

Grokking the Coding Interview: Patterns for Coding Questions

8% completed

- Introduction
- 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)
- 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

Remove Duplicates (easy)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
 - Code
 - Time Complexity
 - Space Complexity
- Similar Questions

Problem Statement

Given an array of sorted numbers, **remove all duplicates** from it. You should **not use any extra space**; after removing the duplicates in-place return the new length of the array.

Example 1:

```
Input: [2, 3, 3, 3, 6, 9, 9]
Output: 4
Explanation: The first four elements after removing the duplicates will be [2, 3, 6, 9].
```

Example 2:

```
Input: [2, 2, 2, 11]
Output: 2
Explanation: The first two elements after removing the duplicates will be [2, 11].
```

Try it yourself

Try solving this question here:

Java Python3 JS C++

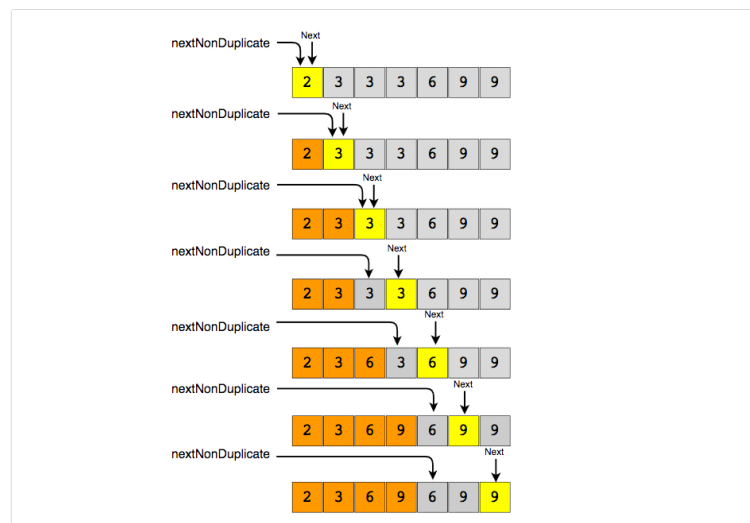
```
1 const remove_duplicates = function(arr) {
2   // TODO: Write your code here
3   return -1;
4 };
5
```

TEST SAVE RESET

Solution

In this problem, we need to remove the duplicates in-place such that the resultant length of the array remains sorted. As the input array is sorted, therefore, one way to do this is to shift the elements left whenever we encounter duplicates. In other words, we will keep one pointer for iterating the array and one pointer for placing the next non-duplicate number. So our algorithm will be to iterate the array and whenever we see a non-duplicate number we move it next to the last non-duplicate number we've seen.

Here is the visual representation of this algorithm for Example-1:



Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1 function remove_duplicates(arr) {
2   // index of the next non-duplicate element
3   let nextNonDuplicate = 1;
4
5   let i = 1;
6   while (i < arr.length) {
```

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)

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

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

```
7   if (arr[nextNonDuplicate - 1] !== arr[i]) {
8       arr[nextNonDuplicate] = arr[i];
9       nextNonDuplicate += 1;
10  }
11  i += 1;
12  }
13
14  return nextNonDuplicate;
15 }
16
17 console.log(remove_duplicates([2, 3, 3, 3, 6, 9, 9]));
18 console.log(remove_duplicates([2, 2, 2, 11]));
```

RUN

SAVE

RESET

Time Complexity 🕒

The time complexity of the above algorithm will be $O(N)$, where 'N' is the total number of elements in the given array.

Space Complexity 📍

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

Similar Questions

Problem 1: Given an unsorted array of numbers and a target 'key', remove all instances of 'key' in-place and return the new length of the array.

Example 1:

```
Input: [3, 2, 3, 6, 3, 10, 9, 3], Key=3
Output: 4
Explanation: The first four elements after removing every 'Key' will be [2, 6, 10, 9].
```

Example 2:

```
Input: [2, 11, 2, 2, 1], Key=2
Output: 2
Explanation: The first two elements after removing every 'Key' will be [11, 1].
```

Solution: This problem is quite similar to our parent problem. We can follow a two-pointer approach and shift numbers left upon encountering the 'key'. Here is what the code will look like:

Java Python3 C++ JS

```
1 function remove_element(arr, key) {
2   let nextElement = 0; // index of the next element which is not 'key'
3   for (i = 0; i < arr.length; i++) {
4     if (arr[i] !== key) {
5       arr[nextElement] = arr[i];
6       nextElement += 1;
7     }
8   }
9   return nextElement;
10 }
11
12 console.log('Array new length: ${remove_element([3, 2, 3, 6, 3, 10, 9, 3])}');
13 console.log('Array new length: ${remove_element([2, 11, 2, 2, 1, 2])}');
```

RUN

SAVE

RESET

Output

3.420s

Close

Array new length: 4
Array new length: 2

Time and Space Complexity: The time complexity of the above algorithm will be $O(N)$, where 'N' is the total number of elements in the given array.

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

← Back

Pair with Target Sum (easy)

MARK AS COMPLETED

Next →

Squaring a Sorted Array (easy)

Report an Issue

Ask a Question