Solution Review: Problem Challenge 3

We'll cover the following

Find the First K Missing Positive Numbers (hard)

Given an unsorted array containing numbers and a number 'k', find the first 'k' missing positive numbers in the array.

Example 1:

```
Input: [3, -1, 4, 5, 5], k=3
Output: [1, 2, 6]
Explanation: The smallest missing positive numbers are 1, 2 and 6.
```

Example 2:

```
Input: [2, 3, 4], k=3
Output: [1, 5, 6]
Explanation: The smallest missing positive numbers are 1, 5 and 6.
```

Example 3:

```
Input: [-2, -3, 4], k=2
Output: [1, 2]
Explanation: The smallest missing positive numbers are 1 and 2.
```

Solution

This problem follows the **Cyclic Sort** pattern and shares similarities with Find the Smallest Missing Positive Number. The only difference is that, in this problem, we need to find the first 'k' missing numbers compared to only the first missing number.

We will follow a similar approach as discussed in Find the Smallest Missing Positive Number to place the numbers on their correct indices and ignore all numbers that are out of the range of the array. Once we are done with the cyclic sort we will iterate through the array to find indices that do not have the correct numbers.

If we are not able to find 'k' missing numbers from the array, we need to add additional numbers to the output array. To find these additional numbers we will use the length of the array. For example, if the length of the array is 4, the next missing numbers will be 4, 5, 6 and so on. One tricky aspect is that any of these additional numbers could be part of the array. Remember, while sorting, we ignored all numbers that are greater than or equal to the length of the array. So all indices that have the missing numbers could possibly have these additional numbers. To handle this, we must keep track of all numbers from those indices that have missing numbers. Let's understand this with an example:

```
    nums: [2, 1, 3, 6, 5], k =2
```

After the cyclic sort our array will look like:

```
    nums: [1, 2, 3, 6, 5]
```

From the sorted array we can see that the first missing number is '4' (as we have '6' on the fourth index) but to find the second missing number we need to remember that the array does contain '6'. Hence, the next missing number is '7'.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
  const n = nums.length;
  let i = 0;
  while (i < n) {
    const j = nums[i] - 1;
    if (nums[i] > 0 && nums[i] <= n && nums[i] !== nums[j]) {
      [nums[i], nums[j]] = [nums[j], nums[i]]; // swap
    } else {
      i += 1;
    }
  }
```

Explore
Tracks
My Courses
Edpresso
Refer a Friend
Create

```
  missingNumbers = [];
  const extraNumbers = new Set();
  for (i = 0; i < n; i++) {
    if (missingNumbers.length < k) {
      if (nums[i] !== i + 1) {
        missingNumbers.push(i + 1);
        extraNumbers.add(nums[i]);
      }
    }
  }

  // add the remaining missing numbers
  i = 1;
  while (missingNumbers.length < k) {
    const candidateNumber = i + n;
    // ignore if the array contains the candidate number
    if (!extraNumbers.has(candidateNumber)) {
      missingNumbers.push(candidateNumber);
```

RUN SAVE RESET

Close

Output 1.746s

```
[ 1, 2, 6 ]
[ 1, 5, 6 ]
[ 1, 2 ]
```

Time complexity

The time complexity of the above algorithm is $O(n+k)$, as the last two `for` loops will run for $O(n)$ and $O(k)$ times respectively.

Space complexity

The algorithm needs $O(k)$ space to store the `extraNumbers`.

← Back
Problem Challenge 3

Next →
Introduction

✓ Mark as Completed

Report an Issue   Ask a Question