Problem 1536: Minimum Swaps to Arrange a Binary Grid

Problem Information

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

Problem Description

Given an

n x n

binary

grid

, in one step you can choose two

adjacent rows

of the grid and swap them.

A grid is said to be

valid

if all the cells above the main diagonal are

zeros

.

Return

the minimum number of steps

needed to make the grid valid, or

-1

if the grid cannot be valid.

The main diagonal of a grid is the diagonal that starts at cell

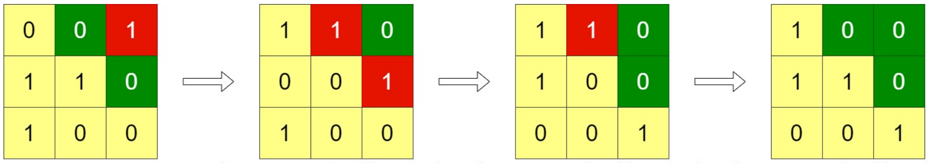
(1, 1)

and ends at cell

(n, n)

.

Example 1:



Swap row 1 with row 2 Swap row 2 with row 3 Swap row 1 with row 2

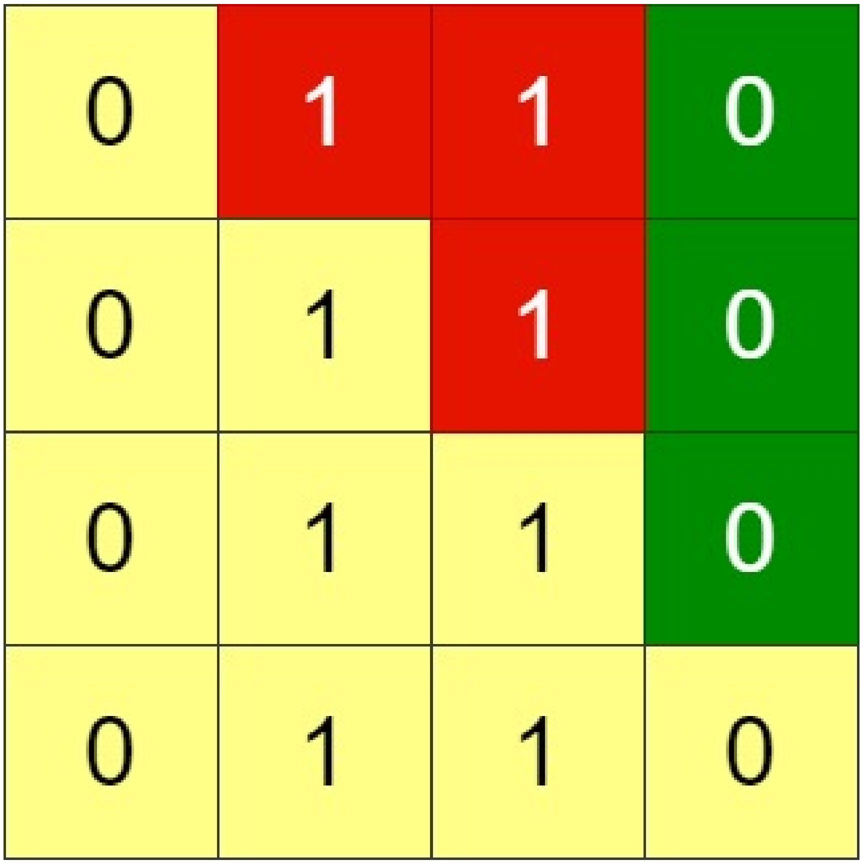
Input:

grid = [[0,0,1],[1,1,0],[1,0,0]]

Output:

3

Example 2:



0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

Input:

grid = [[0,1,1,0],[0,1,1,0],[0,1,1,0],[0,1,1,0]]

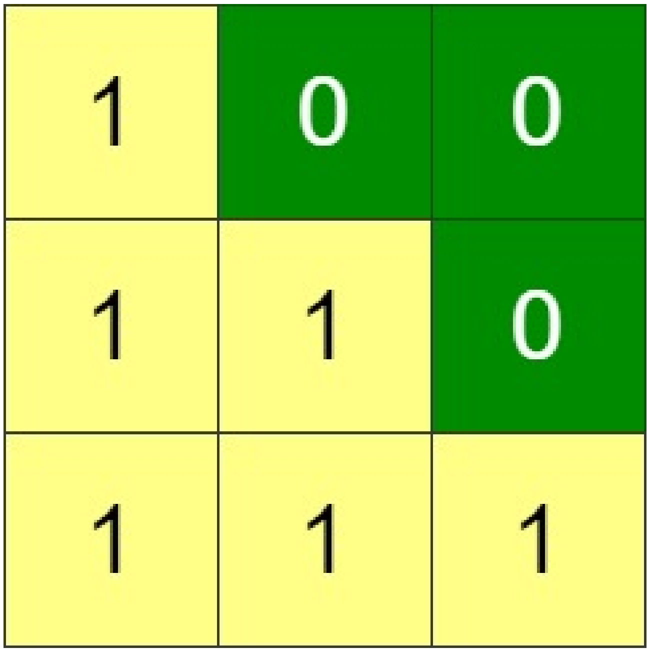
Output:

-1

Explanation:

All rows are similar, swaps have no effect on the grid.

Example 3:



1	0	0
1	1	0
1	1	1

Input:

grid = [[1,0,0],[1,1,0],[1,1,1]]

Output:

0

Constraints:

n == grid.length

== grid[i].length

1 <= n <= 200

grid[i][j]

is either

0

or

1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
```

```
var minSwaps = function(grid) {

};
```

TypeScript:

```
function minSwaps(grid: number[][]): number {

};
```

C#:

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

}
}
```

C:

```
int minSwaps(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minSwaps(grid: Array<IntArray>): Int {

}
}
```

Swift:

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



}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Swaps to Arrange a Binary Grid
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, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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[][]} grid
* @return {number}
*/
var minSwaps = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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(grid: number[][]): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Swaps to Arrange a Binary Grid
* 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** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Swaps to Arrange a Binary Grid
// 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(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSwaps(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Swaps to Arrange a Binary Grid
// 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(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

```dart
class Solution {
int minSwaps(List<List<int>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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