

Grokking the Coding Interview: Patterns for Coding Questions

9% 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

Triplet Sum to Zero (medium)

We'll cover the following

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

Problem Statement

Given an array of unsorted numbers, find all **unique triplets** in it that add up to zero.

Example 1:

```
Input: [-3, 0, 1, 2, -1, 1, -2]
Output: [-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]
Explanation: There are four unique triplets whose sum is equal to zero.
```

Example 2:

```
Input: [-5, 2, -1, -2, 3]
Output: [-5, 2, 3], [-2, -1, 3]
Explanation: There are two unique triplets whose sum is equal to zero.
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 const search_triplets = function(arr) {
2   triplets = [];
3   // TODO: Write your code here
4   return triplets;
5 };
6
```

TESTSAVERESET↺

Solution

This problem follows the **Two Pointers** pattern and shares similarities with [Pair with Target Sum](#). A couple of differences are that the input array is not sorted and instead of a pair we need to find triplets with a target sum of zero.

To follow a similar approach, first, we will sort the array and then iterate through it taking one number at a time. Let's say during our iteration we are at number 'X', so we need to find 'Y' and 'Z' such that $X + Y + Z = 0$. At this stage, our problem translates into finding a pair whose sum is equal to $-X$ (as from the above equation $Y + Z = -X$).

Another difference from [Pair with Target Sum](#) is that we need to find all the unique triplets. To handle this, we have to skip any duplicate number. Since we will be sorting the array, so all the duplicate numbers will be next to each other and are easier to skip.

Code

Here is what our algorithm will look like:

JavaPython3C++JS

```
12 }
13
14
15 function search_pair(arr, target_sum, left, triplets) {
16   let right = arr.length - 1;
17   while (left < right) {
18     const current_sum = arr[left] + arr[right];
19     if (current_sum === target_sum) { // found the triplet
20       triplets.push([-target_sum, arr[left], arr[right]]);
21       left += 1;
22       right -= 1;
23       while (left < right && arr[left] === arr[left - 1]) {
24         left += 1; // skip same element to avoid duplicate triplets
25       }
26       while (left < right && arr[right] === arr[right + 1]) {
27         right -= 1; // skip same element to avoid duplicate triplets
28       }
29     } else if (target_sum > current_sum) {
30       left += 1; // we need a pair with a bigger sum
31     } else {
32       right -= 1; // we need a pair with a smaller sum
33     }
34   }
35 }
36
37
38 console.log(search_triplets([-3, 0, 1, 2, -1, 1, -2]));
39 console.log(search_triplets([-5, 2, -1, -2, 3]));
```

RUNSAVERESET↺

Output3.019s

Close

Tracks

My Courses

Edpresso

Refer a Friend

Create

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)

[[-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]]
[[-5, 2, 3], [-2, -1, 3]]

Time complexity

Sorting the array will take $O(N * \log N)$. The `searchPair()` function will take $O(N)$. As we are calling `searchPair()` for every number in the input array, this means that overall `searchTriplets()` will take $O(N * \log N + N^2)$, which is asymptotically equivalent to $O(N^2)$.

Space complexity

Ignoring the space required for the output array, the space complexity of the above algorithm will be $O(N)$ which is required for sorting.

← Back

Next →

Squaring a Sorted Array (easy)

Triplet Sum Close to Target (medium)

Report an Issue

Ask a Question