**Arrays**

**Single Number**

Given a **non-empty** array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

A screenshot of a computer

Description automatically generated with medium confidence

Notes: <https://www.youtube.com/watch?v=qMPX1AOa83k>

Using XOR works because:

1. XOR of a number with itself is 0
2. XOR of a number with 0 is the number itself

1 ^ 1 = 0

0 ^ 0 = 0

1 ^ 0 = 1

0 ^ 1 = 1

A paper with writing on it

Description automatically generated with low confidence

**Remove Duplicates from Sorted Array**

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in*nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

* Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
* Return k.

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

**Explanation:** Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Key to this problem:

Use 2 pointers: left pointer to mark where to put number and right pointer to iterate through array

A screenshot of a computer code

Description automatically generated with low confidence

<https://www.youtube.com/watch?v=DEJAZBq0FDA>

**Best Time to Buy and Sell Stock II**

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return the***maximum***profit you can achieve.

Key to problem:

Draw picture for visualization. You want to buy low and sell high, so you’ll always be looking at the price before the current price for comparison

A graph on a piece of paper

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with low confidence

**Rotate Array**

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

**Input:** nums = [1,2,3,4,5,6,7], k = 3

**Output:** [5,6,7,1,2,3,4]

Key to problem:

Use reversal to help rotate in place

1. Reverse full array -> [7,6,5,4,3,2,1]
2. Reverse first k elements -> [5,6,7,3,2,1]
3. Reverse rest of array -> [5,6,7,1,2,3]

Because you’ll be using 3 reversals, write helper function to avoid copying the same code 3 times.

<https://www.youtube.com/watch?v=BHr381Guz3Y>

A screenshot of a computer program

Description automatically generated with medium confidence

**Contains Duplicates**

Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

**Input:** nums = [1,1,1,3,3,4,3,2,4,2]

**Output:** true

Key to problem:

Sort the list first then compare element to its neighbor to see if they are duplicates. There is only one pass through the array.

Time complexity O(nlogn) since we sorted

Space O(1)

A screenshot of a computer code

Description automatically generated with low confidence

Another approach is to use a set and store the numbers seen. If you come across a number in the set, then return True. We iterate through list once but take up more space.

Time complexity O(n)

Space O(n)

A screenshot of a computer program

Description automatically generated with low confidence

<https://www.youtube.com/watch?v=3OamzN90kPg>

**Intersection of Two Arrays II**

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

**Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4]

**Output:** [4,9]

**Explanation:** [9,4] is also accepted.

Key to problem:

Use 2 pointers and sort the lists before iterating

A screenshot of a computer

Description automatically generated with medium confidence

**Plus One**

You are given a **large integer** represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

**Input:** digits = [1,2,3]

**Output:** [1,2,4]

**Explanation:** The array represents the integer 123.

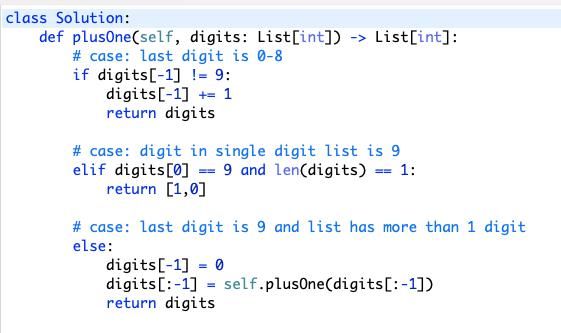
Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1,2,4].

Key to problem:

Account for carryover so [9,9,9] should be [1,0,0,0]

1. Approach problem recursively (easier to digest; same complexity as approach 2 -> O(n) time)



1. Use while loop and track carry -> <https://www.youtube.com/watch?v=jIaA8boiG1s>



**Move Zeros**

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

**Note** that you must do this in-place without making a copy of the array.

**Example 1:**

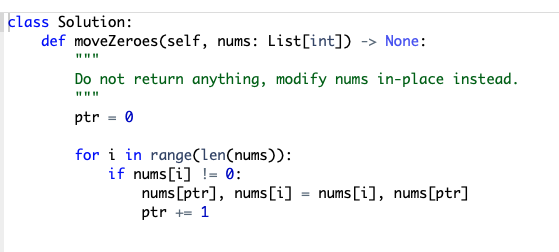
**Input:** nums = [0,1,0,3,12]

**Output:** [1,3,12,0,0]

Key to problem:

Use the 2-pointer approach –> O(n) since only run through list once

Breakdown – if we see a 0, don’t do anything. If we see a non-zero, swap this number with the number that the ptr is pointing to. Increment ptr.



<https://www.youtube.com/watch?v=aayNRwUN3Do>

**Two Sum**

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

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

You can return the answer in any order.

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

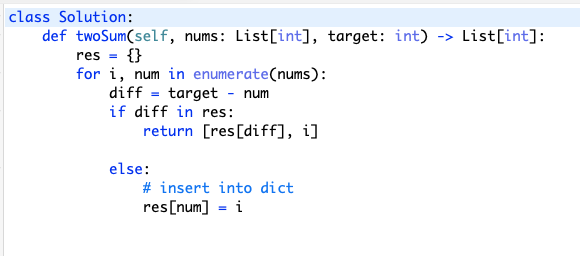
**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

Key to problem:

Use enumerate() to keep track of index and element

This method iterates through list once -> O(n) time; O(n) space

Start with empty hash map. Visit first element and compute difference between target and this element. If that difference is not in hash map, add the element visited and index to hash map. Iterate through list adding elements until we reach a difference that **is** in hash map. The solution is the index of this element (which is in the hash map) plus the index of element we are currently on (i).



<https://www.youtube.com/watch?v=KLlXCFG5TnA>

**Valid Sudoku**

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

**Note:**

* A Sudoku board (partially filled) could be valid but is not necessarily solvable.
* Only the filled cells need to be validated according to the mentioned rules.

**Input:** board =

[["5","3",".",".","7",".",".",".","."]

,["6",".",".","1","9","5",".",".","."]

,[".","9","8",".",".",".",".","6","."]

,["8",".",".",".","6",".",".",".","3"]

,["4",".",".","8",".","3",".",".","1"]

,["7",".",".",".","2",".",".",".","6"]

,[".","6",".",".",".",".","2","8","."]

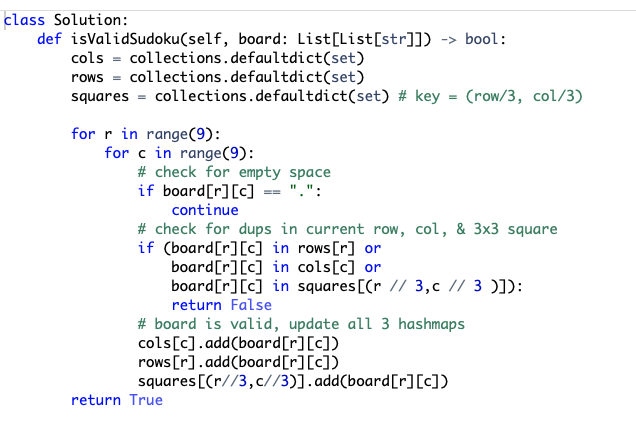
,[".",".",".","4","1","9",".",".","5"]

,[".",".",".",".","8",".",".","7","9"]]

**Output:** true

Key to problem:

Use hash set. Watch video on this one -> <https://www.youtube.com/watch?v=TjFXEUCMqI8>



**Rotate Image**

You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

**Example 1:**

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[7,4,1],[8,5,2],[9,6,3]]

Pulled this from discussion solution. **Come back to this later.**

