Problem 765: Couples Holding Hands

Problem Information

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

Problem Description

There are

n

couples sitting in

2n

seats arranged in a row and want to hold hands.

The people and seats are represented by an integer array

row

where

row[i]

is the ID of the person sitting in the

i

th

seat. The couples are numbered in order, the first couple being

(0, 1)

, the second couple being

(2, 3)

, and so on with the last couple being

(2n - 2, 2n - 1)

.

Return

the minimum number of swaps so that every couple is sitting side by side

. A swap consists of choosing any two people, then they stand up and switch seats.

Example 1:

Input:

row = [0,2,1,3]

Output:

1

Explanation:

We only need to swap the second (row[1]) and third (row[2]) person.

Example 2:

Input:

row = [3,2,0,1]

Output:

0

Explanation:

All couples are already seated side by side.

Constraints:

2n == row.length

2 <= n <= 30

0 <= row[i] < 2n

All the elements of

row

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int minSwapsCouples(vector<int>& row) {

}
};
```

Java:

```
class Solution {
public int minSwapsCouples(int[] row) {

}
}
```

Python3:

```
class Solution:
def minSwapsCouples(self, row: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minSwapsCouples(row: number[]): number {

};
```

C#:

```
public class Solution {
public int MinSwapsCouples(int[] row) {

}
}
```

C:

```
int minSwapsCouples(int* row, int rowSize) {

}
```

Go:

```
func minSwapsCouples(row []int) int {

}
```

Kotlin:

```
class Solution {
fun minSwapsCouples(row: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minSwapsCouples(_ row: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_swaps_couples(row: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} row
# @return {Integer}
def min_swaps_couples(row)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $row
* @return Integer
*/
function minSwapsCouples($row) {

}
}
```

Dart:

```
class Solution {
int minSwapsCouples(List<int> row) {

}
}
```

Scala:

```
object Solution {
def minSwapsCouples(row: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_swaps_couples(row :: [integer]) :: integer
def min_swaps_couples(row) do

end
end
```

Erlang:

```
-spec min_swaps_couples(Row :: [integer()]) -> integer().
min_swaps_couples(Row) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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 minSwapsCouples(vector<int>& row) {

}
};
```

Java Solution:

```
/**
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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 minSwapsCouples(int[] row) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Couples Holding Hands
Difficulty: Hard
Tags: array, graph, greedy, search

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 minSwapsCouples(self, row: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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[]} row
* @return {number}
*/
var minSwapsCouples = function(row) {

};
```

TypeScript Solution:

```
/**
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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 minSwapsCouples(row: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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 MinSwapsCouples(int[] row) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Couples Holding Hands
* Difficulty: Hard
* Tags: array, graph, greedy, search
*
* 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 minSwapsCouples(int* row, int rowSize) {

}
```

Go Solution:

```
// Problem: Couples Holding Hands
// Difficulty: Hard
// Tags: array, graph, greedy, search
//
// 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 minSwapsCouples(row []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSwapsCouples(row: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSwapsCouples(_ row: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Couples Holding Hands
// Difficulty: Hard
// Tags: array, graph, greedy, search
//
// 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_couples(row: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} row
# @return {Integer}
def min_swaps_couples(row)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $row
* @return Integer
*/
function minSwapsCouples($row) {

}
}
```

Dart Solution:

```
class Solution {
int minSwapsCouples(List<int> row) {

}
}
```

Scala Solution:

```
object Solution {
def minSwapsCouples(row: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_swaps_couples(row :: [integer]) :: integer
def min_swaps_couples(row) do

end
end
```

Erlang Solution:

```
-spec min_swaps_couples(Row :: [integer()]) -> integer().
min_swaps_couples(Row) ->
.
```

Racket Solution:

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