**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]

**Output:** 2

**Explanation:** [0, 1] is the longest contiguous subarray with an 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:**

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

**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**

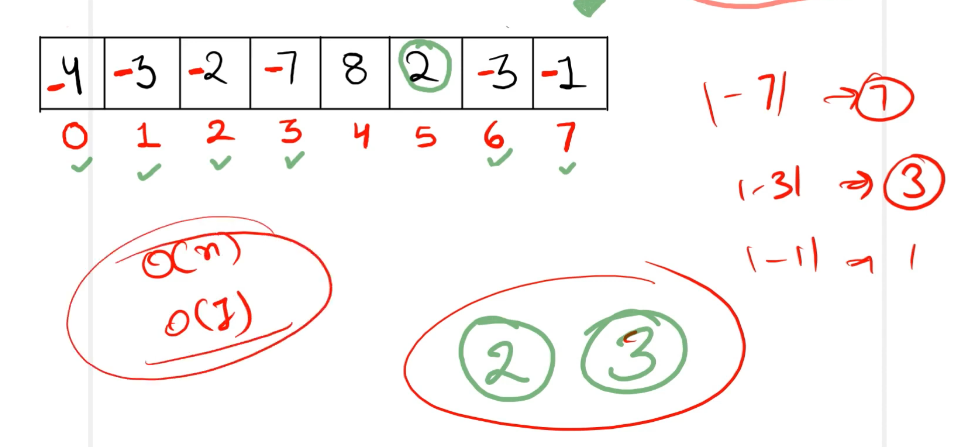
Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears****twice***.

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 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.

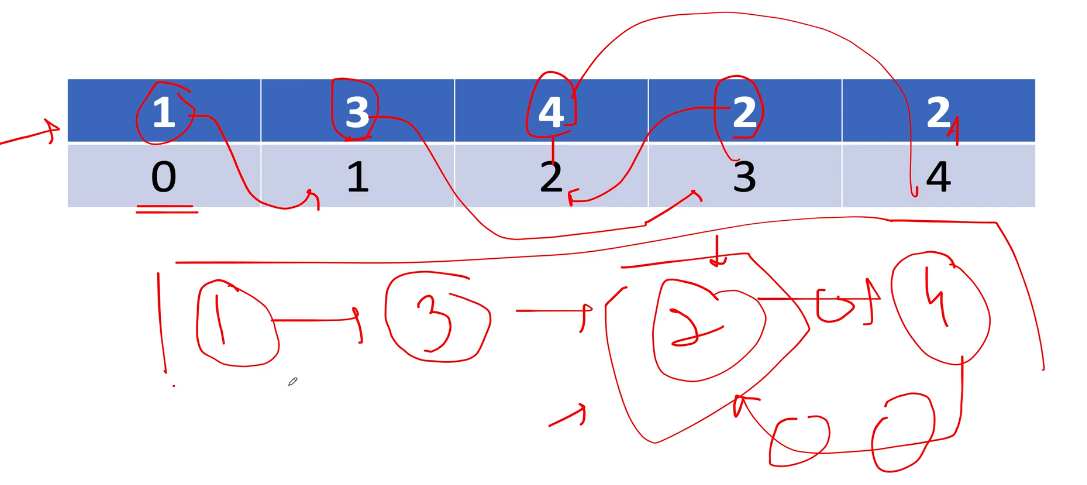
There is only **one repeated number** in nums, return *this repeated number*.

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.

**1838. Frequency of the Most Frequent Element**

The **frequency** of an element is the number of times it occurs in an array.

You are given an integer array nums and an integer k. In one operation, you can choose an index of nums and increment the element at that index by 1.

Return *the****maximum possible frequency****of an element after performing****at most***k*operations*.

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

**Output:** 3

**Explanation:** Increment the first element three times and the second element two times to make nums = [4,4,4].

4 has a frequency of 3.

**Sol:**

First sort the array so the all the closer element will be closer to each other. Then apply siliding window

While j < n

Find the sum of each number and then check

**while** (sum + k < (**long**) nums[j] \* (j - i + 1)) {

sum -= nums[i];

i++;

}

Then find the

ans = Math.*max*(ans, j - i + 1);

and increment the j counter.

**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:**

**334. Increasing Triplet Subsequence**

Given an integer array nums, return true*if there exists a triple of indices*(i, j, k)*such that*i < j < k*and*nums[i] < nums[j] < nums[k]. If no such indices exists, return false.

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

**Output:** true

**Explanation:** Any triplet where i < j < k is valid.

**Sol:**

It is similar to previous problem, here we have to check the size > 2

**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:**

Take 2 pointer left and right. Both will initially pointing to 0

While right < nums.length

Find the maximum in left and right subarray.

Then left will point to right

And right will be pointing to maxReach

If left > right in any case then return false

If maxReach >= nums.length – 1 then return true

**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:**

Its similar to previous question, here we have given that we will always reach to end. So code which is there in prev question for returning true/false…that is not required….instead do ans++

**215. Kth Largest Element in an Array**

Given an integer array nums and an integer k, return *the* kth *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

You must solve it in O(n) time complexity.

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

**Output:** 5

**Sol:**

Take a heap

**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 set and add all the nums into it.

Now iterate the array and…

Take 3 pointers…..currLen =1, left = num, right = num

While(set.contains(left-1))

curLen++;

set.remove(left-1)

left--;

do similiarly for right pointer…instead of – use + as we are cheking in right and instead of – use ++

**300. Longest Increasing Subsequence**

Given an integer array nums, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].

**Input:** nums = [10,9,2,5,3,7,101,18]

**Output:** 4

**Explanation:** The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

**Sol:**

Same as finding the increasing subsequence and then find the max length from each list.

**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]

**1004. Max Consecutive Ones III**

Given a binary array nums and an integer k, return *the maximum number of consecutive*1*'s in the array if you can flip at most* k 0's.

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

**Output:** 6

**Explanation:** [1,1,1,0,0,**1**,1,1,1,1,**1**]

Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

**Sol:**

**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.

**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:

**1679. Max Number of K-Sum Pairs**

You are given an integer array nums and an integer k.

In one operation, you can pick two numbers from the array whose sum equals k and remove them from the array.

Return *the maximum number of operations you can perform on the array*.

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

**Output:** 2

**Explanation:** Starting with nums = [1,2,3,4]:

- Remove numbers 1 and 4, then nums = [2,3]

- Remove numbers 2 and 3, then nums = []

There are no more pairs that sum up to 5, hence a total of 2 operations.

**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.

**673. Number of Longest Increasing Subsequence**

Given an integer array nums, return *the number of longest increasing subsequences.*

**Notice** that the sequence has to be **strictly** increasing.

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

**Output:** 2

**Explanation:** The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

**416. Partition Equal Subset Sum**

Given a **non-empty** array nums containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

**Input:** nums = [1,5,11,5]

**Output:** true

**Explanation:** The array can be partitioned as [1, 5, 5] and [11].

**698. Partition to K Equal Sum Subsets**

Given an integer array nums and an integer k, return true if it is possible to divide this array into k non-empty subsets whose sums are all equal.

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

**Output:** true

**Explanation:** It is possible to divide it into 4 subsets (5), (1, 4), (2,3), (2,3) with equal sums.

**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]

**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.

**540. Single Element in a Sorted Array**

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once*.

Your solution must run in O(log n) time and O(1) space.

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

**Output:** 2

**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

**167. Two Sum II - Input Array Is Sorted**

Given a **1-indexed** array of integers numbers that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.

Return*the indices of the two numbers,*index1*and*index2*,****added by one****as an integer array*[index1, index2]*of length 2.*

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

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

**Output:** [1,2]

**Explanation:** The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].