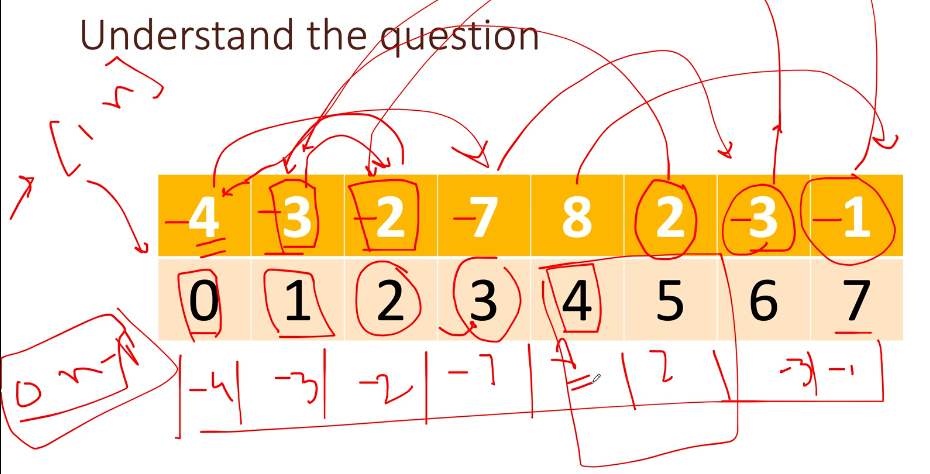
**448. Find All Numbers Disappeared in an Array**

Given an array nums of n integers where nums[i] is in the range [1, n], return *an array of all the integers in the range* [1, n] *that do not appear in* nums.

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

**Output:** [5,6]

**Sol:**



Iterate the nums and get the index by doing nums[i] – 1 and then go to that index and mark it -ve. After coming out of the loop we can see that only index 4, 5 are left which value is not -ve and its position will be 5,6 respectively.

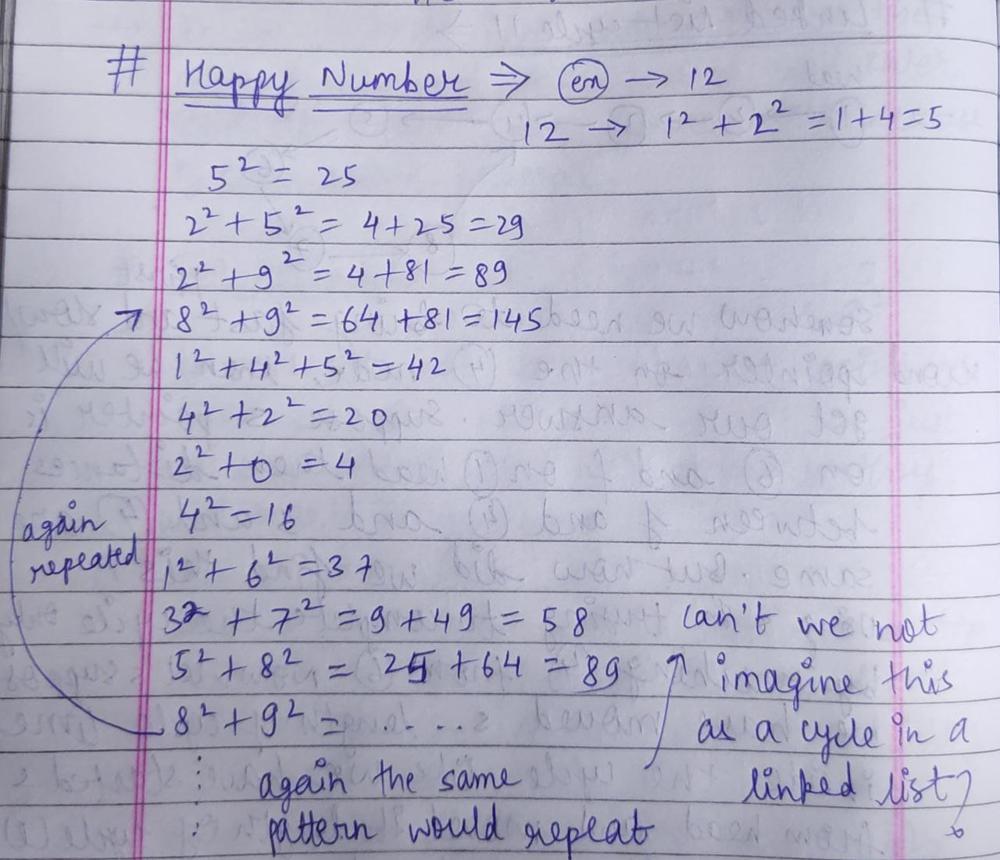
[**202. Happy Number**](https://leetcode.com/problems/happy-number/)

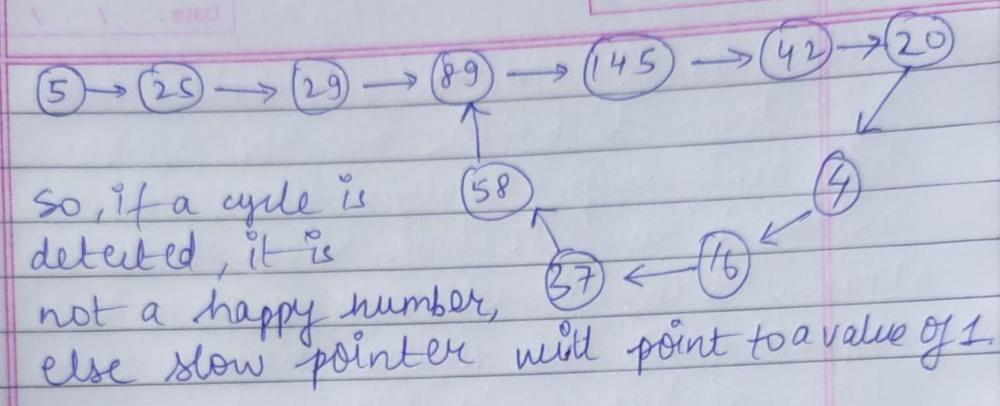
Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

* Starting with any positive integer, replace the number by the sum of the squares of its digits.
* Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
* Those numbers for which this process ends in 1 are happy.

Return true *if* n *is a happy number, and* false *if not*.





**674. Longest Continuous Increasing Subsequence**

Given an unsorted array of integers nums, return *the length of the longest****continuous increasing subsequence****(i.e. subarray)*. The subsequence must be **strictly** increasing.

A **continuous increasing subsequence** is defined by two indices l and r (l < r) such that it is [nums[l], nums[l + 1], ..., nums[r - 1], nums[r]] and for each l <= i < r, nums[i] < nums[i + 1].

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

**Output:** 3

**Explanation:** The longest continuous increasing subsequence is [1,3,5] with length 3.

Even though [1,3,5,7] is an increasing subsequence, it is not continuous as elements 5 and 7 are separated by element 4.

**Sol:**

We can use variable size sliding window here.

We will check if arr[j-1] >= arr[j] it means we got the 1st decrementing element and we will assign i = j;

then will check the max of ans and window size j – i + 1 and increment the j++

**169. Majority Element**

Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

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

**Output:** 3

**Sol:**

Use moose voting algorithm

**485. Max Consecutive Ones**

Given a binary array nums, return *the maximum number of consecutive*1*'s in the array*.

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

**Output:** 3

**Explanation:** The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3.

**88. Merge Sorted Array**

You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

**Merge** nums1 and nums2 into a single array sorted in **non-decreasing order**.

You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

**Merge** nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array*nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of

**Input:** nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

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

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

**268. Missing Number**

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

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

**Output:** 2

**Explanation:** n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

**283. Move Zeroes**

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.

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

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

**Sol:**

Iterate the array and if nums[i] != 0 then nums[index++] = nums[i].

Once we come out of the loop then we will have all non zero value at starting of the array. Then put another loop starting from i= index to arrays length and do nums[i] = 0

**GFG: Convert array into Zig-Zag fashion**

Given an array of distinct elements of size **N**, the task is to rearrange the elements of the array in a zig-zag fashion so that the converted array should be in the below form:

***Input****: N = 7 , arr[] = {4, 3, 7, 8, 6, 2, 1}****Output****: arr[] = {3, 7, 4, 8, 2, 6, 1}****Explanation:****The given array is in zig-zag pattern as we can see**3 < 7 > 4 < 8 > 2 < 6 >1*



**136. 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.

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

**Output:** 1

**Sol:**

Take the 1st element as ans and then perform XOR on rest of the element with 1st element.

XOR : 1 – 1 : 0, 0 – 1 : 1, 1 – 0 : 1

**1636. Sort Array by Increasing Frequency**

Given an array of integers nums, sort the array in **increasing** order based on the frequency of the values. If multiple values have the same frequency, sort them in **decreasing** order.

Return the *sorted array*.

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

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

**Explanation:** '3' has a frequency of 1, '1' has a frequency of 2, and '2' has a frequency of 3.

**Sol:**

Take a map and store the count of each number.

Take a priority queue and put comparator into it.

**new** PriorityQueue<>((a, b) -> map.get(a) == map.get(b) ? b - a : map.get(a) - map.get(b));

if frequency is same then just sort it in descending order. Else sort it based on frequency.

Add all the key of the map to queue.

Then poll one by one and take a count of that number from map.

Add the same polled number that many times into result array.

**905. Sort Array By Parity**

Given an integer array nums, move all the even integers at the beginning of the array followed by all the odd integers.

Return ***any array****that satisfies this condition*.

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

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

**Explanation:** The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.

**Sol:**

Take 2 pointer i and j starting from 0.

Iterate the array and if arr[j] % 2 != 0 then just increment j counter. Else swap arr[i] and ar[j] and increment i++, j++

**922. Sort Array By Parity II**

Given an array of integers nums, half of the integers in nums are **odd**, and the other half are **even**.

Sort the array so that whenever nums[i] is odd, i is **odd**, and whenever nums[i] is even, i is **even**.

Return *any answer array that satisfies this condition*.

**Input:** nums = [4,2,5,7]

**Output:** [4,5,2,7]

**Explanation:** [4,7,2,5], [2,5,4,7], [2,7,4,5] would also have been accepted.

**Sol:**

Take 2 pointer i and j starting i from 0 and j from 1

Iterate the array till I < n && j < n

While(nums[i] %2 == 0)

I += 2;

While(nums[j] % 2 == 1)

J += 2;

Once control comes here it means i points to odd value index and j points to even value index

So we will swap arr[i]] and arr[j]

**1. 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].

**Sol:**

Take a map.

Iterate the array and check key = target – nums[i] is present in map

If yes then add the ans[0] = map.get(key) and ans[1] = i

Else add the key as key and index as value.