Cyclic Sort (easy)

We'll cover the following

Problem Statement

We are given an array containing 'n' objects. Each object, when created, was assigned a unique number from 1 to 'n' based on their creation sequence. This means that the object with sequence number '3' was created just before the object with sequence number '4'.

Write a function to sort the objects in-place on their creation sequence number in $O(n)$ and without any extra space. For simplicity, let's assume we are passed an integer array containing only the sequence numbers, though each number is actually an object.

Example 1:

```
Input: [3, 1, 5, 4, 2]
Output: [1, 2, 3, 4, 5]
```

Example 2:

```
Input: [2, 6, 4, 3, 1, 5]
Output: [1, 2, 3, 4, 5, 6]
```

Example 3:

```
Input: [1, 5, 6, 4, 3, 2]
Output: [1, 2, 3, 4, 5, 6]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const cyclic_sort = function(nums) {
  // TODO: Write your code here
  return nums;
}


console.log(`${cyclic_sort([3, 1, 5, 4, 2])}`)
console.log(`${cyclic_sort([2, 6, 4, 3, 1, 5])}`)
console.log(`${cyclic_sort([1, 5, 6, 4, 3, 2])}`)

```

TEST SAVE RESET

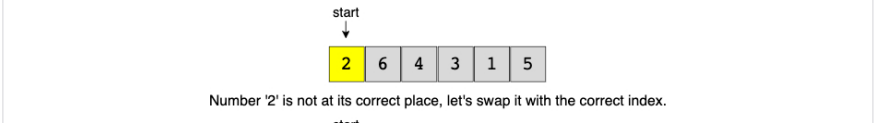
Solution

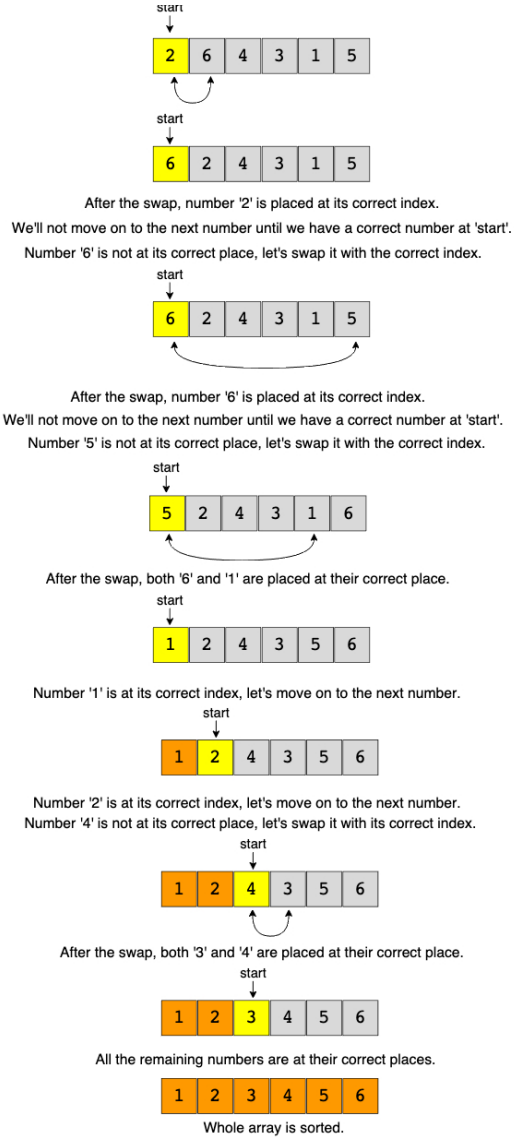
As we know, the input array contains numbers in the range of 1 to 'n'. We can use this fact to devise an efficient way to sort the numbers. Since all numbers are unique, we can try placing each number at its correct place, i.e., placing '1' at index '0', placing '2' at index '1', and so on.

To place a number (or an object in general) at its correct index, we first need to find that number. If we first find a number and then place it at its correct place, it will take us $O(N^2)$, which is not acceptable.

Instead, what if we iterate the array one number at a time, and if the current number we are iterating is not at the correct index, we swap it with the number at its correct index. This way we will go through all numbers and place them in their correct indices, hence, sorting the whole array.

Let's see this visually with the above-mentioned Example-2:





Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

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


console.log(cyclic_sort([3, 1, 5, 4, 2]));
console.log(cyclic_sort([2, 6, 4, 3, 1, 5]));
console.log(cyclic_sort([1, 5, 6, 4, 3, 2]));

```

RUN SAVE RESET

Close

Output 2.215s

```
[ 1, 2, 3, 4, 5 ]
[ 1, 2, 3, 4, 5, 6 ]
[ 1, 2, 3, 4, 5, 6 ]
```

Time complexity

The time complexity of the above algorithm is $O(n)$. Although we are not incrementing the index i when

Explore
Tracks
My Courses
Edpresso
Refer a Friend
Create

Complement of Base 10 Number (medium)
Problem Challenge 1
Solution Review: Problem Challenge 1

Pattern: Top 'K' Elements

Introduction
Top 'K' Numbers (easy)
Kth Smallest Number (easy)
'K' Closest Points to the Origin (easy)
Connect Ropes (easy)
Top 'K' Frequent Numbers (medium)
Frequency Sort (medium)
Kth Largest Number in a Stream (medium)
'K' Closest Numbers (medium)
Maximum Distinct Elements (medium)
Sum of Elements (medium)
Rearrange String (hard)
Problem Challenge 1
Solution Review: Problem

The time complexity of the above algorithm is $O(n)$. Although we are not incrementing the index `i` when swapping the numbers, this will result in more than 'n' iterations of the loop, but in the worst-case scenario, the `while` loop will swap a total of 'n-1' numbers and once a number is at its correct index, we will move on to the next number by incrementing `i`. So overall, our algorithm will take $O(n) + O(n-1)$ which is asymptotically equivalent to $O(n)$.

Space complexity

The algorithm runs in constant space $O(1)$.

MARK AS COMPLETED

← Back
Introduction

Next →
Find the Missing Number (easy)

Report an Issue