CALLICODER

Squares of a Sorted array

by RAJEEV SINGH · ALGORITHMS · AUGUST 03, 2021 · 1 MINS READ

Problem Statement

Given an array of numbers sorted in increasing order, return a new array containing squares of all the numbers of the input array sorted in increasing order.

Example 1:

```
Input: a[] = [-5, -2, -1, 0, 4, 6]
Output: [0, 1, 4, 16, 25, 36]
Explanation: After squaring, the array becomes [25, 4, 1, 0, 16, 36].
After sorting, it becomes [0, 1, 4, 16, 25, 36].
```

Example 2:

```
Input: nums = [-5, -4, -3, -2, -1]
Output: [1, 4, 9, 16, 25]
```

Example 3:

```
Input: nums = [1, 2, 3, 4, 5]
Output: [1, 4, 9, 16, 25]
```

Solution

A simple approach is to square the array and then sort it to get the final array. This approach will have an `O(n*log n)` time complexity.

But given that the array is sorted, we can use the two-pointer approach to solve it in `O(n)` time-complexity.

Two pointer approach

If the array only had positive integers, then we could just square it and return the squared array as output. The only caveat with this question is that it has negative integers as well.

But if we can view the negative part and positive part of the array separately, and iterate over both the parts then we can get the squared array in `O(n)` time complexity.

Here is how the approach will work:

- Find the index of the first non-negative number in the array.

- Use two pointers to iterate over the array - one pointer will move forward to iterate over the non-negative numbers, and the other pointer will move backward to iterate over the negative numbers.
- At any step, we check which pointer gives a smaller square. We add the smaller square to the output array and advance the corresponding pointer.

```
  ←pointer1  pointer2→
[-5, -2, -1, 0, 4, 6]
```

```
class SquaresOfSortedArray {
    public static int[] sortedSquares(int[] a) {
        int n = a.length;
        int[] squaredArr = new int[n];
        int smallestSquareIdx = 0;

        // Find the index of first non-negative element
        int firstNonNegativeElementIndex = n;
        for(int i = 0; i < n; i++) {
            if(a[i] >= 0) {
                firstNonNegativeElementIndex = i;
                break;
            }
        }

        // Pointer to iterate over the negative elements
        int negItr = firstNonNegativeElementIndex-1;
        // Pointer to iterate over the non-negative elements
        int posItr = firstNonNegativeElementIndex;

        while(negItr >= 0 && posItr < n) {
            int negElementSquare = a[negItr]*a[negItr];
            int posElementSquare = a[posItr]*a[posItr];

            if(negElementSquare < posElementSquare) {
                squaredArr[smallestSquareIdx++] = negElementSquare;
                negItr--;
            } else {
                squaredArr[smallestSquareIdx++] = posElementSquare;
                posItr++;
            }
        }

        // Add the square of remaining negative elements to the output
        while(negItr >= 0) {
            squaredArr[smallestSquareIdx++] = a[negItr]*a[negItr];
            negItr--;
        }

        // Add the square of the remaining positive elements to the output
        while(posItr < n) {
            squaredArr[smallestSquareIdx++] = a[posItr]*a[posItr];
            posItr++;
        }
```

```java
        return squaredArr;
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int n = keyboard.nextInt();
        int[] a = new int[n];
        for(int i = 0; i < n; i++) {
            a[i] = keyboard.nextInt();
        }
        keyboard.close();

        int []squaredArray = sortedSquaresSimplified(a);
        for(int i = 0; i < squaredArray.length; i++) {
            System.out.print(squaredArray[i] + " ");
        }
        System.out.println();
    }
}
```

Two pointer approach simplified

An alternate two pointer approach is to use two pointers - one starting from the rightmost element of the array and the other starting from the leftmost element.

```
pointer1→                 ←pointer2
       [-5, -2, -1, 0, 4, 6]
```

At every step, check which pointer gives the highest square. Add the highest square to the output array (starting from right) and advance the corresponding pointer.

```
class SquaresOfSortedArray {
    public static int[] sortedSquaresSimplified(int[] a) {
        int n = a.length;
        int[] squaredArr = new int[n];
        int highestSquareIdx = n-1;

        int left = 0;
        int right = n-1;

        while(left <= right) {
            int leftSquare = a[left]*a[left];
            int rightSquare = a[right]*a[right];

            if(leftSquare > rightSquare) {
                squaredArr[highestSquareIdx--] = leftSquare;
                left++;
            } else {
                squaredArr[highestSquareIdx--] = rightSquare;
                right--;
            }
        }

        return squaredArr;
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int n = keyboard.nextInt();
        int[] a = new int[n];
        for(int i = 0; i < n; i++) {
            a[i] = keyboard.nextInt();
        }
        keyboard.close();

        int []squaredArray = sortedSquaresSimplified(a);
        for(int i = 0; i < squaredArray.length; i++) {
            System.out.print(squaredArray[i] + " ");
        }
        System.out.println();
    }
}
```

Share on social media

What do you think?

5 Responses

Like Love Wow Sad Angry

0 Comments **CalliCoder** **Disqus' Privacy Policy** 1 **Login**

Favorite Tweet Share **Sort by Newest**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Do Not Sell My Data**