

## Grokking the Coding Interview: Patterns for Coding Questions

2% 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

### Pattern: Fast & Slow pointers

### Pattern: Merge Intervals

### Pattern: Cyclic Sort

### Pattern: In-place Reversal of a LinkedList

### Pattern: Tree Breadth First Search

### Pattern: Tree Depth First Search

### Pattern: Two Heaps

### Pattern: Subsets

### Pattern: Modified

## Fruits into Baskets (medium)

### We'll cover the following

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

### Problem Statement

Given an array of characters where each character represents a fruit tree, you are given **two baskets** and your goal is to put **maximum number of fruits in each basket**. The only restriction is that **each basket can have only one type of fruit**.

You can start with any tree, but once you have started you can't skip a tree. You will pick one fruit from each tree until you cannot, i.e., you will stop when you have to pick from a third fruit type.

Write a function to return the maximum number of fruits in both the baskets.

#### Example 1:

```
Input: Fruit=['A', 'B', 'C', 'A', 'C']
Output: 3
Explanation: We can put 2 'C' in one basket and one 'A' in the other from the subarray ['C', 'A', 'C']
```

#### Example 2:

```
Input: Fruit=['A', 'B', 'C', 'B', 'B', 'C']
Output: 5
Explanation: We can put 3 'B' in one basket and two 'C' in the other basket. This can be done if we start with the second letter: ['B', 'C', 'B', 'B', 'C']
```

### Try it yourself

Try solving this question here:

JavaPython3JS C++

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

TESTSAVERESET↺

### Solution

This problem follows the **Sliding Window** pattern and is quite similar to [Longest Substring with K Distinct Characters](#). In this problem, we need to find the length of the longest subarray with no more than two distinct characters (or fruit types!). This transforms the current problem into **Longest Substring with K Distinct Characters** where K=2.

### Code

Here is what our algorithm will look like, only the highlighted lines are different from [Longest Substring with K Distinct Characters](#):

JavaPython3C++JS

```
1 function fruits_into_baskets(fruits) {
2   let windowStart = 0,
3     maxLength = 0,
4     fruitFrequency = {};
5
6   // try to extend the range [windowStart, windowEnd]
7   for (let windowEnd = 0; windowEnd < fruits.length; windowEnd++) {
8     const rightFruit = fruits[windowEnd];
9     if (!(rightFruit in fruitFrequency)) {
10      fruitFrequency[rightFruit] = 0;
11    }
12    fruitFrequency[rightFruit] += 1;
13
14    // shrink the sliding window, until we are left with '2' fruits in the fruit frequency dictionary
15    while (Object.keys(fruitFrequency).length > 2) {
16      const leftFruit = fruits[windowStart];
17      fruitFrequency[leftFruit] -= 1;
18      if (fruitFrequency[leftFruit] === 0) {
19        delete fruitFrequency[leftFruit];
20      }
21      windowStart++;
22    }
23    maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
24  }
25  return maxLength;
26}
```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

Binary Search

Pattern: Bitwise XOR

Pattern: Top 'K' Elements

Pattern: K-way merge

Pattern : 0/1 Knapsack (Dynamic Programming)

Pattern: Topological Sort (Graph)

Miscellaneous

Conclusions

Where to Go from Here

Mark Course as Completed

```
17     fruitFrequency[leftFruit] -= 1;
18     if (fruitFrequency[leftFruit] == 0) {
19         delete fruitFrequency[leftFruit];
20     }
21     windowStart += 1; // shrink the window
22 }
23 maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
24 }
25
26 return maxLength;
27 }
28
```

RUN

SAVE

RESET

**Time Complexity** #

The time complexity of the above algorithm will be  $O(N)$  where 'N' is the number of characters in the input array. The outer **for** loop runs for all characters and the inner **while** loop processes each character 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  $O(1)$  as there can be a maximum of three types of fruits stored in the frequency map.

**Similar Problems** #

**Problem 1: Longest Substring with at most 2 distinct characters**

Given a string, find the length of the longest substring in it with at most two distinct characters.

**Solution:** This problem is exactly similar to our parent problem.

← Back

Longest Substring with K Distinct Cha...

✓ MARK AS COMPLETED

Next →

No-repeat Substring (hard)

Report an Issue