Codility_

CodeCheck Report: training6AH2W2-X6Q

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Peaks Python ⚠	2 min	100%

Total score

100%

Tasks Details

Medium	1. **Peaks** Divide an array into the maximum number of same-sized blocks, each of which should contain an index P such that A[P - 1] < A[P] > A[P + 1].	**Task Score**	**Correctness**	**Performance**
		100%	100%	100%

Task description

Solution

A non-empty array A consisting of N integers is given.

A *peak* is an array element which is larger than its neighbors. More precisely, it is an index P such that 0 < P < N − 1, A[P − 1] < A[P] and A[P] > A[P + 1].

For example, the following array A:

```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 3
A[5] = 4
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
A[10] = 6
A[11] = 2
```

has exactly three peaks: 3, 5, 10.

We want to divide this array into blocks containing the same number of elements. More precisely, we want to choose a number K that will yield the following blocks:

- A[0], A[1], ..., A[K − 1],
- A[K], A[K + 1], ..., A[2K − 1],

  ...
- A[N − K], A[N − K + 1], ..., A[N − 1].

What's more, every block should contain at least one peak. Notice that extreme elements of the blocks (for example A[K − 1] or A[K]) can also be peaks, but only if they have both neighbors (including one in an adjacent blocks).

The goal is to find the maximum number of blocks into which the array A can be divided.

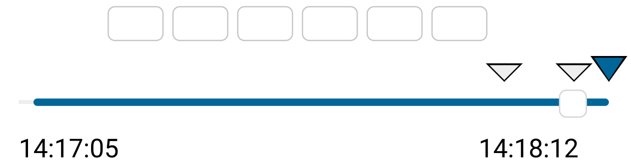
Array A can be divided into blocks as follows:

- one block (1, 2, 3, 4, 3, 4, 1, 2, 3, 4, 6, 2). This block contains three peaks.
- two blocks (1, 2, 3, 4, 3, 4) and (1, 2, 3, 4, 6, 2). Every block has a peak.
- three blocks (1, 2, 3, 4), (3, 4, 1, 2), (3, 4, 6, 2). Every block has a peak. Notice in particular that the first block (1, 2, 3, 4) has a peak at A[3], because A[2] < A[3] > A[4], even though A[4] is in the adjacent block.

However, array A cannot be divided into four blocks, (1, 2, 3), (4, 3, 4), (1, 2, 3) and (4, 6, 2), because the (1, 2, 3) blocks do not contain a peak. Notice in particular that the (4, 3, 4) block contains two peaks: A[3] and A[5].

Programming language used:	Python
Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Task timeline



Code: 14:18:12 UTC, py, final, score: **100**

show code in pop-up

```
# Source
# https://github.com/Dineshkarthik/co


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

def solution(A):
    # Implement your solution here
    # pass
    length = len(A)

    # array ends can't be peaks, len
    if length < 3:
        return 0

    peaks = [0] * length

    # compute a list of 'peaks to the
    for index in range(2, length):
        peaks[index] = peaks[index -

        # check if there was a peak t
        if A[index - 1] > A[index - 2
            peaks[index] += 1

    # candidate is the block size we'
    for candidate in range(3, length

        # skip if not a factor
        if length % candidate != 0:
            continue

        # test at each point n / bloc
        valid = True
        index = candidate
        while index != length:

            # if no peak in this bloc
            if peaks[index] == peaks[
                valid = False
                break
```

The maximum number of blocks that array A can be divided into is three.

Write a function:

```
def solution(A)
```

that, given a non-empty array A consisting of N integers, returns the maximum number of blocks into which A can be divided.

If A cannot be divided into some number of blocks, the function should return 0.

For example, given:

```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 3
A[5] = 4
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
A[10] = 6
A[11] = 2
```

the function should return 3, as explained above.

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 [0..1,000,000,000].

```

            index += candidate

        # one additional check since
        if index == length and peaks[
            valid = False

        if valid:
            return length // candidat

    return 0
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	**Example tests**	
▶	example example test	✔ OK
expand all	**Correctness tests**	
▶	extreme_min extreme min test	✔ OK
▶	extreme_without_peaks test without peaks	✔ OK
▶	prime_length test with prime sequence length	✔ OK
▶	anti_bin_search anti bin_search test	✔ OK
▶	simple1 simple test	✔ OK
▶	simple2 second simple test	✔ OK
expand all	**Performance tests**	
▶	medium_random chaotic medium sequences, length = ~5,000	✔ OK
▶	medium_anti_slow medium test anti slow solutions	✔ OK
▶	large_random chaotic large sequences, length = ~50,000	✔ OK

▶	large_anti_slow large test anti slow solutions	✔ OK
▶	extreme_max extreme max test	✔ OK