

Problem 3354: Make Array Elements Equal to Zero

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`nums`

.

Start by selecting a starting position

`curr`

such that

`nums[curr] == 0`

, and choose a movement

direction

of either left or right.

After that, you repeat the following process:

If

`curr`

is out of the range

$[0, n - 1]$

, this process ends.

If

`nums[curr] == 0`

, move in the current direction by

incrementing

`curr`

if you are moving right, or

decrementing

`curr`

if you are moving left.

Else if

`nums[curr] > 0`

:

Decrement

`nums[curr]`

by 1.

Reverse

your movement direction (left becomes right and vice versa).

Take a step in your new direction.

A selection of the initial position

curr

and movement direction is considered

valid

if every element in

nums

becomes 0 by the end of the process.

Return the number of possible

valid

selections.

Example 1:

Input:

nums = [1,0,2,0,3]

Output:

2

Explanation:

The only possible valid selections are the following:

Choose

curr = 3

, and a movement direction to the left.

[1,0,2,

0

,3] -> [1,0,

2

,0,3] -> [1,0,1,

0

,3] -> [1,0,1,0,

3

] -> [1,0,1,

0

,2] -> [1,0,

1

,0,2] -> [1,0,0,

0

,2] -> [1,0,0,0,

2

] -> [1,0,0,

0

,1] -> [1,0,

0

,0,1] -> [1,

0

,0,0,1] -> [

1

,0,0,0,1] -> [0,

0

,0,0,1] -> [0,0,

0

,0,1] -> [0,0,0,

0

,1] -> [0,0,0,0,

1

] -> [0,0,0,0,0]

.

Choose

curr = 3

, and a movement direction to the right.

[1,0,2,

0

,3] -> [1,0,2,0,

3

] -> [1,0,2,

0

,2] -> [1,0,

2

,0,2] -> [1,0,1,

0

,2] -> [1,0,1,0,

2

] -> [1,0,1,

0

,1] -> [1,0,

1

,0,1] -> [1,0,0,

0

,1] -> [1,0,0,0,

1

] -> [1,0,0,

0

,0] -> [1,0,

0

,0,0] -> [1,

0

,0,0,0] -> [

1

,0,0,0,0] -> [0,0,0,0,0].

Example 2:

Input:

nums = [2,3,4,0,4,1,0]

Output:

0

Explanation:

There are no possible valid selections.

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 100

There is at least one element

i

where

nums[i] == 0

.

Code Snippets

C++:

```
class Solution {
public:
    int countValidSelections(vector<int>& nums) {

    }
};
```

Java:

```
class Solution {
    public int countValidSelections(int[] nums) {

    }
}
```

Python3:

```
class Solution:
    def countValidSelections(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
    def countValidSelections(self, nums):
        """
        :type nums: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countValidSelections(nums: number[]): number {

};
```

C#:

```
public class Solution {
    public int CountValidSelections(int[] nums) {

    }
}
```

C:

```
int countValidSelections(int* nums, int numsSize) {

}
```

Go:

```
func countValidSelections(nums []int) int {

}
```

Kotlin:

```

class Solution {
    fun countValidSelections(nums: IntArray): Int {

    }
}

```

Swift:

```

class Solution {
    func countValidSelections(_ nums: [Int]) -> Int {

    }
}

```

Rust:

```

impl Solution {
    pub fn count_valid_selections(nums: Vec<i32>) -> i32 {

    }
}

```

Ruby:

```

# @param {Integer[]} nums
# @return {Integer}
def count_valid_selections(nums)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer
     */
    function countValidSelections($nums) {

    }
}

```

Dart:

```
class Solution {  
  int countValidSelections(List<int> nums) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def countValidSelections(nums: Array[Int]): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec count_valid_selections(nums :: [integer]) :: integer  
  def count_valid_selections(nums) do  
  
  end  
end
```

Erlang:

```
-spec count_valid_selections(Nums :: [integer()]) -> integer().  
count_valid_selections(Nums) ->  
.
```

Racket:

```
(define/contract (count-valid-selections nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Make Array Elements Equal to Zero
 * 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 countValidSelections(vector<int>& nums) {

    }
};

```

Java Solution:

```

/**
 * Problem: Make Array Elements Equal to Zero
 * 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 countValidSelections(int[] nums) {

    }
}

```

Python3 Solution:

```

"""
Problem: Make Array Elements Equal to Zero
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 countValidSelections(self, nums: List[int]) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def countValidSelections(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Make Array Elements Equal to Zero
 * 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[]} nums
 * @return {number}
 */
var countValidSelections = function(nums) {

};

```

TypeScript Solution:

```

/**
 * Problem: Make Array Elements Equal to Zero
 * 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 countValidSelections(nums: number[]): number {

};

```

C# Solution:

```

/*
 * Problem: Make Array Elements Equal to Zero
 * 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 CountValidSelections(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Make Array Elements Equal to Zero
 * 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 countValidSelections(int* nums, int numsSize) {

}

```

Go Solution:

```

// Problem: Make Array Elements Equal to Zero
// 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 countValidSelections(nums []int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun countValidSelections(nums: IntArray): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func countValidSelections(_ nums: [Int]) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Make Array Elements Equal to Zero
// 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_valid_selections(nums: Vec<i32>) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def count_valid_selections(nums)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer
     */
    function countValidSelections($nums) {

    }
}
```

Dart Solution:

```
class Solution {
    int countValidSelections(List<int> nums) {

    }
}
```

Scala Solution:

```
object Solution {  
  def countValidSelections(nums: Array[Int]): Int = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec count_valid_selections(nums :: [integer]) :: integer  
  def count_valid_selections(nums) do  
  
  end  
end
```

Erlang Solution:

```
-spec count_valid_selections(Nums :: [integer()]) -> integer().  
count_valid_selections(Nums) ->  
.
```

Racket Solution:

```
(define/contract (count-valid-selections nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```