Codility_

CodeCheck Report: trainingTJ9TBA-MDB

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
CountTriangles Java 8 ⚠	1 min	100%

Total score

100%

Tasks Details

1. CountTriangles

Easy

Count the number of triangles that can be built from a given set of edges.

Task Score	Correctness	Performance
100%	100%	100%

Task description

An array A consisting of N integers is given. A triplet (P, Q, R) is *triangular* if it is possible to build a triangle with sides of lengths A[P], A[Q] and A[R]. In other words, triplet (P, Q, R) is triangular if 0 ≤ P < Q < R < N and:

- A[P] + A[Q] > A[R],
- A[Q] + A[R] > A[P],
- A[R] + A[P] > A[Q].

For example, consider array A such that:

```
A[0] = 10    A[1] = 2    A[2] = 5
A[3] = 1     A[4] = 8    A[5] = 12
```

There are four triangular triplets that can be constructed from elements of this array, namely (0, 2, 4), (0, 2, 5), (0, 4, 5), and (2, 4, 5).

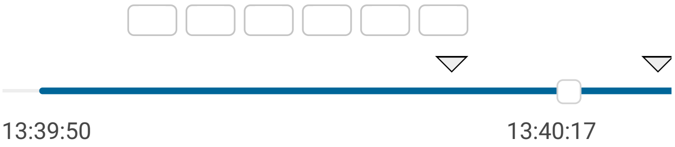
Write a function:

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

Solution

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

Task timeline



Code: 13:40:16 UTC, java, final, score: **100**

show code in pop-up

that, given an array A consisting of N integers, returns the number of triangular triplets in this array.

For example, given array A such that:

```
A[0] = 10    A[1] = 2    A[2] = 5
A[3] = 1     A[4] = 8    A[5] = 12
```

the function should return 4, as explained above.

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

- N is an integer within the range [0..1,000];
- each element of array A is an integer within the range [1..1,000,000,000].

```java
import java.util.*;

class Solution {
    public int solution(int[] A) {

        int N = A.length;
        Arrays.sort(A);

        int count = 0;

        for (int i = 0; i < N - 2; i++) {
            int k = i + 2;
            for (int j = i + 1; j < N - 1; j
                int maxValue = A[i] + A[j];
                while (k < N && A[k] < maxVa
                    k++;
                }
                count += k - j - 1;
            }
        }


        return count;
    }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N**2)**

collapse all Example tests

▼ example ✔ OK

example, positive answer, length=6

1. 0.004 s **OK**

collapse all Correctness tests

▼ extreme_empty ✔ OK

empty sequence + [5,3,3]

1. 0.008 s **OK**
2. 0.008 s **OK**

▼ extreme_single ✔ OK

1-element sequence + [5,3,3]

1. 0.008 s **OK**
2. 0.004 s **OK**

▼ extreme_two_elems ✔ OK

2-element sequence + [5,3,3]

1. 0.004 **OK**

s

2. 0.004 s **OK**

▼ extreme_arith_overflow ✔ **OK**
overflow test, 3 MAXINTs + [5,3,3]

1. 0.004 s **OK**
2. 0.004 s **OK**

▼ simple ✔ **OK**

1. 0.008 s **OK**

▼ medium1 ✔ **OK**
chaotic sequence of values from [1..100K], length=30

1. 0.004 s **OK**

▼ medium2 ✔ **OK**
chaotic sequence of values from [1..1K], length=50

1. 0.004 s **OK**

collapse all

Performance tests

▼ large ✔ **OK**
chaotic sequence with values from [1..10], length=200

1. 0.008 s **OK**

▼ large2 ✔ **OK**
1 followed by an ascending sequence of ~1K elements from [1..2K]

1. 0.028 s **OK**

▼ large_random ✔ **OK**
chaotic sequence of values from [1..1M], length=1K

1. 0.032 s **OK**

▼ large_the_same ✔ **OK**
sequence of the same value value

1. 0.024 s **OK**
2. 0.028 s **OK**