Codility_

CodeCheck Report: trainingXK4NDX-NE7

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
CountTriangles ⚠ Python	19 min	100%

Total score

100%

Tasks Details

		Task Score	Correctness	Performance
Easy	1. **CountTriangles** Count the number of triangles that can be built from a given set of edges.	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).

Solution

Programming language used:	Python
Total time used:	19 minutes ?
Effective time used:	19 minutes ?
Notes:	*not defined yet*

Task timeline ?

Write a function:

```
def solution(A)
```

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].

09:05:59 09:24:32

Code: 09:24:32 UTC, py, final, score: **100**

show code in pop-up

```
# you can write to stdout for debugging p
# print("this is a debug message")

def solution(A):
    # Implement your solution here
    # pass
    N = len(A)
    A.sort()
    count = 0
    for P in range(N):
        R = P + 2
        for Q in range(P + 1, N):
            while R < N and A[P] + A[Q] >
                R += 1
            count += max(0, R - Q - 1)
    return count

```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all **Example tests**

▶ example ✔ **OK**

example, positive answer, length=6

expand all **Correctness tests**

▶ extreme_empty ✔ **OK**

empty sequence + [5,3,3]

▶ extreme_single ✔ **OK**

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

▶ extreme_two_elems ✔ **OK**

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

▶ extreme_arith_overflow ✔ **OK**

overflow test, 3 MAXINTs + [5,3,3]

▶ simple ✔ **OK**

▶ medium1 ✔ **OK**

chaotic sequence of values from [1..100K], length=30

▶ medium2 ✔ **OK**

chaotic sequence of values from [1..1K], length=50

expand all **Performance tests**

▶ large ✔ **OK**

chaotic sequence with values from [1..10], length=200

▶	large2 1 followed by an ascending sequence of ~1K elements from [1..2K]	✔ OK
▶	large_random chaotic sequence of values from [1..1M], length=1K	✔ OK
▶	large_the_same sequence of the same value value	✔ OK