Problem 2960: Count Tested Devices After Test Operations

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

0-indexed

integer array

batteryPercentages

having length

n

, denoting the battery percentages of

n

0-indexed

devices.

Your task is to test each device

i

in order

from

0

to

n - 1

, by performing the following test operations:

If

batteryPercentages[i]

is

greater

than

0

:

Increment

the count of tested devices.

Decrease

the battery percentage of all devices with indices

j

in the range

[i + 1, n - 1]

by

1

, ensuring their battery percentage

never goes below

0

, i.e,

batteryPercentages[j] = max(0, batteryPercentages[j] - 1)

.

Move to the next device.

Otherwise, move to the next device without performing any test.

Return

an integer denoting the number of devices that will be tested after performing the test operations in order.

Example 1:

Input:

batteryPercentages = [1,1,2,1,3]

Output:

3

Explanation:

Performing the test operations in order starting from device 0: At device 0, batteryPercentages[0] > 0, so there is now 1 tested device, and batteryPercentages becomes

[1,0,1,0,2]. At device 1, batteryPercentages[1] == 0, so we move to the next device without testing. At device 2, batteryPercentages[2] > 0, so there are now 2 tested devices, and batteryPercentages becomes [1,0,1,0,1]. At device 3, batteryPercentages[3] == 0, so we move to the next device without testing. At device 4, batteryPercentages[4] > 0, so there are now 3 tested devices, and batteryPercentages stays the same. So, the answer is 3.

Example 2:

Input:

batteryPercentages = [0,1,2]

Output:

2

Explanation:

Performing the test operations in order starting from device 0: At device 0, batteryPercentages[0] == 0, so we move to the next device without testing. At device 1, batteryPercentages[1] > 0, so there is now 1 tested device, and batteryPercentages becomes [0,1,1]. At device 2, batteryPercentages[2] > 0, so there are now 2 tested devices, and batteryPercentages stays the same. So, the answer is 2.

Constraints:

1 <= n == batteryPercentages.length <= 100

0 <= batteryPercentages[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int countTestedDevices(vector<int>& batteryPercentages) {

}
```

```
};
```

Java:

```
class Solution {
public int countTestedDevices(int[] batteryPercentages) {

}
}
```

Python3:

```
class Solution:
def countTestedDevices(self, batteryPercentages: List[int]) -> int:
```

Python:

```
class Solution(object):
def countTestedDevices(self, batteryPercentages):
"""
:type batteryPercentages: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} batteryPercentages
* @return {number}
*/
var countTestedDevices = function(batteryPercentages) {

};
```

TypeScript:

```
function countTestedDevices(batteryPercentages: number[]): number {

};
```

C#:

```
public class Solution {
public int CountTestedDevices(int[] batteryPercentages) {

}
}
```

C:

```
int countTestedDevices(int* batteryPercentages, int batteryPercentagesSize) {

}
```

Go:

```
func countTestedDevices(batteryPercentages []int) int {

}
```

Kotlin:

```
class Solution {
fun countTestedDevices(batteryPercentages: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countTestedDevices(_ batteryPercentages: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_tested_devices(battery_percentages: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} battery_percentages
# @return {Integer}
def count_tested_devices(battery_percentages)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $batteryPercentages
* @return Integer
*/
function countTestedDevices($batteryPercentages) {

}
}
```

Dart:

```
class Solution {
int countTestedDevices(List<int> batteryPercentages) {

}
}
```

Scala:

```
object Solution {
def countTestedDevices(batteryPercentages: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_tested_devices(battery_percentages :: [integer]) :: integer
def count_tested_devices(battery_percentages) do

end
end
```

Erlang:

```
-spec count_tested_devices(BatteryPercentages :: [integer()]) -> integer().
count_tested_devices(BatteryPercentages) ->
.
```

Racket:

```
(define/contract (count-tested-devices batteryPercentages)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int countTestedDevices(vector<int>& batteryPercentages) {

}
};
```

Java Solution:

```
/**
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countTestedDevices(int[] batteryPercentages) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Tested Devices After Test Operations
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countTestedDevices(self, batteryPercentages: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countTestedDevices(self, batteryPercentages):
"""
:type batteryPercentages: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
```

```
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} batteryPercentages
* @return {number}
*/
var countTestedDevices = function(batteryPercentages) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countTestedDevices(batteryPercentages: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public int CountTestedDevices(int[] batteryPercentages) {

}
}
```

C Solution:

```
/*
* Problem: Count Tested Devices After Test Operations
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countTestedDevices(int* batteryPercentages, int batteryPercentagesSize) {

}
```

Go Solution:

```
// Problem: Count Tested Devices After Test Operations
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countTestedDevices(batteryPercentages []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countTestedDevices(batteryPercentages: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countTestedDevices(_ batteryPercentages: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Tested Devices After Test Operations
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_tested_devices(battery_percentages: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} battery_percentages
# @return {Integer}
def count_tested_devices(battery_percentages)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $batteryPercentages
* @return Integer
*/
function countTestedDevices($batteryPercentages) {

}
}
```

Dart Solution:

```
class Solution {
int countTestedDevices(List<int> batteryPercentages) {

}
}
```

Scala Solution:

```
object Solution {
def countTestedDevices(batteryPercentages: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_tested_devices(battery_percentages :: [integer]) :: integer
def count_tested_devices(battery_percentages) do

end
end
```

Erlang Solution:

```
-spec count_tested_devices(BatteryPercentages :: [integer()]) -> integer().
count_tested_devices(BatteryPercentages) ->
.
```

Racket Solution:

```
(define/contract (count-tested-devices batteryPercentages)
(-> (listof exact-integer?) exact-integer?)
)
```