

## Grokking the Coding Interview: Patterns for Coding Questions

8% 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

## Squaring a Sorted Array (easy)

We'll cover the following

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

### Problem Statement

Given a sorted array, create a new array containing **squares of all the number of the input array** in the sorted order.

**Example 1:**

Input: [-2, -1, 0, 2, 3]  
Output: [0, 1, 4, 4, 9]

**Example 2:**

Input: [-3, -1, 0, 1, 2]  
Output: [0 1 1 4 9]

### Try it yourself

Try solving this question here:

Java Python3 JS C++

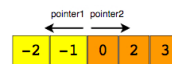
```
1 const make_squares = function(arr) {  
2   squares = []  
3   // TODO: Write your code here  
4   return squares;  
5 };  
6
```

TEST SAVE RESET

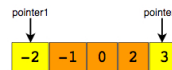
### Solution

This is a straightforward question. The only trick is that we can have negative numbers in the input array, which will make it a bit difficult to generate the output array with squares in sorted order.

An easier approach could be to first find the index of the first non-negative number in the array. After that, we can use **Two Pointers** to iterate the array. One pointer will move forward to iterate the non-negative numbers and the other pointer will move backward to iterate the negative numbers. At any step, whichever number gives us a bigger square will be added to the output array. For the above-mentioned Example-1, we will do something like this:



Since the numbers at both the ends can give us the largest square, an alternate approach could be to use two pointers starting at both the ends of the input array. At any step, whichever pointer gives us the bigger square we add it to the result array and move to the next/previous number according to the pointer. For the above-mentioned Example-1, we will do something like this:



### Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1 function make_squares(arr) {  
2   const n = arr.length;  
3   squares = Array(n).fill(0);  
4   let highestSquareIdx = n - 1;  
5   let left = 0;  
6   let right = n - 1;  
7   while (left <= right) {  
8     let leftSquare = arr[left] * arr[left];  
9     let rightSquare = arr[right] * arr[right];  
10    if (leftSquare > rightSquare) {  
11      squares[highestSquareIdx] = leftSquare;  
12      left++;  
13    } else {  
14      squares[highestSquareIdx] = rightSquare;  
15      right--;  
16    }  
17    highestSquareIdx--;  
18  }  
19 }
```

MW

Explore

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)

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

19

20

21

22

23

24

25

return squares;

}

console.log('Squares: \${make\_squares([-2, -1, 0, 2, 3])}');

console.log('Squares: \${make\_squares([-3, -1, 0, 1, 2])}');

RUN

SAVE

RESET

Close

3.077s

Output

Squares: 0,1,4,4,9

Squares: 0,1,1,4,9

Time complexity

The time complexity of the above algorithm will be  $O(N)$  as we are iterating the input array only once.

Space complexity

The space complexity of the above algorithm will also be  $O(N)$ ; this space will be used for the output array.

Back

Remove Duplicates (easy)

MARK AS COMPLETED

Next

Triplet Sum to Zero (medium)

Report an Issue

Ask a Question