Codility_

CodeCheck Report: trainingKPWV6P-4R4

Test Name:

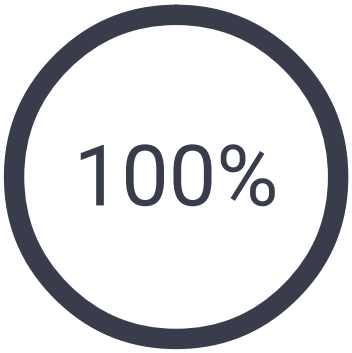
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
AbsDistinct Java 8	1 min	100%

Total score



Tasks Details

Easy

1. AbsDistinct

Compute number of distinct absolute values of sorted array elements.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A non-empty array A consisting of N numbers is given. The array is sorted in non-decreasing order. The *absolute distinct count* of this array is the number of distinct absolute values among the elements of the array.

For example, consider array A such that:

```
A[0] = −5
A[1] = −3
A[2] = −1
A[3] =  0
A[4] =  3
A[5] =  6
```

The absolute distinct count of this array is 5, because there are 5 distinct absolute values among the elements of this array, namely 0, 1, 3, 5 and 6.

Write a function:

```
class Solution { public int solution(int[] A); }
```

that, given a non-empty array A consisting of N numbers, returns absolute distinct count of array A.

Solution

Programming language used:	Java 8
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline

22:55:43 22:56:31

Code: 22:56:31 UTC, java, final, score: **100**

show code in pop-up

```
1  // you can also use imports, for example:
2  // import java.util.*;
3
```

For example, given array A such that:

```
A[0] = −5
A[1] = −3
A[2] = −1
A[3] =  0
A[4] =  3
A[5] =  6
```

the function should return 5, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..100,000];
- each element of array A is an integer within the range [−2,147,483,648..2,147,483,647];
- array A is sorted in non-decreasing order.


```
4  // you can write to stdout for debugging purpo
5  // System.out.println("this is a debug message
6
7  class Solution {
8      public int solution(int[] A) {
9          int count = 0;
10
11         int leftIndex = 0;
12         int rightIndex = A.length - 1;
13         if (A[0] == Integer.MIN_VALUE) {
14             count++;
15             for (; leftIndex <= rightIndex; le
16                 if (A[leftIndex] != Integer.MI
17                     break;
18                 }
19             }
20         }
21
22         for (int i = leftIndex; i <= rightInde
23             A[i] = -A[i];
24         }
25
26         int currentAbs = Math.max(A[leftIndex]
27         while (currentAbs >= 0) {
28             count++;
29
30             int nextLeft = -1;
31             while (leftIndex <= rightIndex) {
32                 if (A[leftIndex] == currentAbs
33                     leftIndex++;
34                 } else {
35                     nextLeft = A[leftIndex];
36                     break;
37                 }
38             }
39
40             int nextRight = -1;
41             while (rightIndex > leftIndex) {
42                 if (A[rightIndex] == currentAb
43                     rightIndex--;
44                 } else {
45                     nextRight = A[rightIndex];
46                     break;
47                 }
48             }
49
50             currentAbs = Math.max(nextLeft, ne
51         }
52
53         return count;
54     }
55 }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N) or O(N*log(N))**

collapse all	Example tests	
▼ example example test		✔ OK
1. 0.004 s OK		

collapse all — Correctness tests

▼ one_element ✔ OK

1. 0.004 s OK
2. 0.004 s OK
3. 0.004 s OK

▼ two_elements ✔ OK

1. 0.004 s OK
2. 0.004 s OK
3. 0.004 s OK
4. 0.004 s OK
5. 0.004 s OK
6. 0.004 s OK
7. 0.004 s OK
8. 0.004 s OK
9. 0.008 s OK

▼ same_elements ✔ OK

1. 0.004 s OK
2. 0.004 s OK
3. 0.008 s OK

▼ simple ✔ OK

1. 0.008 s OK

▼ simple_no_zero ✔ OK

1. 0.008 s OK

▼ simple_no_same ✔ OK

1. 0.008 s OK

▼ simple_no_negative ✔ OK

1. 0.004 s OK

▼ simple_no_positive ✔ OK

1. 0.004 s OK

▼ arith_overlow ✔ OK

1. 0.008 s OK
2. 0.008 s OK
3. 0.008 s OK

▼ medium_chaotic1 ✔ OK

1. 0.004 s OK

▼ medium_chaotic2 ✔ OK

1. 0.004 s OK

collapse all — Performance tests

▼ long_sequence_no_negative ✔ OK

1. 0.132 s OK

▼ long_sequence_no_positive	✔ OK
1. 0.108 s OK	
▼ long_sequence	✔ OK
1. 0.240 s OK	