Problem 2037: Minimum Number of Moves to Seat Everyone

Problem Information

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

Problem Description

There are

n

availabe

seats and

n

students

standing

in a room. You are given an array

seats

of length

n

, where

seats[i]

is the position of the

i

th

seat. You are also given the array

students

of length

n

, where

students[j]

is the position of the

j

th

student.

You may perform the following move any number of times:

Increase or decrease the position of the

i

th

student by

1

(i.e., moving the

i

th

student from position

x

to

x + 1

or

x - 1

)

Return

the

minimum number of moves

required to move each student to a seat

such that no two students are in the same seat.

Note that there may be

multiple

seats or students in the

same

position at the beginning.

Example 1:

Input:

seats = [3,1,5], students = [2,7,4]

Output:

4

Explanation:

The students are moved as follows: - The first student is moved from position 2 to position 1 using 1 move. - The second student is moved from position 7 to position 5 using 2 moves. - The third student is moved from position 4 to position 3 using 1 move. In total, 1 + 2 + 1 = 4 moves were used.

Example 2:

Input:

seats = [4,1,5,9], students = [1,3,2,6]

Output:

7

Explanation:

The students are moved as follows: - The first student is not moved. - The second student is moved from position 3 to position 4 using 1 move. - The third student is moved from position 2 to position 5 using 3 moves. - The fourth student is moved from position 6 to position 9 using 3 moves. In total, 0 + 1 + 3 + 3 = 7 moves were used.

Example 3:

Input:

seats = [2,2,6,6], students = [1,3,2,6]

Output:

4

Explanation:

Note that there are two seats at position 2 and two seats at position 6. The students are moved as follows: - The first student is moved from position 1 to position 2 using 1 move. - The second student is moved from position 3 to position 6 using 3 moves. - The third student is not moved. - The fourth student is not moved. In total, 1 + 3 + 0 + 0 = 4 moves were used.

Constraints:

n == seats.length == students.length

1 <= n <= 100

1 <= seats[i], students[j] <= 100

Code Snippets

C++:

```
class Solution {
public:
int minMovesToSeat(vector<int>& seats, vector<int>& students) {

}
};
```

Java:

```
class Solution {
public int minMovesToSeat(int[] seats, int[] students) {

}
}
```

Python3:

```
class Solution:
def minMovesToSeat(self, seats: List[int], students: List[int]) -> int:
```

Python:

```
class Solution(object):
def minMovesToSeat(self, seats, students):
"""
:type seats: List[int]
:type students: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} seats
* @param {number[]} students
* @return {number}
*/
var minMovesToSeat = function(seats, students) {

};
```

TypeScript:

```
function minMovesToSeat(seats: number[], students: number[]): number {

};
```

C#:

```
public class Solution {
public int MinMovesToSeat(int[] seats, int[] students) {

}
}
```

C:

```
int minMovesToSeat(int* seats, int seatsSize, int* students, int
studentsSize) {
```

```
}
```

Go:

```
func minMovesToSeat(seats []int, students []int) int {

}
```

Kotlin:

```
class Solution {
fun minMovesToSeat(seats: IntArray, students: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minMovesToSeat(_ seats: [Int], _ students: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_moves_to_seat(seats: Vec<i32>, students: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} seats
# @param {Integer[]} students
# @return {Integer}
def min_moves_to_seat(seats, students)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $seats
* @param Integer[] $students
* @return Integer
*/
function minMovesToSeat($seats, $students) {

}
}
```

Dart:

```
class Solution {
int minMovesToSeat(List<int> seats, List<int> students) {

}
}
```

Scala:

```
object Solution {
def minMovesToSeat(seats: Array[Int], students: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_moves_to_seat(seats :: [integer], students :: [integer]) :: integer
def min_moves_to_seat(seats, students) do

end
end
```

Erlang:

```
-spec min_moves_to_seat(Seats :: [integer()], Students :: [integer()]) ->
integer().
min_moves_to_seat(Seats, Students) ->
.
```

Racket:

```
(define/contract (min-moves-to-seat seats students)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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 minMovesToSeat(vector<int>& seats, vector<int>& students) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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 minMovesToSeat(int[] seats, int[] students) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Moves to Seat Everyone
Difficulty: Easy
Tags: array, greedy, sort

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 minMovesToSeat(self, seats: List[int], students: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minMovesToSeat(self, seats, students):
"""
:type seats: List[int]
:type students: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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[]} seats
* @param {number[]} students
* @return {number}
*/
var minMovesToSeat = function(seats, students) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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 minMovesToSeat(seats: number[], students: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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 MinMovesToSeat(int[] seats, int[] students) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Moves to Seat Everyone
* Difficulty: Easy
* Tags: array, greedy, sort
*
* 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 minMovesToSeat(int* seats, int seatsSize, int* students, int
studentsSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Moves to Seat Everyone
// Difficulty: Easy
// Tags: array, greedy, sort
//
// 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 minMovesToSeat(seats []int, students []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMovesToSeat(seats: IntArray, students: IntArray): Int {

}
```

```
}
```

Swift Solution:

```
class Solution {
func minMovesToSeat(_ seats: [Int], _ students: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Moves to Seat Everyone
// Difficulty: Easy
// Tags: array, greedy, sort
//
// 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_moves_to_seat(seats: Vec<i32>, students: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} seats
# @param {Integer[]} students
# @return {Integer}
def min_moves_to_seat(seats, students)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $seats
```

```
* @param Integer[] $students
* @return Integer
*/
function minMovesToSeat($seats, $students) {

}
}
```

Dart Solution:

```
class Solution {
int minMovesToSeat(List<int> seats, List<int> students) {

}
}
```

Scala Solution:

```
object Solution {
def minMovesToSeat(seats: Array[Int], students: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_moves_to_seat(seats :: [integer], students :: [integer]) :: integer
def min_moves_to_seat(seats, students) do

end
end
```

Erlang Solution:

```
-spec min_moves_to_seat(Seats :: [integer()], Students :: [integer()]) ->
integer().
min_moves_to_seat(Seats, Students) ->
.
```

Racket Solution:

```
(define/contract (min-moves-to-seat seats students)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```