



CodeCheck Report: trainingSDMNM2-YCS

[Check out Codility training tasks](#)

Test Name:

Summary    Timeline

Tasks summary

Task	Time spent	Score
Fish C#	67 min	100%

Total score

100%

Tasks Details

Easy	1. Fish	Task Score	Correctness	Performance	
	N voracious fish are moving along a river. Calculate how many fish are alive.				
		100%	100%	100%	

Task description

You are given two non-empty arrays A and B consisting of N integers. Arrays A and B represent N voracious fish in a river, ordered downstream along the flow of the river.

The fish are numbered from 0 to N - 1. If P and Q are two fish and P < Q, then fish P is initially upstream of fish Q. Initially, each fish has a unique position.

Fish number P is represented by A[P] and B[P]. Array A contains the sizes of the fish. All its elements are unique. Array B contains the directions of the fish. It contains only 0s and/or 1s, where:

- 0 represents a fish flowing upstream,
- 1 represents a fish flowing downstream.

If two fish move in opposite directions and there are no other (living) fish between them, they will eventually meet each other. Then only one fish can stay alive – the larger fish eats the smaller one. More precisely, we say that two fish P and Q meet each other

Solution

Programming language used:			C#
Total time used:	67 minutes		
Effective time used:	67 minutes		
Notes:	not defined yet		

Task timeline



when  $P < Q$ ,  $B[P] = 1$  and  $B[Q] = 0$ , and there are no living fish between them. After they meet:

- If  $A[P] > A[Q]$  then P eats Q, and P will still be flowing downstream,
- If  $A[Q] > A[P]$  then Q eats P, and Q will still be flowing upstream.

We assume that all the fish are flowing at the same speed. That is, fish moving in the same direction never meet. The goal is to calculate the number of fish that will stay alive.

For example, consider arrays A and B such that:

$A[0] = 4$	$B[0] = 0$
$A[1] = 3$	$B[1] = 1$
$A[2] = 2$	$B[2] = 0$
$A[3] = 1$	$B[3] = 0$
$A[4] = 5$	$B[4] = 0$

Initially all the fish are alive and all except fish number 1 are moving upstream. Fish number 1 meets fish number 2 and eats it, then it meets fish number 3 and eats it too. Finally, it meets fish number 4 and is eaten by it. The remaining two fish, number 0 and 4, never meet and therefore stay alive.

Write a function:

```
class Solution { public int solution(int[] A,
int[] B); }
```

that, given two non-empty arrays A and B consisting of N integers, returns the number of fish that will stay alive.

For example, given the arrays shown above, the function should return 2, 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  $[0..1,000,000,000]$ ;
- each element of array B is an integer that can have one of the following values: 0, 1;
- the elements of A are all distinct.

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Code: 14:53:00 UTC, cs, final, [show code in pop-up](#)  
score: 100

```
1 using System;
2 // you can also use other imports, for example:
3 // using System.Collections.Generic;
4
5 // you can write to stdout for debugging purposes,
6 // Console.WriteLine("this is a debug message");
7
8 using System;
9 using System.Collections.Generic;
10
11 class Solution {
12     public int solution(int[] A, int[] B) {
13         int N = A.Length;
14
15         Stack<int> upStream = new Stack<int>();
16         Stack<int> downStream = new Stack<int>();
17
18         for (int i = 0; i < N; i++) {
19             if ((B[i] == 0)) {
20                 upStream.Push(A[i]);
21
22                 while (downStream.Count > 0) {
23                     if (downStream.Peek() < A[i]) {
24                         downStream.Pop();
25                     } else {
26                         upStream.Pop();
27                         break;
28                     }
29                 }
30             } else {
31                 downStream.Push(A[i]);
32             }
33         }
34
35         return downStream.Count + upStream.Count;
36     }
37 }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

Example tests	
▶ example	✓ OK
example test	
Correctness tests	
▶ extreme_small	✓ OK
1 or 2 fishes	
▶ simple1	✓ OK
simple test	
▶ simple2	✓ OK
simple test	
▶	

Test results - Codility

small_random		✓ OK
small random test, N = ~100		
expand all	Performance tests	
▶	medium_random	✓ OK
small medium test, N = ~5,000		
▶	large_random	✓ OK
large random test, N = ~100,000		
▶	extreme_range1	✓ OK
all except one fish flowing in the same direction		
▶	extreme_range2	✓ OK
all fish flowing in the same direction		