Problem 2659: Make Array Empty

Problem Information

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

Problem Description

You are given an integer array

nums

containing

distinct

numbers, and you can perform the following operations

until the array is empty

:

If the first element has the

smallest

value, remove it

Otherwise, put the first element at the

end

of the array.

Return

an integer denoting the number of operations it takes to make

nums

empty.

Example 1:

Input:

nums = [3,4,-1]

Output:

5

Operation

Array

1

[4, -1, 3]

2

[-1, 3, 4]

3

[3, 4]

4

[4]

5

[]

Example 2:

Input:

nums = [1,2,4,3]

Output:

5

Operation

Array

1

[2, 4, 3]

2

[4, 3]

3

[3, 4]

4

[4]

5

[]

Example 3:

Input:

nums = [1,2,3]

Output:

3

Operation

Array

1

[2, 3]

2

[3]

3

[]

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

All values in

nums

are

distinct

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long countOperationsToEmptyArray(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long CountOperationsToEmptyArray(int[] nums) {

}
}
```

C:

```
long long countOperationsToEmptyArray(int* nums, int numsSize) {

}
```

Go:

```
func countOperationsToEmptyArray(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun countOperationsToEmptyArray(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_operations_to_empty_array(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

```
object Solution {
def countOperationsToEmptyArray(nums: Array[Int]): Long = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-operations-to-empty-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long countOperationsToEmptyArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long countOperationsToEmptyArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Array Empty
Difficulty: Hard
Tags: array, tree, greedy, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countOperationsToEmptyArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long CountOperationsToEmptyArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Make Array Empty
* Difficulty: Hard
* Tags: array, tree, greedy, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long countOperationsToEmptyArray(int* nums, int numsSize) {
```

```
}
```

Go Solution:

```
// Problem: Make Array Empty
// Difficulty: Hard
// Tags: array, tree, greedy, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countOperationsToEmptyArray(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countOperationsToEmptyArray(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Array Empty
// Difficulty: Hard
// Tags: array, tree, greedy, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_operations_to_empty_array(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def countOperationsToEmptyArray(nums: Array[Int]): Long = {
```

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-operations-to-empty-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```