Problem 2274: Maximum Consecutive Floors Without Special Floors

Problem Information

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

Problem Description

Alice manages a company and has rented some floors of a building as office space. Alice has decided some of these floors should be

special floors

, used for relaxation only.

You are given two integers

bottom

and

top

, which denote that Alice has rented all the floors from

bottom

to

top

(

inclusive

). You are also given the integer array

special

, where

special[i]

denotes a special floor that Alice has designated for relaxation.

Return

the

maximum

number of consecutive floors without a special floor

.

Example 1:

Input:

bottom = 2, top = 9, special = [4,6]

Output:

3

Explanation:

The following are the ranges (inclusive) of consecutive floors without a special floor: - (2, 3) with a total amount of 2 floors. - (5, 5) with a total amount of 1 floor. - (7, 9) with a total amount of 3 floors. Therefore, we return the maximum number which is 3 floors.

Example 2:

Input:

bottom = 6, top = 8, special = [7,6,8]

Output:

0

Explanation:

Every floor rented is a special floor, so we return 0.

Constraints:

1 <= special.length <= 10

5

1 <= bottom <= special[i] <= top <= 10

9

All the values of

special

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
```

```
int maxConsecutive(int bottom, int top, vector<int>& special) {

}
};
```

Java:

```
class Solution {
public int maxConsecutive(int bottom, int top, int[] special) {

}
}
```

Python3:

```
class Solution:
def maxConsecutive(self, bottom: int, top: int, special: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxConsecutive(self, bottom, top, special):
"""
:type bottom: int
:type top: int
:type special: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} bottom
* @param {number} top
* @param {number[]} special
* @return {number}
*/
var maxConsecutive = function(bottom, top, special) {

};
```

TypeScript:

```
function maxConsecutive(bottom: number, top: number, special: number[]):
number {

};
```

C#:

```
public class Solution {
public int MaxConsecutive(int bottom, int top, int[] special) {

}
}
```

C:

```
int maxConsecutive(int bottom, int top, int* special, int specialSize) {

}
```

Go:

```
func maxConsecutive(bottom int, top int, special []int) int {

}
```

Kotlin:

```
class Solution {
fun maxConsecutive(bottom: Int, top: Int, special: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxConsecutive(_ bottom: Int, _ top: Int, _ special: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_consecutive(bottom: i32, top: i32, special: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} bottom
# @param {Integer} top
# @param {Integer[]} special
# @return {Integer}
def max_consecutive(bottom, top, special)

end
```

PHP:

```
class Solution {

/**
* @param Integer $bottom
* @param Integer $top
* @param Integer[] $special
* @return Integer
*/
function maxConsecutive($bottom, $top, $special) {

}
}
```

Dart:

```
class Solution {
int maxConsecutive(int bottom, int top, List<int> special) {

}
}
```

Scala:

```
object Solution {
def maxConsecutive(bottom: Int, top: Int, special: Array[Int]): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec max_consecutive(bottom :: integer, top :: integer, special ::
[integer]) :: integer
def max_consecutive(bottom, top, special) do

end
end
```

Erlang:

```
-spec max_consecutive(Bottom :: integer(), Top :: integer(), Special ::
[integer()]) -> integer().
max_consecutive(Bottom, Top, Special) ->
.
```

Racket:

```
(define/contract (max-consecutive bottom top special)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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 maxConsecutive(int bottom, int top, vector<int>& special) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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 maxConsecutive(int bottom, int top, int[] special) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Consecutive Floors Without Special Floors
Difficulty: Medium
Tags: array, 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 maxConsecutive(self, bottom: int, top: int, special: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def maxConsecutive(self, bottom, top, special):
"""
:type bottom: int
:type top: int
:type special: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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} bottom
* @param {number} top
* @param {number[]} special
* @return {number}
*/
var maxConsecutive = function(bottom, top, special) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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 maxConsecutive(bottom: number, top: number, special: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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 MaxConsecutive(int bottom, int top, int[] special) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Consecutive Floors Without Special Floors
* Difficulty: Medium
* Tags: array, 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 maxConsecutive(int bottom, int top, int* special, int specialSize) {

}
```

Go Solution:

```
// Problem: Maximum Consecutive Floors Without Special Floors
// Difficulty: Medium
// Tags: array, 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 maxConsecutive(bottom int, top int, special []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxConsecutive(bottom: Int, top: Int, special: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxConsecutive(_ bottom: Int, _ top: Int, _ special: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Consecutive Floors Without Special Floors
// Difficulty: Medium
// Tags: array, 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 max_consecutive(bottom: i32, top: i32, special: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} bottom
# @param {Integer} top
# @param {Integer[]} special
# @return {Integer}
def max_consecutive(bottom, top, special)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $bottom
* @param Integer $top
* @param Integer[] $special
* @return Integer
*/
function maxConsecutive($bottom, $top, $special) {

}
}
```

Dart Solution:

```
class Solution {
int maxConsecutive(int bottom, int top, List<int> special) {

}
}
```

Scala Solution:

```
object Solution {
def maxConsecutive(bottom: Int, top: Int, special: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_consecutive(bottom :: integer, top :: integer, special ::
[integer]) :: integer
def max_consecutive(bottom, top, special) do

end
end
```

Erlang Solution:

```
-spec max_consecutive(Bottom :: integer(), Top :: integer(), Special ::
[integer()]) -> integer().
max_consecutive(Bottom, Top, Special) ->
.
```

Racket Solution:

```
(define/contract (max-consecutive bottom top special)
(-> exact-integer? exact-integer? (listof exact-integer?) exact-integer?)
)
```