# Grokking the Coding Interview: Patterns for **Coding Questions**



### Pattern: Fast & Slow pointers



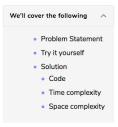
# Pattern: Merge Intervals



# Pattern: Cyclic Sort



# Subarrays with Product Less than a Target (medium)



#### 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:



## Solution

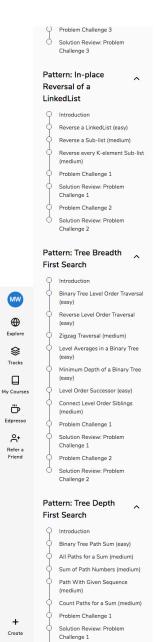
 $This \ problem \ follows \ the \ \textbf{Sliding Window} \ and \ the \ \textbf{Two Pointers} \ pattern \ and \ shares \ similarities \ with \ \textbf{Triplets}$ with Smaller Sum with two differences:

- 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

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

Here is what our algorithm will look like:

```
function find_subarrays(arr, target) {
          let result = [],
  product = 1,
  left = 0;
           for (right = 0; right < arr.length; right++) {</pre>
            product *= arr[right];
while (|product >= target && left < arr.length)) {
    product /= arr[left];
    left += 1;</pre>
             const tempList = new Deque();
for (let i = right; i > left - 1; i--) {
  tempList.unshift(arr[i]);
                 result.push(tempList.toArray());
      console.log(find_subarrays([2, 5, 3, 10], 30));
console.log(find_subarrays([8, 2, 6, 5], 50));
                                                                                                                                                RESET []
                                                                                                                                                       Close
                                                                                                                                                      3.818s
 [[2],[5],[2,5],[3],[5,3],[10]]
[[8],[2],[8,2],[6],[2,6],[5],[6,5]]
```



#### 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 lief

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  $\frac{1}{n}$  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.

