

Problem 2340: Minimum Adjacent Swaps to Make a Valid Array

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

Swaps

of

adjacent

elements are able to be performed on

nums

A

valid

array meets the following conditions:

The largest element (any of the largest elements if there are multiple) is at the rightmost position in the array.

The smallest element (any of the smallest elements if there are multiple) is at the leftmost position in the array.

Return

the

minimum

swaps required to make

nums

a valid array

.

Example 1:

Input:

nums = [3,4,5,5,3,1]

Output:

6

Explanation:

Perform the following swaps: - Swap 1: Swap the 3

rd

and 4

th

elements, nums is then [3,4,5,

3

,

5

,1]. - Swap 2: Swap the 4

th

and 5

th

elements, nums is then [3,4,5,3,

1

,

5

]. - Swap 3: Swap the 3

rd

and 4

th

elements, nums is then [3,4,5,

1

,

3

,5]. - Swap 4: Swap the 2

nd

and 3

rd

elements, nums is then [3,4,

1

,

5

,3,5]. - Swap 5: Swap the 1

st

and 2

nd

elements, nums is then [3,

1

,

4

,5,3,5]. - Swap 6: Swap the 0

th

and 1

st

elements, nums is then [

1

,

3

,4,5,3,5]. It can be shown that 6 swaps is the minimum swaps required to make a valid array.

Example 2:

Input:

nums = [9]

Output:

0

Explanation:

The array is already valid, so we return 0.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 10$

5

Code Snippets

C++:

```
class Solution {
public:
    int minimumSwaps(vector<int>& nums) {
        ...
    }
};
```

Java:

```
class Solution {
    public int minimumSwaps(int[] nums) {
        ...
    }
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function minimumSwaps(nums: number[]): number {  
}  
};
```

C#:

```
public class Solution {  
    public int MinimumSwaps(int[] nums) {  
        }  
    }  
}
```

C:

```
int minimumSwaps(int* nums, int numsSize) {  
  
}
```

Go:

```
func minimumSwaps(nums []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun minimumSwaps(nums: IntArray): Int {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer
     */
    function minimumSwaps($nums) {
        }
    }
}
```

Dart:

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

Scala:

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

```
}
```

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (minimum-swaps nums)
  (-> (listof exact-integer?) exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimum Adjacent Swaps to Make a Valid Array
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumSwaps(vector<int>& nums) {
```

```
}
```

```
} ;
```

Java Solution:

```
/**  
 * Problem: Minimum Adjacent Swaps to Make a Valid Array  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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 minimumSwaps(int[] nums) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Minimum Adjacent Swaps to Make a Valid Array  
Difficulty: Medium  
Tags: array, greedy  
  
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 minimumSwaps(self, nums: List[int]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

```

JavaScript Solution:

```
/**
 * Problem: Minimum Adjacent Swaps to Make a Valid Array
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumSwaps = function(nums) {
}
```

TypeScript Solution:

```
/**
 * Problem: Minimum Adjacent Swaps to Make a Valid Array
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumSwaps(nums: number[]): number {
}
```

C# Solution:

```
/*
 * Problem: Minimum Adjacent Swaps to Make a Valid Array
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 MinimumSwaps(int[] nums) {
        return 0;
    }
}
```

C Solution:

```
/*
 * Problem: Minimum Adjacent Swaps to Make a Valid Array
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumSwaps(int* nums, int numsSize) {
    return 0;
}
```

Go Solution:

```
// Problem: Minimum Adjacent Swaps to Make a Valid Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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 minimumSwaps(nums []int) int {
}
```

Kotlin Solution:

```
class Solution {
    fun minimumSwaps(nums: IntArray): Int {
        return 0
    }
}
```

Swift Solution:

```
class Solution {
    func minimumSwaps(_ nums: [Int]) -> Int {
        return 0
    }
}
```

Rust Solution:

```
// Problem: Minimum Adjacent Swaps to Make a Valid Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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 minimum_swaps(nums: Vec<i32>) -> i32 {
        return 0
    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

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

}
```

Scala Solution:

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

}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-swaps nums)  
(-> (listof exact-integer?) exact-integer?)  
)
```