1.

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

**Solution: Use the map{}, dic in python**

2.

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.

**Example:**

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

**Output:** 7 -> 0 -> 8

**Explanation:** 342 + 465 = 807.

3.

Given a string, find the length of the **longest substring** without repeating characters.

**Example 1:**

**Input:** "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

**Example 2:**

**Input:** "bbbbb"

**Output:** 1

**Explanation:** The answer is "b", with the length of 1.

**Example 3:**

**Input:** "pwwkew"

**Output:** 3

**Explanation:** The answer is "wke", with the length of 3.

Note that the answer must be a **substring**, "pwke" is a *subsequence* and not a substring.

**Solution:**

Need 3 temporary variables to find the longest substring: start, maxLength,

and usedChars.

Start by walking through string of characters, one at a time.

Check if the current character is in the usedChars map, this would mean we

have already seen it and have stored it's corresponding index.

If it's in there and the start index is <= that index, update start

to the last seen duplicate's index+1. This will put the start index at just

past the current value's last seen duplicate. This allows us to have the

longest possible substring that does not contain duplicates.

If it's not in the usedChars map, we can calculate the longest substring

seen so far. Just take the current index minus the start index. If that

value is longer than maxLength, set maxLength to it.

Finally, update the usedChars map to contain the current value that we just

finished processing.

**4. Median of Two Sorted Arrays**

Hard

6053919Add to ListShare

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

**Example 1:**

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

**Example 2:**

nums1 = [1, 2]

nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5