Codility_

Candidate Report: Anonymous

Check out Codility training tasks

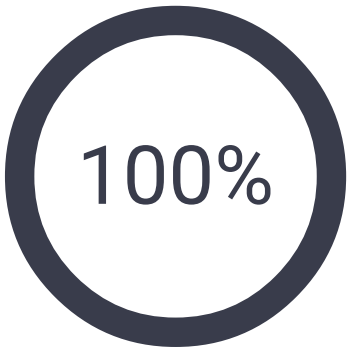
Test Name:

Summary Timeline Feedback

Tasks summary

Task	Time spent	Score
EquiLeader Swift 4	1 min	100%

Total score



Tasks Details

Easy

1. EquiLeader

Find the index S such that the leaders of the sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] are the same.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A non-empty array A consisting of N integers is given.

The *leader* of this array is the value that occurs in more than half of the elements of A.

An *equi leader* is an index S such that 0 ≤ S < N − 1 and two sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N − 1] have leaders of the same value.

For example, given array A such that:

```
A[0] = 4
A[1] = 3
A[2] = 4
A[3] = 4
A[4] = 4
A[5] = 2
```

we can find two equi leaders:

- 0, because sequences: (4) and (3, 4, 4, 4, 2) have the same leader, whose value is 4.
- 2, because sequences: (4, 3, 4) and (4, 4, 2) have the same leader, whose value is 4.

The goal is to count the number of equi leaders.

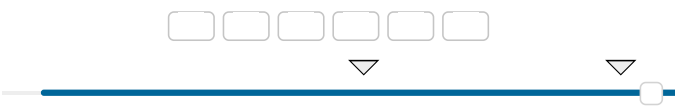
Write a function:

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

Solution

Programming language used:	Swift 4
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline



23:25:32 23:25:57

Code: 23:25:57 UTC, swift4, final, score: **100** show code in pop-up

```
1  import Foundation
2  import Glibc
3
4  public struct Stack<T> {
5
6    /// Datastructure consisting of a generic item.
7    fileprivate var array = [T]()
```

that, given a non-empty array A consisting of N integers, returns the number of equi leaders.

For example, given:

```
A[0] = 4
A[1] = 3
A[2] = 4
A[3] = 4
A[4] = 4
A[5] = 2
```

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 [−1,000,000,000..1,000,000,000].


```
8
9     /// The number of items in the stack.
10    public var count: Int {
11      return array.count
12    }
13
14    /// Verifies if the stack is empty.
15    public var isEmpty: Bool {
16      return array.isEmpty
17    }
18
19    /**
20       Pushes an item to the top of the stack.
21
22       - Parameter element: The item being pushed.
23    */
24    public mutating func push(_ element: T) {
25      array.append(element)
26    }
27
28    /**
29       Removes and returns the item at the top of the sta
30
31       - Returns: The item at the top of the stack.
32    */
33    public mutating func pop() -> T? {
34      return array.popLast()
35    }
36
37    /// Returns the item at the top of the stack.
38    public var top: T? {
39      return array.last
40    }
41  }
42
43
44
45  public func solution(_ A : inout [Int]) -> Int {
46
47      if A.count == 1 {return 0}
48      var dict = Dictionary<Int,Int>()
49      for item in A{
50          dict[item] = 1 + (dict[item] ?? 0)
51      }
52      if let max = dict.values.max(), !(max > A.count/2)
53          return 0
54      }
55
56  //    print(dict)
57  //    print()
58  //    print()
59
60      var counter = 0
61      var leftDict = Dictionary<Int,Int>()
62      var rightDict = dict
63      outer: for i in 0...A.count-1 {
64          let v = A[i]
65          leftDict[v] = 1 + (leftDict[v] ?? 0)
66          rightDict[v] = (rightDict[v] != nil && rightDi
67
68  //        print(leftDict, rightDict)
69
70          guard let maxValueInLeft = leftDict.values.max
71          let dominantInLeft = leftDict.first(where: {$0
72          let isThereLeftLeader = maxValueInLeft > leftD
73
74          guard let maxValueInRight = rightDict.values.ma
75          let dominantInRight = rightDict.first(where: {
76          let isThereRightLeader = maxValueInRight > rig
77
78  //        print(maxValueInLeft, maxValueInRight)
79  //        print(dominantInLeft, dominantInRight)
80  //        print(isThereLeftLeader, isThereRightLeader)
81  //        print("-")
82
83          if isThereLeftLeader && isThereRightLeader &&
84              counter += 1
85          }
86      }
87
88      return counter
89  }
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶ example example test		✔ OK
expand all	Correctness tests	
▶ single single element		✔ OK
▶ double two elements		✔ OK
▶ simple simple test		✔ OK
▶ small_random small random test with two values, length = ~100		✔ OK
▶ small random + 200 * [MIN_INT] + random ,length = ~300		✔ OK
collapse all	Performance tests	
▼ large_random large random test with two values, length = ~50,000		✔ OK
1. 0.056 s OK		
▼ large random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000		✔ OK
1. 0.076 s OK		
▼ large_range 1, 2, ..., N, length = ~100,000		✔ OK
1. 0.064 s OK 2. 0.028 s OK		
▼ extreme_large all the same values		✔ OK
1. 0.084 s OK		