Codility_

CodeCheck Report: trainingGC8RP2-CQ9

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
CountBoundedSlices Python	11 min	60%

Total score

Tasks Details

Medium

1. CountBoundedSlices

Calculate the number of slices in which (maximum - minimum <= K).

Task Score	Correctness	Performance
60%	100%	20%

Task description

An integer K and a non-empty array A consisting of N integers are given.

A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a *slice* of array A.

A *bounded slice* is a slice in which the difference between the maximum and minimum values in the slice is less than or equal to K. More precisely it is a slice, such that max(A[P], A[P + 1], ..., A[Q]) − min(A[P], A[P + 1], ..., A[Q]) ≤ K.

The goal is to calculate the number of bounded slices.

For example, consider K = 2 and array A such that:

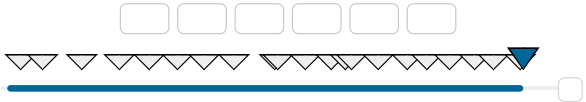
```
A[0] = 3
A[1] = 5
A[2] = 7
A[3] = 6
A[4] = 3
```

There are exactly nine bounded slices: (0, 0), (0, 1), (1, 1), (1, 2),

Solution

Programming language used:	Python
Total time used:	11 minutes
Effective time used:	11 minutes
Notes:	*not defined yet*

Task timeline



12:57:11 13:08:11

(1, 3), (2, 2), (2, 3), (3, 3), (4, 4).

Write a function:

```
def solution(K, A)
```

that, given an integer K and a non-empty array A of N integers, returns the number of bounded slices of array A.

If the number of bounded slices is greater than 1,000,000,000, the function should return 1,000,000,000.

For example, given:

```
A[0] = 3
A[1] = 5
A[2] = 7
A[3] = 6
A[4] = 3
```

the function should return 9, as explained above.

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

- N is an integer within the range [1..100,000];
- K is an integer within the range [0..1,000,000,000];
- each element of array A is an integer within the range [−1,000,000,000..1,000,000,000].

Code: 13:08:10 UTC, py, final, score: **60** show code in pop-up

```
1  # you can write to stdout for debugging purp
2  # print("this is a debug message")
3
4  def solution(K, A):
5      # Implement your solution here
6      N = len(A)
7      count = 0
8      end = 0
9
10     for start in range(N):
11         while end < N and max(A[start:end+1]
12             end += 1
13         count += end - start
14
15         if count > 1_000_000_000:
16             return 1_000_000_000
17
18     return count
```

Analysis summary

The following issues have been detected: timeout errors.

Analysis

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

expand all	Example tests	
▶	example example test	✔ OK
expand all	**Correctness tests**	
▶	single single element	✔ OK
▶	double two elements	✔ OK
▶	small_functional small functional tests	✔ OK
▶	small_random small random sequences length = ~100	✔ OK
▶	small_random2 small random sequences length = ~100	✔ OK
expand all	**Performance tests**	
▶	medium_random chaotic medium sequences length = ~3,000	✘ TIMEOUT ERROR running time: 0.188 sec., time limit: 0.100 sec.
▶	large_range large range test, length = ~100,000	✘ TIMEOUT ERROR Killed. Hard limit reached: 6.000 sec.
▶	large_random random large sequences length = ~100,000	✔ OK
▶	large_answer	✘ TIMEOUT ERROR

test with large answer	Killed. Hard limit reached: 6.000 sec.
▶ large_extreme all maximal value = ~100,000	✘ TIMEOUT ERROR Killed. Hard limit reached: 6.000 sec.