += num2.top();
            num2.pop();
        }

        carry = sum / 10;
        sum %= 10;

        ListNode* newNode = new ListNode(sum);
        newNode->next = result;
        result = newNode;
    }

    return result;
}
```

Solutions :
Approach 2 - Recursive approach

1. The optimized solution uses a recursive approach to add the numbers from the least significant digit to the most significant digit.
2. We first calculate the lengths of l1 and l2 and swap them if necessary so that l1 is always the shorter linked list.
3. We recursively call **addTwoNumbers** to add the digits starting from the least significant digit. Furthermore, we also pass the difference in lengths as a parameter to handle the carry propagation.
4. In each recursion, we calculate the sum of the current digits along with the carry. If there is a carry, we add it to the next digit in l1 (if available) using the addCarry function.
5. Finally, we handle any remaining carry and return the result.

Both the brute force and optimized solutions provide the same output, but the optimized solution has a better time complexity of O(max(N, M)), where N and M are the lengths of l1 and l2 respectively.

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    int len1 = getLength(l1);
    int len2 = getLength(l2);

    if (len1 > len2) {
        ListNode* temp = l1;
        l1 = l2;
        l2 = temp;
    }

    int diff = abs(len1 - len2);

    ListNode* result = addTwoNumbers(l1, l2, diff);
    if (result && result->val >= 10) {
        int carry = result->val / 10;
        result->val %= 10;
        ListNode* newNode = new ListNode(carry);
        newNode->next = result;
        result = newNode;
    }

    return result;
}
```

2 Problem statement: Multiply string (Medium)

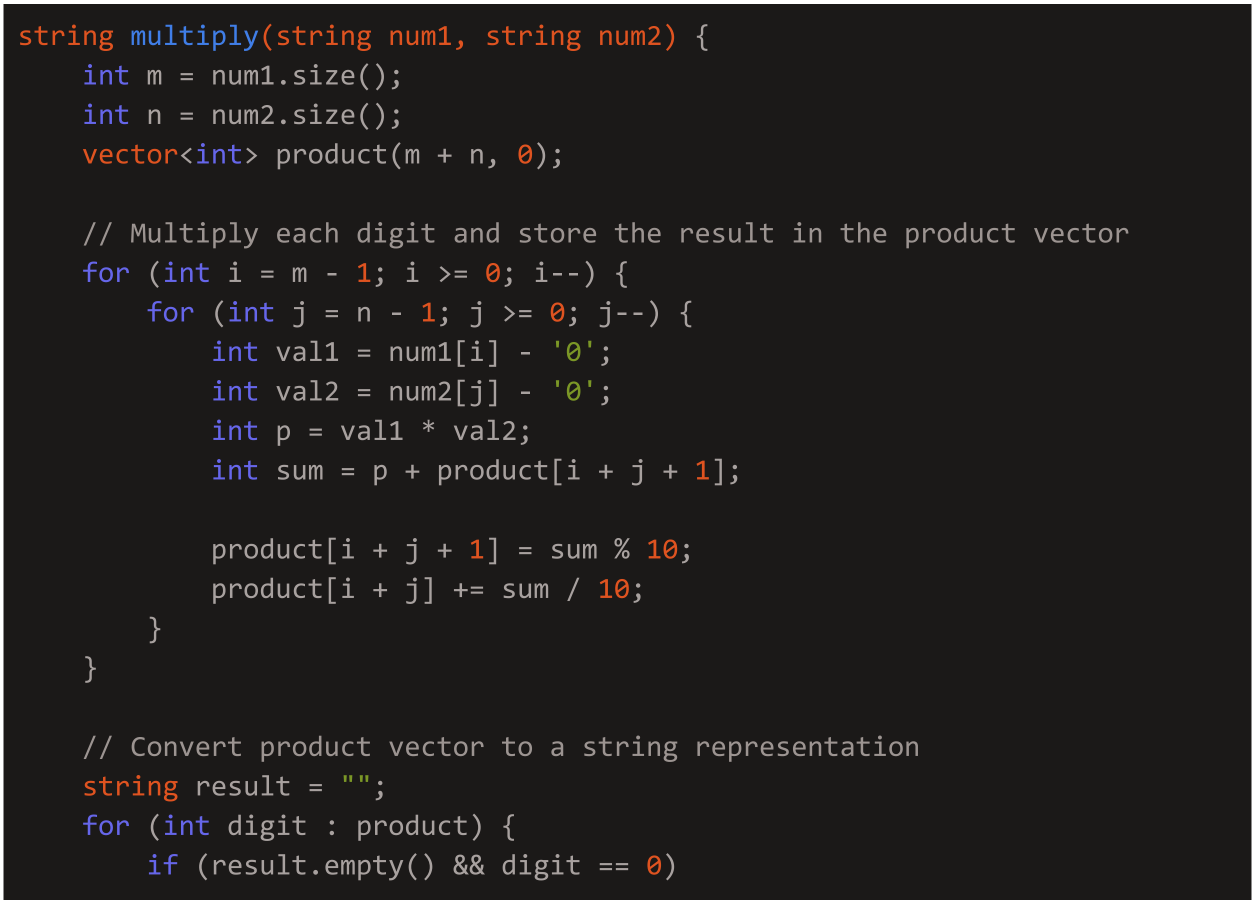
Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

Note: You must not use any built-in Big Integer library or convert the inputs to integer directly.

Example 1:
Input: num1 = "2", num2 = "3"
Output: "6"

Example 2:
Input: num1 = "123", num2 = "456"
Output: "56088"

Solutions :
Approach 1



```
string multiply(string num1, string num2) {
    int m = num1.size();
    int n = num2.size();
    vector<int> product(m + n, 0);

    // Multiply each digit and store the result in the product vector
    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            int val1 = num1[i] - '0';
            int val2 = num2[j] - '0';
            int p = val1 * val2;
            int sum = p + product[i + j + 1];

            product[i + j + 1] = sum % 10;
            product[i + j] += sum / 10;
        }
    }

    // Convert product vector to a string representation
    string result = "";
    for (int digit : product) {
        if (result.empty() && digit == 0)
```

```
            continue;
        result += to_string(digit);
    }

    return result.empty() ? "0" : result;
}
```

1. We create a product vector of size m + n to store the partial products. The maximum length of the product will be the sum of the lengths of num1 and num2.
2. We iterate through each digit of num1 and num2 in reverse order, starting from the least significant digits.
3. For each pair of digits, we calculate their product p and add it to the corresponding position in the product vector. We also consider the carry-over from previous positions.
4. After calculating all the partial products, we convert the product vector into a string representation. We skip leading zeros by checking if the result is empty and the current digit is zero.
5. Finally, we return the result.

This solution has a time complexity of O(m * n), where m and n are the lengths of num1 and num2 respectively.