Problem 2459: Sort Array by Moving Items to Empty Space

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

containing

each

element from

0

to

n - 1

(

inclusive

). Each of the elements from

1

to

n - 1

represents an item, and the element

0

represents an empty space.

In one operation, you can move

any

item to the empty space.

nums

is considered to be sorted if the numbers of all the items are in

ascending

order and the empty space is either at the beginning or at the end of the array.

For example, if

n = 4

,

nums

is sorted if:

nums = [0,1,2,3]

or

nums = [1,2,3,0]

...and considered to be unsorted otherwise.

Return

the

minimum

number of operations needed to sort

nums

.

Example 1:

Input:

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

Output:

3

Explanation:

- Move item 2 to the empty space. Now, nums = [4,0,2,3,1]. - Move item 1 to the empty space. Now, nums = [4,1,2,3,0]. - Move item 4 to the empty space. Now, nums = [0,1,2,3,4]. It can be proven that 3 is the minimum number of operations needed.

Example 2:

Input:

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

Output:

0

Explanation:

nums is already sorted so return 0.

Example 3:

Input:

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

Output:

2

Explanation:

- Move item 2 to the empty space. Now, nums = [1,2,0,4,3]. - Move item 3 to the empty space. Now, nums = [1,2,3,4,0]. It can be proven that 2 is the minimum number of operations needed.

Constraints:

n == nums.length

2 <= n <= 10

5

0 <= nums[i] < n

All the values of

nums

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun sortArray(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sort-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 sortArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 sortArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Array by Moving Items to Empty Space
Difficulty: Hard
Tags: array, greedy, sort

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 sortArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 sortArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 sortArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 SortArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort Array by Moving Items to Empty Space
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 sortArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sort Array by Moving Items to Empty Space
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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 sortArray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sortArray(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sort Array by Moving Items to Empty Space
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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 sort_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

```dart
class Solution {
int sortArray(List<int> nums) {

}
}
```

Scala Solution:

```scala
object Solution {
def sortArray(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sort-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```