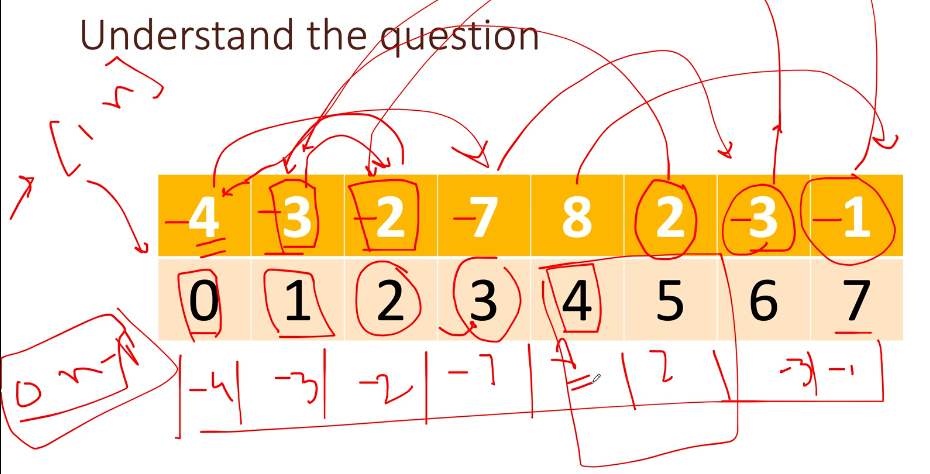
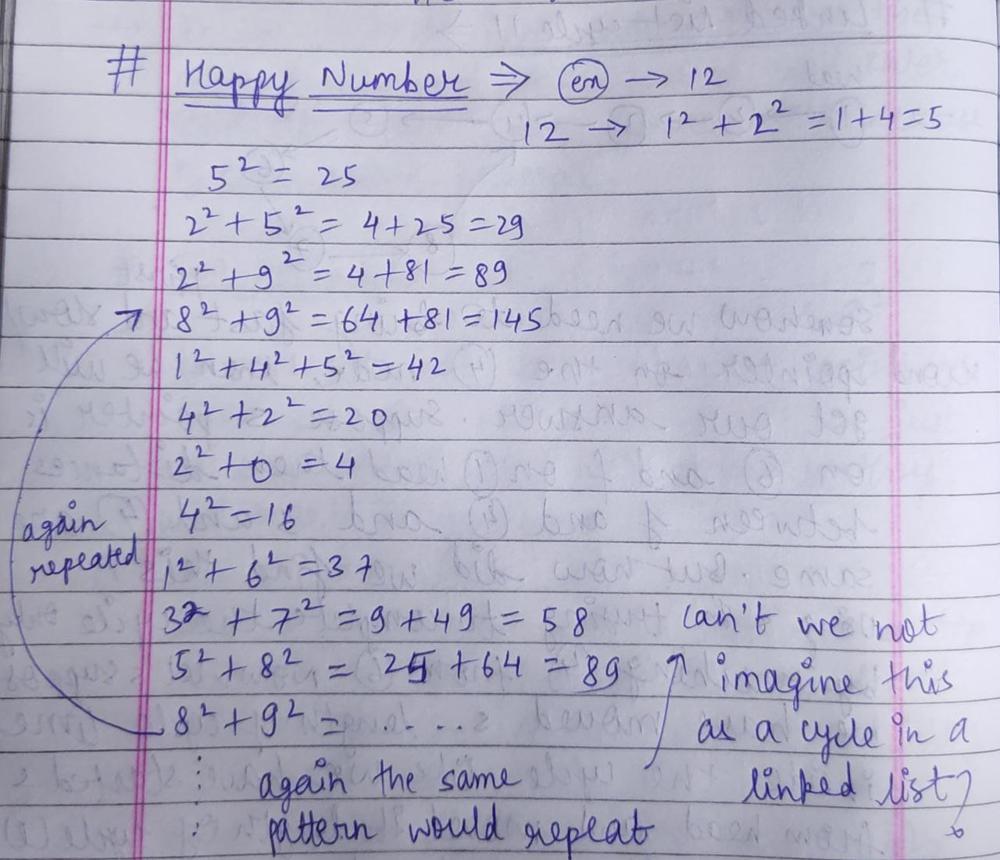
**[448. Find All Numbers Disappeared in an Array](https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/)**

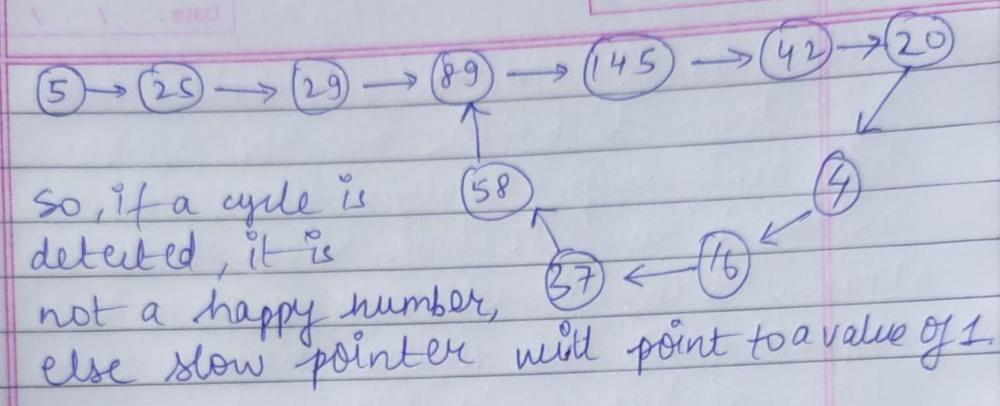
**Sol:**



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/)





[**169. Majority Element**](https://leetcode.com/problems/majority-element/)

**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.

Sol:

XOR operation we should know

1. If we XOR something with 0, it will return the same number

Example: 5 ^ 0 = 5

1. If we XOR 2 same numbers then output will be 0

Example: 5 ^ 5 = 0

1. If we XOR 2 different numbers then it will result nothing

Example: 4 ^ 5 = 4 ^ 5

First we will XOR each number present in the nums array to itself

Then we will XOR it with every number present in the range [0, n]

So in this way every number will appear twice except missing number. So number which appears twice will lead to 0 and only thing will be remaining is missing number

**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.

**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.

**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].