

## Grokking the Coding Interview: Patterns for Coding Questions

48% completed



### 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

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)

## Find the Duplicate Number (easy)

### We'll cover the following

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

### Problem Statement

We are given an unsorted array containing 'n+1' numbers taken from the range 1 to 'n'. The array has only one duplicate but it can be repeated multiple times. **Find that duplicate number without using any extra space.** You are, however, allowed to modify the input array.

#### Example 1:

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

#### Example 2:

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

#### Example 3:

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

### Try it yourself

Try solving this question here:

Java

Python3

**JS**

C++

```
1 const find_duplicate = function(nums) {
2   // TODO: Write your code here
3   return -1;
4 };
5
```

TEST

SAVE

RESET

### Solution

This problem follows the **Cyclic Sort** pattern and shares similarities with [Find the Missing Number](#). Following a similar approach, we will try to place each number on its correct index. Since there is only one duplicate, if while swapping the number with its index both the numbers being swapped are same, we have found our duplicate!

### Code

Here is what our algorithm will look like:

Java

Python3

C++

**JS**

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

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

## Pattern: Top 'K' Elements

RUN

SAVE

RESET

Close

Output2.140s

4  
3  
4

Time complexity #

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

Space complexity #

The algorithm runs in constant space  $O(1)$  but modifies the input array.

## Similar Problems #

**Problem 1:** Can we solve the above problem in  $O(1)$  space and without modifying the input array?

**Solution:** While doing the cyclic sort, we realized that the array will have a cycle due to the duplicate number and that the start of the cycle will always point to the duplicate number. This means that we can use the fast & the slow pointer method to find the duplicate number or the start of the cycle similar to [Start of LinkedList Cycle](#).

Java

Python3

C++

JS

```
1 function find_duplicate(arr) {
2   let slow = arr[0];
3   fast = arr[arr[0]];
4   while (slow !== fast) {
5     slow = arr[slow];
6     fast = arr[arr[fast]];
7   }
8   // find cycle length
9   let current = arr[slow];
10  let cycleLength = 1;
11  while (current !== arr[slow]) {
12    current = arr[current];
13    cycleLength += 1;
14  }
15
16  return find_start(arr, cycleLength);
17 }
18
19 function find_start(arr, cycleLength) {
20   let pointer1 = arr[0];
21   let pointer2 = arr[0];
22   // move pointer2 ahead 'cycleLength' steps
23   while (cycleLength > 0) {
24     pointer2 = arr[pointer2];
25     cycleLength -= 1;
26   }
27   // increment both pointers until they meet at the start of the cycle
28   while (pointer1 !== pointer2) {
```

RUN

SAVE

RESET

Close

Output1.661s

4  
3  
4

The time complexity of the above algorithm is  $O(n)$  and the space complexity is  $O(1)$ .

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you instead of you applying to them. [See how](#)

← Back

Find all Missing Numbers (easy)

Find all Duplicate Numbers (easy)

Next →

Mark as Completed

Report an Issue

Ask a Question

