**11. Container with Most Water**

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

**Notice** that you may not slant the container.



**Input:** height = [1,8,6,2,5,4,8,3,7]

**Output:** 49

**Explanation:** The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

**Sol:**

Take 2 pointer, start will point to 0 and end will point to end of the array. Find the min of the hight of the pillars and the width between pillar will be end – start. Find the maxWater by using formula minHeight \* width. If height[start] > height[end] then do end-- else start++

**525. Contiguous Array**

Given a binary array nums, return *the maximum length of a contiguous subarray with an equal number of*0*and*1.

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

**Output:** 2

**Explanation:** [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

**Sol:**

Instead of 0 mark it to -1…now find the longest subarray with sum 0.

**523. Continuous Subarray Sum**

Given an integer array nums and an integer k, return true *if*nums*has a continuous subarray of size****at least two****whose elements sum up to a multiple of* k*, or*false*otherwise*.

An integer x is a multiple of k if there exists an integer n such that x = n \* k. 0 is **always** a multiple of k.

**Input:** nums = [23,2,4,6,7], k = 6

**Output:** true

**Explanation:** [2, 4] is a continuous subarray of size 2 whose elements sum up to 6.

**Sol:**

Multiplication of 6 is 0, 6, 12, 18…

If sum%6 == 0 it means, sum is multiplication of 6. So we can search in map of remainder zero.

Similar to previous problem. Only diff is we need to find the subarray of length >= 2

**1248. Count Number of Nice Subarrays**

Given an array of integers nums and an integer k. A continuous subarray is called **nice** if there are k odd numbers on it.

Return the number of ***nice*** sub-arrays.

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

**Output:** 2

**Explanation:** The only sub-arrays with 3 odd numbers are [1,1,2,1] and [1,2,1,1].

Sol:

**791. Custom Sort String**

You are given two strings order and s. All the characters of order are **unique** and were sorted in some custom order previously.

Permute the characters of s so that they match the order that order was sorted. More specifically, if a character x occurs before a character y in order, then x should occur before y in the permuted string.

Return *any permutation of*s*that satisfies this property*.

**Input:** order = "cba", s = "abcd"

**Output:** "cbad"

**Explanation:**

"a", "b", "c" appear in order, so the order of "a", "b", "c" should be "c", "b", and "a".

Since "d" does not appear in order, it can be at any position in the returned string. "dcba", "cdba", "cbda" are also valid outputs.

**Sol:**

Count the freq of each char of s string.

Then iterate the order string and if count > 0 of that char then append that char into Stringbuilder and decrement the count.

Once we come out of the loop only char which is not common in s String will be remaining.

Then append that char to string builder.

**442. Find All Duplicates in an Array**

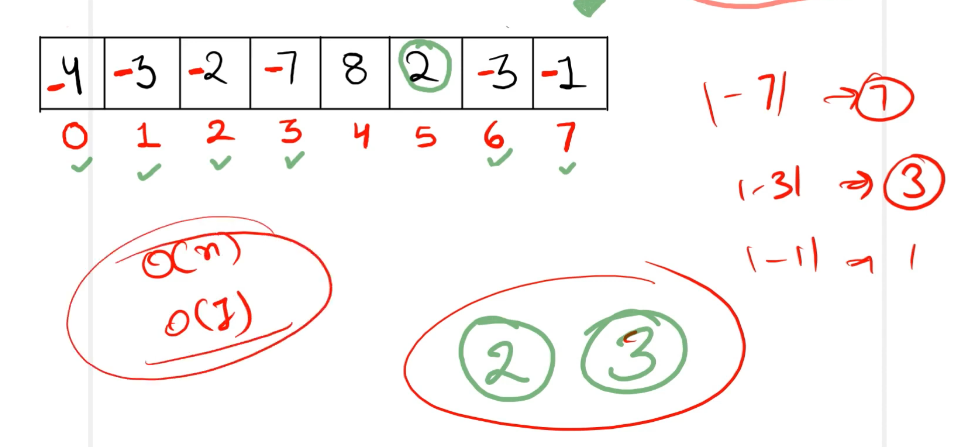
Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears****twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

**Input:** nums = [4,3,2,7,8,2,3,1]

**Output:** [2,3]

**Sol:**



Iterate the array and find nums[i] – 1 as index and check if nums[index] < 0

If it is -ve then add this to result.

Else negate the nums[i]

**287. Find the Duplicate Number**

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

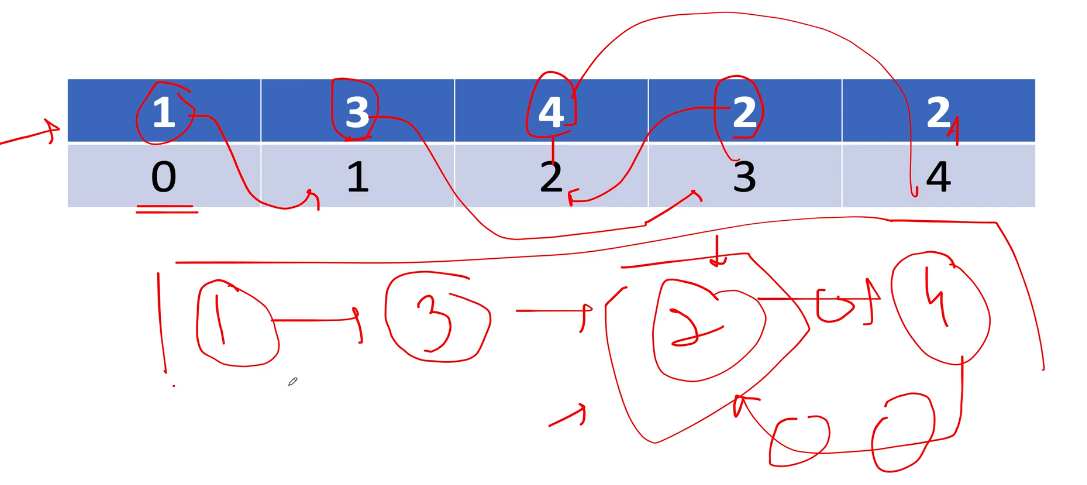
There is only **one repeated number** in nums, return *this repeated number*.

You must solve the problem **without** modifying the array nums and uses only constant extra space.

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

**Output:** 2

**Sol:**



We start from 0th index whose value is 1….create a node(1)

Now go to index 1 whose value is 3 …..create a node(3)

Now go to index 3 whose value is 2……create a node(2)

Now go to index 2 whose value is 4…..create a node(4)

Now go to index 4 whose value is 2……it forms a cycle.

And start node of the cycle is duplicate number.

We will take 2 pointers slow and fast, both starting from nums[0]…..its same finding the start node of the loop in linkedlist.

**18. 4Sum**

Given an array nums of n integers, return *an array of all the****unique****quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that:

* 0 <= a, b, c, d < n
* a, b, c, and d are **distinct**.
* nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in **any order**.

**Input:** nums = [1,0,-1,0,-2,2], target = 0

**Output:** [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Sol:

First sort the array. then

for (int i = 0; i < n; i++) {

int target\_3 = target - nums[i];

for (int j = i + 1; j < n; j++) {

int target\_2 = target\_3 - nums[j];

And then search the remaining array whose sum == target\_2

**491. Increasing Subsequences**

Given an integer array nums, return all the different possible increasing subsequences of the given array with **at least two elements**. You may return the answer in **any order**.

The given array may contain duplicates, and two equal integers should also be considered a special case of increasing sequence.

**Input:** nums = [4,6,7,7]

**Output:** [[4,6],[4,6,7],[4,6,7,7],[4,7],[4,7,7],[6,7],[6,7,7],[7,7]]

**Sol:**

Use backtracking

**55. Jump Game**

You are given an integer array nums. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return true*if you can reach the last index, or*false*otherwise*.

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

**Output:** true

**Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Sol:**

**45. Jump Game II**

Given an array of non-negative integers nums, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

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

**Output:** 2

**Explanation:** The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Sol:**

**128. Longest Consecutive Sequence**

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

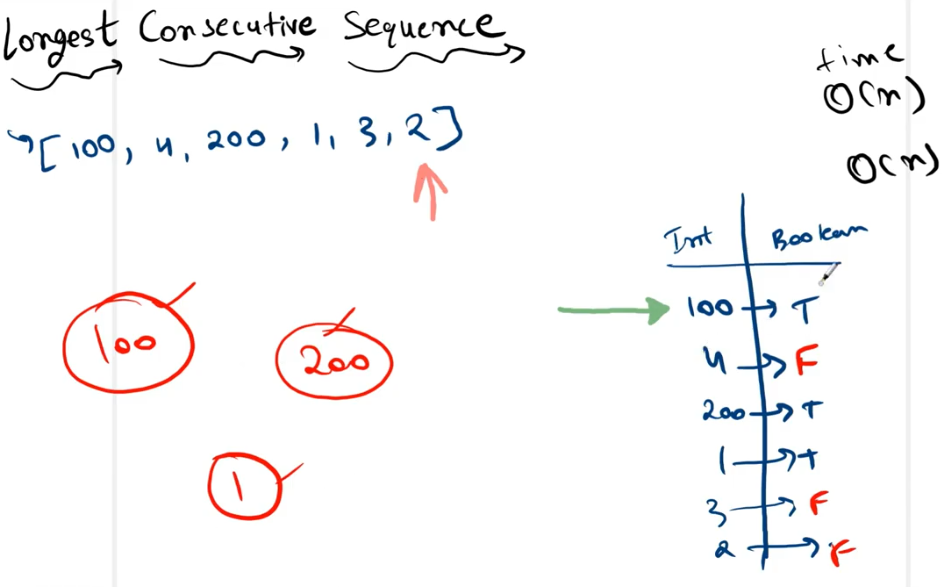
You must write an algorithm that runs in O(n) time.

**Input:** nums = [100,4,200,1,3,2]

**Output:** 4

**Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore, its length is 4.

**Sol:**



Take a map of int and Boolean type. Initially insert all the element to map as every element itself is a increasing seq.

Now check if map contains nums[i] – 1 if it contains then update that num to false as why we should start from this num as we have smaller number to start.

After doing above step we will have entry whose value is true and that will be the starting point of the sequence.

Now just find the max length of the sequence.

**229. Majority Element II**

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

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

**Output:** [3]

Sol:

Same way as majority element but here we will get 2 element as majority element so we will take candidate1 and candidate2

**152. Maximum Product Subarray**

Given an integer array nums, find a contiguous non-empty subarray within the array that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

A **subarray** is a contiguous subsequence of the array.

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

**Output:** 6

**Explanation:** [2,3] has the largest product 6.

Sol:

**53. Maximum Subarray**

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

**Input:** nums = [-2,1,-3,4,-1,2,1,-5,4]

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.

Sol:

Use kadane algo

**918. Maximum Sum Circular Subarray**

Given a **circular integer array** nums of length n, return *the maximum possible sum of a non-empty****subarray****of*nums.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].

A **subarray** may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.

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

**Output:** 3

**Explanation:** Subarray [3] has maximum sum 3.

Sol:

**56. Merge Intervals**

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

**Input:** intervals = [[1,3],[2,6],[8,10],[15,18]]

**Output:** [[1,6],[8,10],[15,18]]

**Explanation:** Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Sol:

Sort the array and do the remaining operation

**209. Minimum Size Subarray Sum**

Given an array of positive integers nums and a positive integer target, return the minimal length of a **contiguous subarray** [numsl, numsl+1, ..., numsr-1, numsr] of which the sum is greater than or equal to target. If there is no such subarray, return 0 instead.

**Input:** target = 7, nums = [2,3,1,2,4,3]

**Output:** 2

**Explanation:** The subarray [4,3] has the minimal length under the problem constraint.

**Sol:**

Use sliding window

**31. Next Permutation**

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

* For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer

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

**Output:** [1,3,2]

**Sol:**

**238. Product of Array Except Self**

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

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

**Output:** [24,12,8,6]

**Sol:**

**2149. Rearrange Array Elements by Sign**

You are given a **0-indexed** integer array nums of **even** length consisting of an **equal** number of positive and negative integers.

You should **rearrange** the elements of nums such that the modified array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in nums is **preserved**.
3. The rearranged array begins with a positive integer.

Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*.

**Input:** nums = [3,1,-2,-5,2,-4]

**Output:** [3,-2,1,-5,2,-4]

**Explanation:**

The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4].

The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4].

Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.

**189. Rotate Array**

Given an array, 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]

**Explanation:**

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

**48. 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.



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

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

**74. Search a 2D Matrix**

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

* Integers in each row are sorted from left to right.
* The first integer of each row is greater than the last integer of the previous row.



**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

Sol:

**73. Set Matrix Zeroes**

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

You must do it [in place](https://en.wikipedia.org/wiki/In-place_algorithm).s



**Input:** matrix = [[1,1,1],[1,0,1],[1,1,1]]

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

Sol:

**581. Shortest Unsorted Continuous Subarray**

Given an integer array nums, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order.

Return *the shortest such subarray and output its length*.

**Input:** nums = [2,6,4,8,10,9,15]

**Output:** 5

**Explanation:** You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.

**Sol:**

Find the min and max for each element which is not in the increasing order (which is either in mountain shape or in v shape with its neighbours). And update the min, max overall. Once we get the min and max then check the position where it should be inserted to make the whole array sorted. And j – I + 1 will give the window size of the array which we need to sort.

**75. Sort Colors**

Given an array nums with n objects colored red, white, or blue, sort them [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

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

**Output:** [0,0,1,1,2,2]

**Sol:**

**54. Spiral Matrix**

Given an m x n matrix, return all elements of the matrix in spiral order.



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

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

**Sol:**

**560. Subarray Sum Equals K**

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to* k.

A subarray is a contiguous **non-empty** sequence of elements within an array.

**Input:** nums = [1,1,1], k = 2

**Output:** 2

Sol:

**974. Subarray Sums Divisible by K**

Given an integer array nums and an integer k, return the number of non-empty ***subarrays*** that have a sum divisible by k.

A **subarray** is a **contiguous** part of an array.

**Input:** nums = [4,5,0,-2,-3,1], k = 5

**Output:** 7

**Explanation:** There are 7 subarrays with a sum divisible by k = 5:

[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]

Sol:

**15. 3Sum**

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

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

**Output:** [[-1,-1,2],[-1,0,1]]

**Explanation:**

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

The distinct triplets are [-1,0,1] and [-1,-1,2].

Notice that the order of the output and the order of the triplets does not matter.

Sol:

Sort the array.

Apply loop, and inside loop use binary search to find the sum of two number.