Codility_

Candidate Report: Anonymous

Check out Codility training tasks

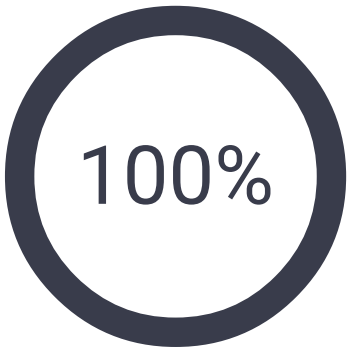
Test Name:

Summary Timeline Feedback

Tasks summary

Task	Time spent	Score
Dominator Swift 4	18 min	100%

Total score



Tasks Details

Easy

1. Dominator

Find an index of an array such that its value occurs at more than half of indices in the array.

Task Score	Correctness	Performance
100%	100%	100%

Task description

An array A consisting of N integers is given. The *dominator* of array A is the value that occurs in more than half of the elements of A.

For example, consider array A such that

```
A[0] = 3    A[1] = 4    A[2] =  3
A[3] = 2    A[4] = 3    A[5] = -1
A[6] = 3    A[7] = 3
```

The dominator of A is 3 because it occurs in 5 out of 8 elements of A (namely in those with indices 0, 2, 4, 6 and 7) and 5 is more than a half of 8.

Write a function

```
public func solution(_ A : inout [Int]) -> Int
```

that, given an array A consisting of N integers, returns index of any element of array A in which the dominator of A occurs. The function should return −1 if array A does not have a dominator.

For example, given array A such that

```
A[0] = 3    A[1] = 4    A[2] =  3
A[3] = 2    A[4] = 3    A[5] = -1
A[6] = 3    A[7] = 3
```

the function may return 0, 2, 4, 6 or 7, as explained above.

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

- N is an integer within the range [0..100,000];

Solution

Programming language used:	Swift 4
Total time used:	18 minutes
Effective time used:	18 minutes
Notes:	*not defined yet*

Task timeline

14:24:44 14:42:27

Code: 14:42:27 UTC, swift4, final, score: **100** show code in pop-up

```
1  import Foundation
2  import Glibc
3
4  // you can write to stdout for debugging purposes, e.g
5  // print("this is a debug message")
6
7  public struct Stack<T> {
```

- each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].


```
8
9     /// Datastructure consisting of a generic item.
10    fileprivate var array = [T]()
11
12    /// The number of items in the stack.
13    public var count: Int {
14      return array.count
15    }
16
17    /// Verifies if the stack is empty.
18    public var isEmpty: Bool {
19      return array.isEmpty
20    }
21
22    /**
23       Pushes an item to the top of the stack.
24
25       - Parameter element: The item being pushed.
26    */
27    public mutating func push(_ element: T) {
28      array.append(element)
29    }
30
31    /**
32       Removes and returns the item at the top of the sta
33
34       - Returns: The item at the top of the stack.
35    */
36    public mutating func pop() -> T? {
37      return array.popLast()
38    }
39
40    /// Returns the item at the top of the stack.
41    public var top: T? {
42      return array.last
43    }
44  }
45
46
47  public func solution(_ A : inout [Int]) -> Int {
48
49      var stack = Stack<Int>()
50      var index = -1
51      outer: for item in A {
52          if let top = stack.top {
53              if item != top {
54                  stack.pop()
55                  continue outer
56              }
57          }
58          stack.push(item)
59      }
60
61      guard let element = stack.top else { return index }
62
63      if (A.filter{$0 == element}.count) > A.count/2 {
64          for (i,item) in A.enumerated(){
65              if stack.top! == item {
66                  index = i
67                  break
68              }
69          }
70      }
71
72      return index
73  }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	**Example tests**	
▶	example example test	✔ OK
expand all	**Correctness tests**	
▶	small_nondominator all different and all the same elements	✔ OK
▶	small_half_positions half elements the same, and half + 1 elements the same	✔ OK
▶	small small test	✔ OK
▶	small_pyramid decreasing and plateau, small	✔ OK
▶	extreme_empty_and_single_item empty and single element arrays	✔ OK
▶	extreme_half1 array with exactly N/2 values 1, N even + [0,0,1,1,1]	✔ OK
▶	extreme_half2 array with exactly floor(N/2) values 1, N odd + [0,0,1,1,1]	✔ OK
▶	extreme_half3 array with exactly ceil(N/2) values 1 + [0,0,1,1,1]	✔ OK
expand all	**Performance tests**	
▶	medium_pyramid decreasing and plateau, medium	✔ OK
▶	large_pyramid decreasing and plateau, large	✔ OK
▶	medium_random random test with dominator, N = 10,000	✔ OK
▶	large_random random test with dominator, N = 100,000	✔ OK