

Question 1

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.

Example 1:

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**.

Solution:

Java: <https://pastebin.com/ZdAM2s5J>

Python: <https://pastebin.com/HJ2sXVbV>

Javascript: <https://pastebin.com/JJr1RvBt>

TC: $O(n)$

SC: $O(n)$

Question 2

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.

Example 1:

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].

Solution:

Java: <https://pastebin.com/fSmRxP9M>

Python: <https://pastebin.com/DsTCaT1w>

Javascript: <https://pastebin.com/jEDe46tn>

TC : $O(n \log n)$
SC : $O(\log n)$

Question 3

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 **n**.

Example 1:

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**.

Solution:

Java: <https://pastebin.com/TgFWjiHF>

Python: <https://pastebin.com/zzHVTdMA>

Javascript: <https://pastebin.com/7PRKPCWU>

TC : $O(n \log n)$
SC : $O(\log n)$

Question 4

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

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Example 1:

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

Output: 3

Solution:

Java: <https://pastebin.com/mVJL4PDB>

Python: <https://pastebin.com/E3AR6D6M>

Javascript: <https://pastebin.com/uQipiegQ>

TC: $O(n \log n)$

SC : $O(\log n)$

Question 5

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.

Example 1:

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

Output: 2

Solution:

Java: <https://pastebin.com/iAeDYGKd>

Python: <https://pastebin.com/vauPf0Yb>

Javascript: <https://pastebin.com/CQ0V3Q9u>

TC : $O(n)$

SC : $O(n)$

Question 6

There are many situations where we use integer values as index in array to see presence or absence. We can use bit manipulations to optimize space in such problems.

Let us consider the below problem as an example.

Given two numbers say a and b, mark the multiples of 2 and 5 between a and b and output each of the multiples.

Note : We have to mark the multiples i.e save (key, value) pairs in memory such that each key either has a value as 1 or 0 representing a multiple of 2 or 5 or not respectively.

Examples :

Input : 2 10

Output : 2 4 5 6 8 10

Input: 60 95

Output: 60 62 64 65 66 68 70 72 74 75 76 78
80 82 84 85 86 88 90 92 94 95

Solution:

Java: <https://pastebin.com/14gqxY5W>

Python: <https://pastebin.com/jYse75qN>

Javascript: <https://pastebin.com/PxRpLKyK>

TC : $O(n)$

SC : $O(n)$

Question 7

Given an array of positive integers. We need to make the given array a 'Palindrome'. The only allowed operation is "merging" (of two adjacent elements). Merging two adjacent elements means replacing them with their sum. The task is to find the minimum number of merge operations required to make the given array a 'Palindrome'.

To make any array a palindrome, we can simply apply merge operation $n-1$ times where n is the size of the array (because a single-element array is always palindromic, similar to a single-character string). In that case, the size of array will be reduced to 1. But in this problem, we are asked to do it in the minimum number of operations.

Example :

Input : `arr[] = {15, 4, 15}`

Output : 0

Array is already a palindrome. So we do not need any merge operation.

Input : `arr[] = {1, 4, 5, 1}`

Output : 1

We can make given array palindrome with minimum one merging (merging 4 and 5 to make 9)

Input : `arr[] = {11, 14, 15, 99}`

Output : 3

We need to merge all elements to make a palindrome.

Solution:

Java: <https://pastebin.com/ZGHtqWQJ>

Python: <https://pastebin.com/b3NyqXs4>

Javascript: <https://pastebin.com/dY3EMMyV>

TC : $O(n)$

SC : $O(1)$