Question: https://leetcode.com/problems/squares-of-a-sorted-array/

We are provided with a non decreasing array of integers, and we need to create an array of square of the elements from the given array in non decreasing order.

So the naive approach would be to create an array containing the square of the elements, and then sorting the array.

The best sorting technique would also use atleast O(nlogn) time complexity to solve it.

So lets optimize it!

In order to do so we need to make it clear what we are doing, what we are given, and what will we achieve.

1. We are given non decreasingly ordered array of integers.
2. We need to find the square of elements and place them in non decreasing order in another array.
3. Our result will be an array of square, now we know that square of any integer will return a positive number.

So here lies the problem that if the array is [-4,1,3]

Then simply squaring the elements would give [16,1,9]

So seeing this we can conclude one thing either the square of smallest element of original array would be the largest square or the square of greatest element of original array would be the largest square.

So if we compare the absolute values of the smallest and largest element in the array and accordingly place the greatest element’s square in the last position of result array then there would be no need to sort it after wards.

Solution:  
class Solution {

public int[] sortedSquares(int[] A) {

//result array

int[] res = new int[A.length];

//pointers for left and right

int lo = 0; int hi = A.length - 1;

//iterate from n to 0

for (int i = A.length - 1; i >= 0; i--) {

//check if abs left is less than or equal to abs right

if (Math.abs(A[lo]) >= Math.abs(A[hi])) {

//add left squared to result array

res[i] = A[lo] \* A[lo];

//increment left pointer

lo++;

} else {

//add right squared to result array

res[i] = A[hi] \* A[hi];

//decrement right pointer

hi--;

}

}

//result

return res;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>