codility

Check out Codility training tasks

Candidate Report: Anonymous

Test Name:

Summary Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

	Time Spent	Task Score
PermCheck Submitted in: Java 8	1 min	100%

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
```

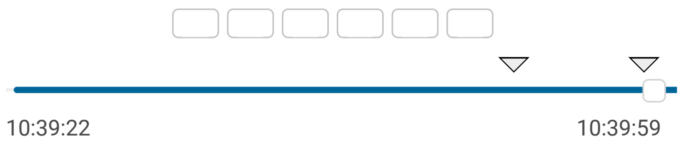
the function should return 1.

Given array A such that:

Solution

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

Task timeline



Code: 10:39:59 UTC, java, final, score: 100 show code in pop-up

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

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
    public static int max = 1000000;
        public static boolean[] mem;

    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].

```
	public static int solution(int[] a) {
		mem = new boolean[max + 1];

		for (int i = 0; i < a.length; i++) {
			if (a[i] > a.length)
				return 0;
			mem[a[i]] = true;
		}

		for (int i = 1; i < a.length + 1; i++) {
			if (!mem[i])
				return 0;
		}
		return 1;
	}
}
```

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 permutation, N = ~100,000	✓ OK
▶	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