

# 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 numbers of the input array** in the sorted order.

### Example 1:

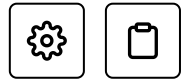
Input: [-2, -1, 0, 2, 3]  
Output: [0, 1, 4, 4, 9]

### 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    let [left,right,leftSquare,rightSquare] = [0, arr.length-1, 0, 0]
5    while (left<=right){
6      leftSquare = arr[left]*arr[left];
7      rightSquare = arr[right]*arr[right];
8      if (leftSquare<rightSquare){
9        squares.unshift(rightSquare);
10       right--;
11     } else{
12       squares.unshift(leftSquare);
13       left++;
14     }
15   }
16   return squares;
17 };
18
```



Show Results

Show Console

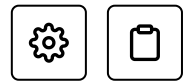


2.82s

2 of 2 Tests Passed

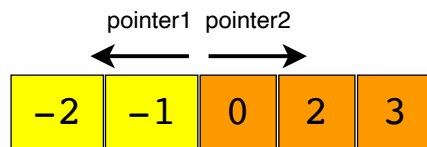
Result	Input	Expected Output	Actual Output	Reason
✓	make_squares([-2,-1,0,2,3])	0,1,4,4,9	0,1,4,4,9	Succeeded
✓	make_squares([-3,-1,0,1,2])	0,1,1,4,9	0,1,1,4,9	Succeeded

## Solution #

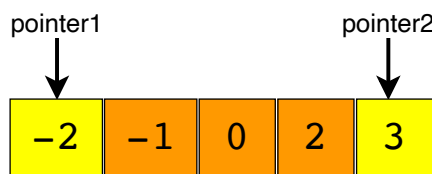


This is a straightforward question. The only trick is that we can have negative numbers in the input array, which will make it a bit difficult to generate the output array with squares in sorted order.

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:

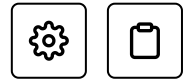


Since the numbers at both ends can give us the largest square, an alternate approach could be to use two pointers starting at both ends of the input array. At any step, whichever pointer gives us the bigger square, we add it to the result array and move to the next/previous number according to the pointer. 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
<pre>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       right = n - 1; 7   while (left &lt;= right) { 8     let leftSquare = arr[left] * arr[left], 9         rightSquare = arr[right] * arr[right]; 10    if (leftSquare &gt; rightSquare) { 11      squares[highestSquareIdx] = leftSquare; 12      left += 1; 13    } else { 14      squares[highestSquareIdx] = rightSquare; 15      right -= 1; 16    } 17    highestSquareIdx -= 1; 18  } 19 20  return squares; 21 } 22 23 24 console.log(`Squares: \${make_squares([-2, -1, 0, 25 console.log(`Squares: \${make_squares([-3, -1, 0,</pre>			
<div> </div>			

## Time complexity #

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

## Space complexity #

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

[← Back](#)[Next →](#)[Remove Duplicates \(easy\)](#)[Triplet Sum to Zero \(medium\)](#)[Mark as Completed](#)[Report  
an Issue](#)[Ask a Question](#)

([https://discuss.educative.io/tag/squaring-a-sorted-array-easy\\_\\_pattern-two-pointers\\_\\_grokking-the-coding-interview-patterns-for-coding-questions](https://discuss.educative.io/tag/squaring-a-sorted-array-easy__pattern-two-pointers__grokking-the-coding-interview-patterns-for-coding-questions))