Problem 2938: Separate Black and White Balls

Problem Information

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

Problem Description

There are

n

balls on a table, each ball has a color black or white.

You are given a

0-indexed

binary string

s

of length

n

, where

1

and

0

represent black and white balls, respectively.

In each step, you can choose two adjacent balls and swap them.

Return

the

minimum

number of steps to group all the black balls to the right and all the white balls to the left

.

Example 1:

Input:

s = "101"

Output:

1

Explanation:

We can group all the black balls to the right in the following way: - Swap s[0] and s[1], s = "011". Initially, 1s are not grouped together, requiring at least 1 step to group them to the right.

Example 2:

Input:

s = "100"

Output:

2

Explanation:

We can group all the black balls to the right in the following way: - Swap s[0] and s[1], s = "010". - Swap s[1] and s[2], s = "001". It can be proven that the minimum number of steps needed is 2.

Example 3:

Input:

s = "0111"

Output:

0

Explanation:

All the black balls are already grouped to the right.

Constraints:

1 <= n == s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

```
class Solution {
public:
long long minimumSteps(string s) {

}
};
```

Java:

```
class Solution {
public long minimumSteps(String s) {

}
}
```

Python3:

```
class Solution:
def minimumSteps(self, s: str) -> int:
```

Python:

```
class Solution(object):
def minimumSteps(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var minimumSteps = function(s) {

};
```

TypeScript:

```
function minimumSteps(s: string): number {

};
```

C#:

```
public class Solution {
public long MinimumSteps(string s) {

}
}
```

C:

```
long long minimumSteps(char* s) {

}
```

Go:

```
func minimumSteps(s string) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumSteps(s: String): Long {

}
}
```

Swift:

```
class Solution {
func minimumSteps(_ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_steps(s: String) -> i64 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def minimum_steps(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function minimumSteps($s) {

}
}
```

Dart:

```
class Solution {
int minimumSteps(String s) {

}
}
```

Scala:

```
object Solution {
def minimumSteps(s: String): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_steps(s :: String.t) :: integer
def minimum_steps(s) do

end
end
```

Erlang:

```
-spec minimum_steps(S :: unicode:unicode_binary()) -> integer().
minimum_steps(S) ->
.
```

Racket:

```
(define/contract (minimum-steps s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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:
long long minimumSteps(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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 long minimumSteps(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Separate Black and White Balls
Difficulty: Medium
Tags: array, string, 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 minimumSteps(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumSteps(self, s):
"""
:type s: str
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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 {string} s
* @return {number}
*/
var minimumSteps = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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 minimumSteps(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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 long MinimumSteps(string s) {

}
}
```

C Solution:

```
/*
* Problem: Separate Black and White Balls
* Difficulty: Medium
* Tags: array, string, 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
*/

long long minimumSteps(char* s) {

}
```

Go Solution:

```
// Problem: Separate Black and White Balls
// Difficulty: Medium
// Tags: array, string, 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 minimumSteps(s string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumSteps(s: String): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumSteps(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Separate Black and White Balls
// Difficulty: Medium
// Tags: array, string, 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_steps(s: String) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def minimum_steps(s)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function minimumSteps($s) {

}
}
```

Dart Solution:

```
class Solution {
int minimumSteps(String s) {

}
}
```

Scala Solution:

```
object Solution {
def minimumSteps(s: String): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_steps(s :: String.t) :: integer
def minimum_steps(s) do

end
end
```

Erlang Solution:

```
-spec minimum_steps(S :: unicode:unicode_binary()) -> integer().
minimum_steps(S) ->
.
```

Racket Solution:

```
(define/contract (minimum-steps s)
(-> string? exact-integer?)
)
```