Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

30% completed

Search Course

Cyclic Sort (easy)
Find the Missing Number (easy)
Find all Missing Numbers (easy)
Find the Duplicate Number (easy)
Find all Duplicate Numbers (easy)

Find all Missing Numbers (easy)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
- Time complexity
- Space complexity

Problem Statement

We are given an unsorted array containing numbers taken from the range 1 to 'n'. The array can have duplicates, which means some numbers will be missing. Find all those missing numbers.

Example 1:

```
Input: [2, 3, 1, 8, 2, 3, 5, 1]
Output: 4, 6, 7
Explanation: The array should have all numbers from 1 to 8, due to duplicate
s 4, 6, and 7 are missing.
```

Example 2:

```
Input: [2, 4, 1, 2]
Output: 3
```

Example 3:

```
Input: [2, 3, 2, 1]
Output: 4
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
import java.util.*;

class AllMissingNumbers {

  public static List<Integer> findNumbers(int[] nums) {
    List<Integer> missingNumbers = new ArrayList<>();
    // TODO: Write your code here
    return missingNumbers;
  }
}
```

Test | Save | Reset

Solution

This problem follows the **Cyclic Sort** pattern and shares similarities with Find the Missing Number with one difference. In this problem, there can be many duplicates whereas in 'Find the Missing Number' there were no duplicates and the range was greater than the length of the array.

However, we will follow a similar approach though as discussed in Find the Missing Number to place the numbers on their correct indices. Once we are done with the cyclic sort we will iterate the array to find all indices that are missing the correct numbers.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
import java.util.*;

class AllMissingNumbers {

  public static List<Integer> findNumbers(int[] nums) {
    int i = 0;
    while (i < nums.length) {
      if (nums[i] != nums[nums[i] - 1])
        swap(nums, i, nums[i] - 1);
      else
        i++;
    }

    List<Integer> missingNumbers = new ArrayList<>();
    for (i = 0; i < nums.length; i++)
      if (nums[i] != i + 1)
        missingNumbers.add(i + 1);

    return missingNumbers;
  }

  private static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }

  public static void main(String[] args) {
```

Run | Save | Reset

Time complexity

The time complexity of the above algorithm is $O(n)$.

Space complexity

Ignoring the space required for the output array, the algorithm runs in constant space $O(1)$.