Codility_

Candidate Report: trainingFGU4UV-UY4

Check out Codility training tasks

Test Name:

Summary Review (0) Timeline

Tasks summary

Task	Time spent	Score
CountNonDivisible Java 8	1 min	100%

Total score

100%

Tasks Details

Medium

1. CountNonDivisible
Calculate the number of elements of an array that are not divisors of each element.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are given an array A consisting of N integers.

For each number A[i] such that 0 ≤ i < N, we want to count the number of elements of the array that are not the divisors of A[i]. We say that these elements are non-divisors.

For example, consider integer N = 5 and array A such that:

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

For the following elements:

- A[0] = 3, the non-divisors are: 2, 6,
- A[1] = 1, the non-divisors are: 3, 2, 3, 6,
- A[2] = 2, the non-divisors are: 3, 3, 6,
- A[3] = 3, the non-divisors are: 2, 6,
- A[4] = 6, there aren't any non-divisors.

Write a function:

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

that, given an array A consisting of N integers, returns a sequence of integers representing the amount of non-divisors.

Result array should be returned as an array of integers.

Solution

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

Task timeline

09:49:47 09:50:30

Code: 09:50:30 UTC, java, final, score: **100**

show code in pop-up

```java
// 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");
import java.util.HashMap;
class Solution {
```

For example, given:

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

the function should return [2, 4, 3, 2, 0], as explained above.

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

- N is an integer within the range [1..50,000];
- each element of array A is an integer within the range [1..2 * N].

```
    public int[] solution(int[] A) {

            int N = A.length;
            HashMap<Integer, Integer> count = new HashMap<

            for (int i : A) {
                Integer key = count.get(i);
                if (key != null) {
                    count.put(i, key + 1);
                } else {
                    count.put(i, 1);
                }
            }

            HashMap<Integer, Integer> divs = new HashMap<>
            for (Integer n : count.keySet()) {
                int sum = 0;
                int j = 1;
                while (j * j <= n) {
                    if (n % j == 0) {
                        if (count.containsKey(j)) {
                            sum += count.get(j);
                        }
                        //find n = j*k cases to add both t
                        int k = n / j;
                        if (k != j) {
                            if (count.containsKey(k)) {
                                sum += count.get(k);
                            }
                        }
                    }
                    j++;
                }

                divs.put(n, N - sum);
            }

            for (int i = 0; i < A.length; i++) {
                A[i] = divs.get(A[i]);
            }

            return A;
        }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example example test	✓ OK
expand all	Correctness tests	
▶	extreme_simple extreme simple	✓ OK
▶	double two elements	✓ OK
▶	simple simple tests	✓ OK
▶	primes prime numbers	✓ OK
▶	small_random small, random numbers, length = 100	✓ OK
expand all	Performance tests	

▶	medium_random medium, random numbers length = 5,000	✓ OK
▶	large_range 1, 2, ..., N, length = ~20,000	✓ OK
▶	large_random large, random numbers, length = ~30,000	✓ OK
▶	large_extreme large, all the same values, length = 50,000	✓ OK