Codility_

CodeCheck Report: training4JYJKA-XWZ

Test Name:

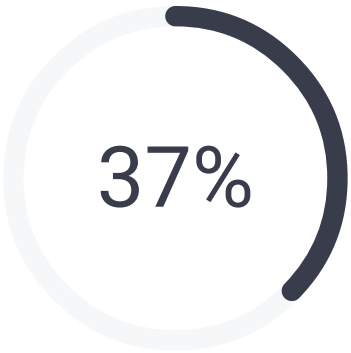
Check out Codility training tasks

Summary Timeline AI Assistant Transcript

Tasks summary

Task	Time spent	Score
Fish C++ ⚠	42 min	37%

Total score



Tasks Details

1. Fish

Easy

N voracious fish are moving along a river. Calculate how many fish are alive.

Task Score	Correctness	Performance
37%	50%	25%

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.

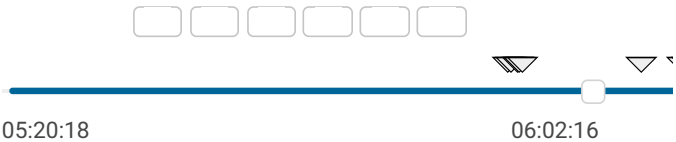
If two fish move in opposite directions and there are no other (living) fish between them, they will eventually meet each other. Then only one fish can stay alive − the larger fish eats the smaller one. More precisely, we say that two fish P and Q meet each other when P < Q, B[P] = 1 and B[Q] = 0, and 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,

Solution

Programming language used:	C++
Total time used:	42 minutes
Effective time used:	42 minutes
Notes:	*not defined yet*

Task timeline



Code: 06:02:16 UTC, cpp, final, score: **37** show code in pop-up

```
1  // you can use includes, for example:
2  // #include <algorithm>
```

- 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:

```
int solution(vector<int> &A, vector<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.


```
3
4   // you can write to stdout for debugging purpo
5   // cout << "this is a debug message" << endl;
6   // void print(vector<int>& v){
7   //     for (auto e:v) {
8   //         cout << e << " " << endl;
9   //     }
10  // }
11
12  int solution(vector<int> &A, vector<int> &B) {
13      vector<int> v;
14      int alive = 0;
15      for (int i = 0; i < B.size(); i++) {
16          if (B[i] == 1) {
17              v.push_back(A[i]);
18          }
19          else { //B[i] == 0
20              if (v.empty()) {
21                  alive++;
22              } else { // meet downstream fish
23                  while (!v.empty()) {
24                      if (A[i] < v.back()) breal
25                      else {
26                          v.pop_back();
27                          alive++;
28                      }
29                  }
30              }
31          }
32      }
33      return alive + v.size();
34  }
```

Analysis summary

The following issues have been detected: wrong answers.

Analysis

expand all **Example tests**

▶	example example test	✔ OK

expand all **Correctness tests**

▶	extreme_small 1 or 2 fishes	✔ OK
▶	simple1 simple test	✔ OK
▼	simple2 simple test	✘ WRONG ANSWER got 5 expected 1
	1. 0.001 s **WRONG ANSWER**, got 5 expected 1	
	2. 0.001 s **WRONG ANSWER**, got 5 expected 2	
▼	small_random small random test, N = ~100	✘ WRONG ANSWER got 44 expected 16
	1. 0.001 s **WRONG ANSWER**, got 44 expected 16	

expand all **Performance tests**

▼	medium_random small medium test, N = ~5,000	✘ WRONG ANSWER got 2531 expected 41
	1. 0.001 s **WRONG ANSWER**, got 2531 expected 41	
▼	large_random large random test, N = ~100,000	✘ WRONG ANSWER got 49989 expected 840

1. 0.012 s **WRONG ANSWER**, got 49989 expected 840

▼ extreme_range1 ✘ **WRONG ANSWER**
all except one fish flowing in the same direction got 99999 expected 1

1. 0.008 s **WRONG ANSWER**, got 99999 expected 1
2. 0.008 s **OK**

▶ extreme_range2 ✔ **OK**
all fish flowing in the same direction