Problem 3616: Number of Student Replacements

Problem Information

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

Problem Description

You are given an integer array

ranks

where

ranks[i]

represents the rank of the

i

th

student arriving

in order

. A lower number indicates a

better

rank.

Initially, the first student is

selected

by default.

A

replacement

occurs when a student with a

strictly

better rank arrives and

replaces

the current selection.

Return the total number of replacements made.

Example 1:

Input:

ranks = [4,1,2]

Output:

1

Explanation:

The first student with

ranks[0] = 4

is initially selected.

The second student with

ranks[1] = 1

is better than the current selection, so a replacement occurs.

The third student has a worse rank, so no replacement occurs.

Thus, the number of replacements is 1.

Example 2:

Input:

ranks = [2,2,3]

Output:

0

Explanation:

The first student with

ranks[0] = 2

is initially selected.

Neither of

ranks[1] = 2

or

ranks[2] = 3

is better than the current selection.

Thus, the number of replacements is 0.

Constraints:

1 <= ranks.length <= 10

5

1 <= ranks[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int totalReplacements(vector<int>& ranks) {

}
};
```

Java:

```
class Solution {
public int totalReplacements(int[] ranks) {

}
}
```

Python3:

```
class Solution:
def totalReplacements(self, ranks: List[int]) -> int:
```

Python:

```
class Solution(object):
def totalReplacements(self, ranks):
"""
```

```
:type ranks: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function totalReplacements(ranks: number[]): number {

};
```

C#:

```
public class Solution {
public int TotalReplacements(int[] ranks) {

}
}
```

C:

```
int totalReplacements(int* ranks, int ranksSize) {

}
```

Go:

```
func totalReplacements(ranks []int) int {

}
```

Kotlin:

```
class Solution {
fun totalReplacements(ranks: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func totalReplacements(_ ranks: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn total_replacements(ranks: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} ranks
# @return {Integer}
def total_replacements(ranks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $ranks
* @return Integer
*/
function totalReplacements($ranks) {

}
}
```

Dart:

```
class Solution {
int totalReplacements(List<int> ranks) {

}
}
```

Scala:

```
object Solution {
def totalReplacements(ranks: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec total_replacements(ranks :: [integer]) :: integer
def total_replacements(ranks) do

end
end
```

Erlang:

```
-spec total_replacements(Ranks :: [integer()]) -> integer().
total_replacements(Ranks) ->
.
```

Racket:

```
(define/contract (total-replacements ranks)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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 totalReplacements(vector<int>& ranks) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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 totalReplacements(int[] ranks) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Student Replacements
Difficulty: Medium
Tags: array
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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[]} ranks
* @return {number}
*/
var totalReplacements = function(ranks) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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 totalReplacements(ranks: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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 TotalReplacements(int[] ranks) {

}
}
```

C Solution:

```
/*
* Problem: Number of Student Replacements
* Difficulty: Medium
* Tags: array
*
* 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 totalReplacements(int* ranks, int ranksSize) {

}
```

Go Solution:

```
// Problem: Number of Student Replacements
// Difficulty: Medium
// Tags: array
//
// 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 totalReplacements(ranks []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun totalReplacements(ranks: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func totalReplacements(_ ranks: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Student Replacements
// Difficulty: Medium
// Tags: array
```

```
//
// 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 total_replacements(ranks: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} ranks
# @return {Integer}
def total_replacements(ranks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $ranks
* @return Integer
*/
function totalReplacements($ranks) {

}
}
```

Dart Solution:

```
class Solution {
int totalReplacements(List<int> ranks) {

}
}
```

Scala Solution:

```
object Solution {
def totalReplacements(ranks: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec total_replacements(ranks :: [integer]) :: integer
def total_replacements(ranks) do

end
end
```

Erlang Solution:

```
-spec total_replacements(Ranks :: [integer()]) -> integer().
total_replacements(Ranks) ->
.
```

Racket Solution:

```
(define/contract (total-replacements ranks)
(-> (listof exact-integer?) exact-integer?)
)
```