Grokking the Coding Interview: Patterns for Coding Questions

10% completed

Search Course

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

Subarrays with Product Less than a Target (medium)

We'll cover the following

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

Problem Statement

Given an array with positive numbers and a target number, find all of its contiguous subarrays whose **product is less than the target number**.

Example 1:

```
Input: [2, 5, 3, 10], target=30
Output: [2], [5], [2, 5], [3], [5, 3], [10]
Explanation: There are six contiguous subarrays whose product is less than the target.
```

Example 2:

```
Input: [8, 2, 6, 5], target=50
Output: [8], [2], [8, 2], [6], [2, 6], [5], [6, 5]
Explanation: There are seven contiguous subarrays whose product is less than the target.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const find_subarrays = function(arr, target) {
  result = [];
  // TODO: Write your code here
  return result;
};

```

TEST SAVE RESET

Solution

This problem follows the **Sliding Window** and the **Two Pointers** pattern and shares similarities with Triplets with Smaller Sum with two differences:

1. In this problem, the input array is not sorted.
2. Instead of finding triplets with sum less than a target, we need to find all subarrays having a product less than the target.

The implementation will be quite similar to Triplets with Smaller Sum.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com


function find_subarrays(arr, target) {
  let result = [],
    product = 1,
    left = 0;
  for (right = 0; right < arr.length; right++) {
    product *= arr[right];
    while ((product >= target && left < arr.length)) {
      product /= arr[left];
      left += 1;
    }
    // since the product of all numbers from left to right is less than the target therefore,
    // all subarrays from left to right will have a product less than the target too; to avoid
    // duplicates, we will start with a subarray containing only arr[right] and then extend it
    const tempList = new Deque();
    for (let i = right; i > left - 1; i--) {
      tempList.unshift(arr[i]);
      result.push(tempList.toArray());
    }
  }
  return result;
}


console.log(find_subarrays([2, 5, 3, 10], 30));
console.log(find_subarrays([8, 2, 6, 5], 50));
```

RUN SAVE RESET

Close

Output 3.818s

```
[[2], [5], [2, 5], [3], [5, 3], [10]]
[[8], [2], [8, 2], [6], [2, 6], [5], [6, 5]]
```

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

Time complexity

The main `for-loop` managing the sliding window takes $O(N)$ but creating subarrays can take up to $O(N^2)$ in the worst case. Therefore overall, our algorithm will take $O(N^3)$.

Space complexity

Ignoring the space required for the output list, the algorithm runs in $O(N)$ space which is used for the temp list.

Can you try estimating how much space will be required for the output list?

Hide Hint

The worst case will happen when every subarray has a product less than the target!

So the question will be, how many contiguous subarray an array can have?

It is definately not all Permutations of the given array, is it all Combinations of the given array?

It is not all the Combinations of all elements of the array!

For an array with distinct elements, finding all of its contiguous subarrays is like finding the number of ways to choose two indices i and j in the array such that `i <= j`.

If there are a total of `n` elements in the array, here is how we can count all the contiguous subarrays:

- When `i = 0`, `j` can have any value from '0' to 'n-1', giving a total of 'n' choices.
- When `i = 1`, `j` can have any value from '1' to 'n-1', giving a total of 'n-1' choices.
- Similarly, when `i = 2`, `j` can have 'n-2' choices.
  ...
  ...
- When `i = n-1`, `j` can only have '1' choice.

Let's combine all the choices:

```
n + (n-1) + (n-2) + ... 3 + 2 + 1
```

Which gives us a total of: $n * (n + 1)/2$

So, at the most, we need a space of $O(n^2)$ for all the output lists.

MARK AS COMPLETED
← Back
Triplets with Smaller Sum (medium)
Next →
Dutch National Flag Problem (medium)
Report an Issue
Ask a Question