educative

Grokking the Coding Interview: Patterns for Coding Questions

11% completed

Search Course

Pair with Target Sum (easy)
Remove Duplicates (easy)
Squaring a Sorted Array (easy)
Triplet Sum to Zero (medium)
Triplet Sum Close to Target (medium)
Triplets with Smaller Sum (medium)
Subarrays with Product Less than a Target (medium)
Dutch National Flag Problem (medium)
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: Fast & Slow pointers

Introduction
LinkedList Cycle (easy)
Start of LinkedList Cycle (medium)
Happy Number (medium)
Middle of the LinkedList (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: Merge Intervals

Introduction
Merge Intervals (medium)
Insert Interval (medium)
Intervals Intersection (medium)
Conflicting Appointments (medium)
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: Cyclic Sort

Introduction
Cyclic Sort (easy)
Find the Missing Number (easy)

Dutch National Flag Problem (medium)

We'll cover the following

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

Problem Statement

Given an array containing 0s, 1s and 2s, sort the array in-place. You should treat numbers of the array as objects, hence, we can't count 0s, 1s, and 2s to recreate the array.

The flag of the Netherlands consists of three colors: red, white and blue; and since our input array also consists of three different numbers that is why it is called Dutch National Flag problem.

Example 1:

```
Input: [1, 0, 2, 1, 0]
Output: [0 0 1 1 2]
```

Example 2:

```
Input: [2, 2, 0, 1, 2, 0]
Output: [0 0 1 2 2 2 ]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const dutch_flag_sort = function(arr) {
  // TODO: Write your code here
};

```

TEST SAVE RESET

Solution

The brute force solution will be to use an in-place sorting algorithm like Heapsort which will take $O(N * logN)$. Can we do better than this? Is it possible to sort the array in one iteration?

We can use a **Two Pointers** approach while iterating through the array. Let's say the two pointers are called `low` and `high` which are pointing to the first and the last element of the array respectively. So while iterating, we will move all 0s before `low` and all 2s after `high` so that in the end, all 1s will be between `low` and `high`.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
  // all elements from >= low < i are 1
  let low = 0,
    high = arr.length - 1,
    i = 0;
  while (i <= high) {
    if (arr[i] === 0) {
      [arr[i], arr[low]] = [arr[low], arr[i]]; // swap
      // increment 'i' and 'low'
      i += 1;
      low += 1;
    } else if (arr[i] === 1) {
      i += 1;
    } else { // the case for arr[i] === 2
      [arr[i], arr[high]] = [arr[high], arr[i]]; // swap
      // decrement 'high' only, after the swap the number at index 'i' could be 0, 1, or 2
      high -= 1;
    }
  }
}


let arr = [1, 0, 2, 1, 0];
dutch_flag_sort(arr);
console.log(arr);

arr = [2, 2, 0, 1, 2, 0];
```

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

```
dutch_flag_sort(arr);
console.log(arr);
```

Output 2.559s

```
[ 0, 0, 1, 1, 2 ]
[ 0, 0, 1, 2, 2, 2 ]
```

Time complexity

The time complexity of the above algorithm will be $O(N)$ as we are iterating the input array only once.

Space complexity

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

MARK AS COMPLETED
Back
Subarrays with Product Less than a Ta...
Next
Problem Challenge 1
Report an Issue