

Grokking the Coding Interview: Patterns for Coding Questions

1% completed

Introduction

- Who should take this course?
- Course Overview

Pattern: Sliding Window

- Introduction
- Maximum Sum Subarray of Size K (easy)
- Smallest Subarray with a given sum (easy)**
- Longest Substring with K Distinct Characters (medium)
- Fruits into Baskets (medium)
- No-repeat Substring (hard)
- Longest Substring with Same Letters after Replacement (hard)
- Longest Subarray with Ones after Replacement (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
- Problem Challenge 4
- Solution Review: Problem Challenge 4

Pattern: Two Pointers

- 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)

Smallest Subarray with a given sum (easy)

We'll cover the following

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

Problem Statement

Given an array of positive numbers and a positive number 'S', find the length of the **smallest contiguous subarray whose sum is greater than or equal to 'S'**. Return 0, if no such subarray exists.

Example 1:

```
Input: [2, 1, 5, 2, 3, 2], S=7
Output: 2
Explanation: The smallest subarray with a sum great than or equal to '7' is [5, 2].
```

Example 2:

```
Input: [2, 1, 5, 2, 8], S=7
Output: 1
Explanation: The smallest subarray with a sum greater than or equal to '7' is [8].
```

Example 3:

```
Input: [3, 4, 1, 1, 6], S=8
Output: 3
Explanation: Smallest subarrays with a sum greater than or equal to '8' are [3, 4, 1] or [1, 1, 6].
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 const smallest_subarray_with_given_sum = function(s, arr) {
2   // TODO: Write your code here
3   return -1;
4 };
5
6
```

TESTSAVERESET↺

Solution

This problem follows the **Sliding Window** pattern and we can use a similar strategy as discussed in [Maximum Sum Subarray of Size K](#). There is one difference though: in this problem, the size of the sliding window is not fixed. Here is how we will solve this problem:

- First, we will add-up elements from the beginning of the array until their sum becomes greater than or equal to 'S'.
- These elements will constitute our sliding window. We are asked to find the smallest such window having a sum greater than or equal to 'S'. We will remember the length of this window as the smallest window so far.
- After this, we will keep adding one element in the sliding window (i.e. slide the window ahead), in a stepwise fashion.
- In each step, we will also try to shrink the window from the beginning. We will shrink the window until the window's sum is smaller than 'S' again. This is needed as we intend to find the smallest window. This shrinking will also happen in multiple steps; in each step we will do two things:
 - Check if the current window length is the smallest so far, and if so, remember its length.
 - Subtract the first element of the window from the running sum to shrink the sliding window.

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



- 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

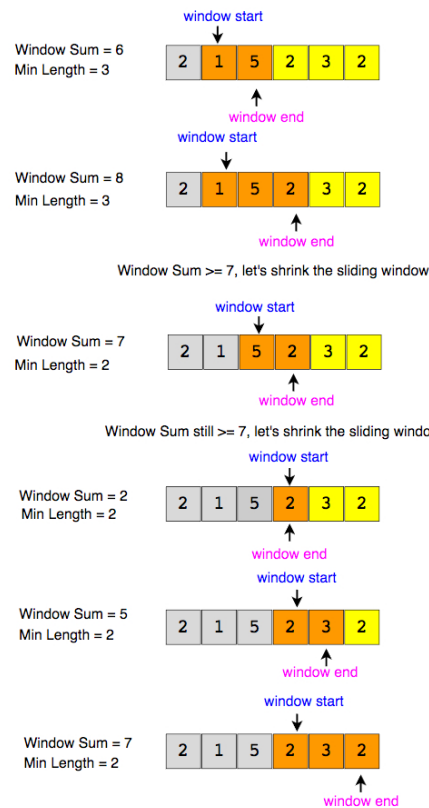
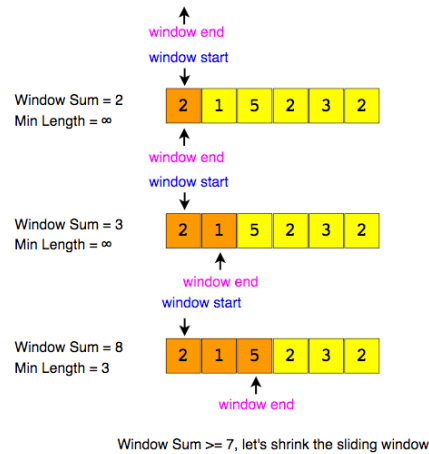
- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem 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)



Code

Here is what our algorithm will look:

```

1 function smallest_subarray_with_given_sum(s, arr) {
2   let windowSum = 0,
3   minLength = Infinity,
4   windowStart = 0;
5
6   for (windowEnd = 0; windowEnd < arr.length; windowEnd++) {
7     windowSum += arr[windowEnd]; // add the next element
8     // shrink the window as small as possible until the 'window_sum' is smaller than 's'
9     while (windowSum >= s) {
10      minLength = Math.min(minLength, windowEnd - windowStart + 1);
11      windowSum -= arr[windowStart];
12      windowStart += 1;
13    }
14  }
15
16  if (minLength === Infinity) {
17    return 0;
18  }
19  return minLength;
20 }
21
22 console.log('Smallest subarray length: ', smallest_subarray_with_given_sum(7, [2, 1, 5, 2, 3, 2]));
23 console.log('Smallest subarray length: ', smallest_subarray_with_given_sum(7, [2, 1, 5, 2, 8]));
24 console.log('Smallest subarray length: ', smallest_subarray_with_given_sum(8, [3, 4, 1, 1, 6]));
25
26

```

Edpresso

Refer a Friend

Create

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 time complexity of the above algorithm will be $O(N)$. The outer **for** loop runs for all elements and the inner **while** loop processes each element only once, therefore the time complexity of the algorithm will be $O(N + N)$ which is asymptotically equivalent to $O(N)$.

Space Complexity

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

← Back

Maximum Sum Subarray of Size K (ea...

✓ MARK AS COMPLETED

Next →

Longest Substring with K Distinct Cha...

Report an Issue