Codility_

# CodeCheck Report: trainingS6SDST-D5T

Test Name:

Check out Codility training tasks

Summary Timeline

## Tasks summary

| Task | Time spent | Score |
|---|---|---|
| Peaks<br>C# ⚠ | 63 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

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].

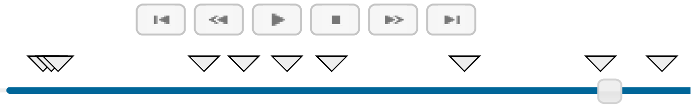
For example, the following array A:

```
    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
```

### Solution

| | |
|---|---|
| Programming language used: | C# |
| Total time used: | 63 minutes ? |
| Effective time used: | 63 minutes ? |
| Notes: | *not defined yet* |

### Task timeline ?



```
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].

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

Write a function:

```
class Solution { public int solution(int[] 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
```

00:54:35 01:56:55

Code: 01:56:55 UTC, cs, final, score: **100**

show code in pop-up

```
using System;
using System.Collections.Generic;

/**
 * 10.4 - Peaks
 * Paulo Santos
 * 15.Dec.2022
 */
class Solution {
    public int solution(int[] A) {

        /*
         * List all the peakes
         */
        var len = A.Length;
        var peaks = new List<int>();
        for (var i = 1 ; i < len - 1; i++)
            if ((A[i - 1] < A[i]) && (A[i] > A[i +
                peaks.Add(i);

        if (peaks.Count == 0)
            return 0;

        for (var size = peaks.Count; size >= 0; siz
            if ((A.Length % size) == 0) {
                var blockSize = A.Length / size;
                var found = new bool[size];
                var foundCnt = 0;
                foreach(var p in peaks) {
                    var blkNum = p / blockSize;
                    if (!found[blkNum]) {
                        found[blkNum] = true;
                        foundCnt += 1;
                    }
                }

                if (foundCnt == size)
                    return size;
            }
        }
        return 0;
    }
}
```

## Analysis summary

The solution obtained perfect score.

## Analysis

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

expand all **Example tests**

▶

```
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].

| | |
|---|---|
| example<br>example test | ✓ OK |
| expand all | **Correctness tests** |
| ▶ extreme_min<br>extreme min test | ✓ OK |
| ▶ extreme_without_peaks<br>test without peaks | ✓ OK |
| ▶ prime_length<br>test with prime sequence length | ✓ OK |
| ▶ anti_bin_search<br>anti bin_search test | ✓ OK |
| ▶ simple1<br>simple test | ✓ OK |
| ▶ simple2<br>second simple test | ✓ OK |
| expand all | **Performance tests** |
| ▶ medium_random<br>chaotic medium sequences, length = ~5,000 | ✓ OK |
| ▶ medium_anti_slow<br>medium test anti slow solutions | ✓ OK |
| ▶ large_random<br>chaotic large sequences, length = ~50,000 | ✓ OK |
| ▶ large_anti_slow<br>large test anti slow solutions | ✓ OK |
| ▶ extreme_max<br>extreme max test | ✓ OK |