

Grokking the Coding Interview: Patterns for Coding Questions

26% completed

- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

Cyclic Sort (easy)

We'll cover the following

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

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++

```
1 const cyclic_sort = function(nums) {
2   // TODO: Write your code here
3   return nums;
4 }
5
6
7 console.log(`${cyclic_sort([3, 1, 5, 4, 2])}`)
8 console.log(`${cyclic_sort([2, 6, 4, 3, 1, 5])}`)
9 console.log(`${cyclic_sort([1, 5, 6, 4, 3, 2])}`)
10
```

TEST

SAVE

RESET

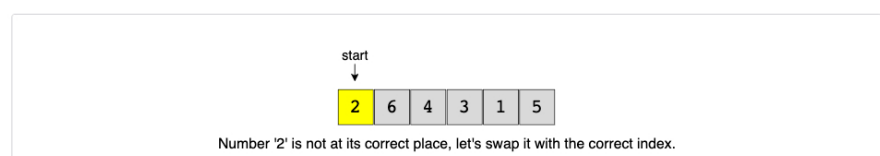
Solution

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:



- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

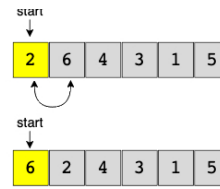
- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

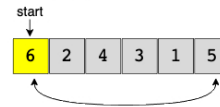
- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Bitwise XOR

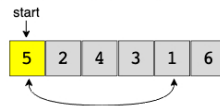
- Introduction
- Single Number (easy)
- Two Single Numbers (medium)



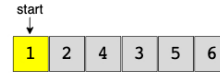
After the swap, number '2' is placed at its correct index.
We'll not move on to the next number until we have a correct number at 'start'.
Number '6' is not at its correct place, let's swap it with the correct index.



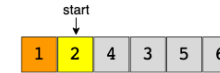
After the swap, number '6' is placed at its correct index.
We'll not move on to the next number until we have a correct number at 'start'.
Number '5' is not at its correct place, let's swap it with the correct index.



After the swap, both '6' and '1' are placed at their correct place.



Number '1' is at its correct index, let's move on to the next number.



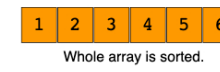
Number '2' is at its correct index, let's move on to the next number.
Number '4' is not at its correct place, let's swap it with its correct index.



After the swap, both '3' and '4' are placed at their correct place.



All the remaining numbers are at their correct places.



Whole array is sorted.

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```

1 function cyclic_sort(nums) {
2   let i = 0;
3   while (i < nums.length) {
4     const j = nums[i] - 1;
5     if (nums[i] !== nums[j]) {
6       [nums[i], nums[j]] = [nums[j], nums[i]]; // swap
7     } else {
8       i += 1;
9     }
10  }
11  return nums;
12 }
13
14
15 console.log(cyclic_sort([3, 1, 5, 4, 2]));
16 console.log(cyclic_sort([2, 6, 4, 3, 1, 5]));
17 console.log(cyclic_sort([1, 5, 6, 4, 3, 2]));
18

```

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 $O(n^2)$. 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 $O(1)$.

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. [See how](#)

← Back

Introduction

Find the Missing Number (easy)

MARK AS COMPLETED

Next →

Report an Issue