Codility_

Candidate Report: trainingE5F8UM-N9S

Check out Codility training tasks

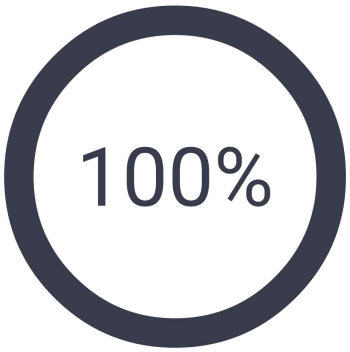
Test Name:

Summary Review (0) Timeline

Tasks summary

Task	Time spent	Score
PermCheck Java 8	1 min	100%

Total score



Tasks Details

Easy

1. PermCheck

Check whether array A is a permutation.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A non-empty array A consisting of N integers is given.

A *permutation* is a sequence containing each element from 1 to N once, and only once.

For example, array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
A[3] = 2
```

is a permutation, but array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
```

is not a permutation, because value 2 is missing.

The goal is to check whether array A is a permutation.

Write a function:

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

that, given an array A, returns 1 if array A is a permutation and 0 if it is not.

For example, given array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
A[3] = 2
```

Solution

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

Task timeline

20:17:46 20:18:33

Code: 20:18:33 UTC, java, final, score: **100** show code in pop-up

```
1  // you can also use imports, for example:
2  // import java.util.*;
3
4  // you can write to stdout for debugging purposes, e.g.
5  // System.out.println("this is a debug message");
6  import java.util.Arrays;
7
8  class Solution {
9      public int solution(int[] A) {
```

the function should return 1.

Given array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
```

the function should return 0.

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 [1..1,000,000,000].


```
10          int expectedInt = 1, missingInt = 0, result = 0;
11
12                  //Check if the length is zero
13                  if (A.length == 0) {
14                          result = 0;
15                  }
16                  else if( A.length > 0 ){
17
18                          //sort the array
19                          Arrays.sort(A);
20
21                      for (int x : A){ //loop to find the mi
22                          if (x == expectedInt){
23                              expectedInt++;
24                          } else {
25                              missingInt = expectedInt;
26                              break;
27                          }
28                      }
29                  }
30                  //final check of the missing int
31                  result = missingInt > 0 ? 0 : 1;
32
33                  return result;
34      }
35  }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example1 the first example test	✓ OK
▶	example2 the second example test	✓ OK
expand all	Correctness tests	
▶	extreme_min_max single element with minimal/maximal value	✓ OK
▶	single single element	✓ OK
▶	double two elements	✓ OK
▶	antiSum1 total sum is correct, but it is not a permutation, N <= 10	✓ OK
▶	small_permutation permutation + one element occurs twice, N = ~100	✓ OK
▶	permutations_of_ranges permutations of sets like [2..100] for which the anwsers should be false	✓ OK
expand all	Performance tests	
▶	medium_permutation permutation + few elements occur twice, N = ~10,000	✓ OK
▶	antiSum2 total sum is correct, but it is not a	✓ OK

	permutation, N = ~100,000	
▶	large_not_permutation permutation + one element occurs three times, N = ~100,000	✓ OK
▶	large_range sequence 1, 2, ..., N, N = ~100,000	✓ OK
▶	extreme_values all the same values, N = ~100,000	✓ OK
▶	various_permutations all sequences are permutations	✓ OK