Codility_

CodeCheck Report: training4A6Y8N-CM7

Test Name:

Check out Codility training tasks

Summary | Timeline

Tasks summary

Task	Time spent	Score
StoneWall C#	35 min	100%

Total score

Tasks Details

Easy

1. StoneWall

Cover "Manhattan skyline" using the minimum number of rectangles.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are going to build a stone wall. The wall should be straight and N meters long, and its thickness should be constant; however, it should have different heights in different places. The height of the wall is specified by an array H of N positive integers. H[I] is the height of the wall from I to I+1 meters to the right of its left end. In particular, H[0] is the height of the wall's left end and H[N−1] is the height of the wall's right end.

The wall should be built of cuboid stone blocks (that is, all sides of such blocks are rectangular). Your task is to compute the minimum number of blocks needed to build the wall.

Write a function:

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

that, given an array H of N positive integers specifying the height of the wall, returns the minimum number of blocks needed to build it.

For example, given array H containing N = 9 integers:

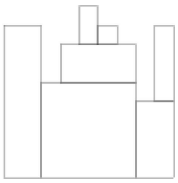
Solution

Programming language used:	C#
Total time used:	35 minutes
Effective time used:	35 minutes
Notes:	*not defined yet*

Task timeline

```
H[0] = 8    H[1] = 8    H[2] = 5
H[3] = 7    H[4] = 9    H[5] = 8
H[6] = 7    H[7] = 4    H[8] = 8
```

the function should return 7. The figure shows one possible arrangement of seven blocks.



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

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

Code: 08:49:58 UTC, cs, final, score: **100** show code in pop-up

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

/**
 * 7.4 - Stonewall
 * Paulo Santos
 * 09.Dec.2022
 */
class Solution {
    public int solution(int[] H) {

        /*
         * Number of blocks
         */
        var lst = new List<Tuple<int, int>>();

        /*
         * The start of the wall
         */
        var blk = new Tuple<int, int>(0, H[0]);
        var stk = new Stack<Tuple<int, int>>();

        lst.Add(blk);
        stk.Push(blk);

        for(var i = 1; i < H.Length; i++) {
            /*
             * Check if the stack of
             * blocks has the same height
             * of the wall
             */
            if (H[i] == stk.Peek().Item2)
                continue;

            /*
             * Is it's height greater?
             */
            if (H[i] > stk.Peek().Item2) {
                /*
                 * Add another block
                 */
                blk = new Tuple<int, int>(stk.Peek(
                lst.Add(blk);
                stk.Push(blk);
                continue;
            }

            /*
             * It's smaller.
             * Go through the stack to see
             * If we can find a smaller block
             */
            blk = null;
            stk.Pop();
            while (stk.Count > 0) {
                if (H[i] == stk.Peek().Item2)
                {
                    blk = stk.Peek();
                    break;
                }
                if (H[i] > stk.Peek().Item2) {
                    blk = new Tuple<int, int>(stk.P
                    lst.Add(blk);
                    break;
                }
                stk.Pop();
            }

            if (blk == null) {
                blk = new Tuple<int, int>(0, H[i]);
```

```
71                  lst.Add(blk);
72              }
73
74              stk.Push(blk);
75          }
76
77          return lst.Count;
78      }
79  }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example	✓ OK
expand all	**Correctness tests**	
▶	simple1	✓ OK
▶	simple2	✓ OK
▶	simple3	✓ OK
▶	simple4	✓ OK
▶	boundary_cases	✓ OK
expand all	**Performance tests**	
▶	medium1	✓ OK
▶	medium2	✓ OK
▶	medium3	✓ OK
▶	medium4	✓ OK
▶	large_piramid	✓ OK
▶	large_increasing_decreasing	✓ OK
▶	large_up_to_20	✓ OK
▶	large_up_to_100	✓ OK
▶	large_max	✓ OK