

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 def make_squares(arr):
2     squares = []
3     # TODO: Write your code here
4     return squares
5
```

Test

Save

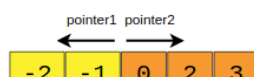
Reset



Solution

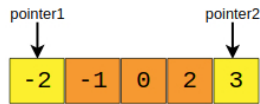
This is a straightforward question. The only trick is that we can have negative numbers in the input array, which will make it a bit difficult to generate the output array with squares in sorted order.

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 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

```
1 def make_squares(arr):
2     n = len(arr)
3     squares = [0 for x in range(n)]
4     highestSquareIdx = n - 1
5     left, right = 0, n - 1
6     while left <= right:
7         leftSquare = arr[left] * arr[left]
8         rightSquare = arr[right] * arr[right]
9         if leftSquare > rightSquare:
10            squares[highestSquareIdx] = leftSquare
11            left += 1
12        else:
13            squares[highestSquareIdx] = rightSquare
14            right -= 1
15            highestSquareIdx -= 1
16    return squares
17
18
19
20 def main():
21
22     print("Squares: " + str(make_squares([-2, -1, 0, 2, 3])))
23     print("Squares: " + str(make_squares([-3, -1, 0, 1, 2])))
24
25
26 main()
27
```

Run Save Reset

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 \(medium\)](#)

✓ Completed

