Problem 3587: Minimum Adjacent Swaps to Alternate Parity

Problem Information

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

Problem Description

You are given an array

nums

of

distinct

integers.

In one operation, you can swap any two

adjacent

elements in the array.

An arrangement of the array is considered

valid

if the parity of adjacent elements

alternates

, meaning every pair of neighboring elements consists of one even and one odd number.

Return the

minimum

number of adjacent swaps required to transform

nums

into any valid arrangement.

If it is impossible to rearrange

nums

such that no two adjacent elements have the same parity, return

-1

.

Example 1:

Input:

nums = [2,4,6,5,7]

Output:

3

Explanation:

Swapping 5 and 6, the array becomes

[2,4,5,6,7]

Swapping 5 and 4, the array becomes

[2,5,4,6,7]

Swapping 6 and 7, the array becomes

[2,5,4,7,6]

. The array is now a valid arrangement. Thus, the answer is 3.

Example 2:

Input:

nums = [2,4,5,7]

Output:

1

Explanation:

By swapping 4 and 5, the array becomes

[2,5,4,7]

, which is a valid arrangement. Thus, the answer is 1.

Example 3:

Input:

nums = [1,2,3]

Output:

0

Explanation:

The array is already a valid arrangement. Thus, no operations are needed.

Example 4:

Input:

nums = [4,5,6,8]

Output:

-1

Explanation:

No valid arrangement is possible. Thus, the answer is -1.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

All elements in

nums

are

distinct

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 minSwaps(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 minSwaps(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Adjacent Swaps to Alternate Parity
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 minSwaps(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 minSwaps = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 minSwaps(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 MinSwaps(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Adjacent Swaps to Alternate Parity
* 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 minSwaps(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Adjacent Swaps to Alternate Parity
// 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 minSwaps(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Adjacent Swaps to Alternate Parity
// 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 min_swaps(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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