Problem 1052: Grumpy Bookstore Owner

Problem Information

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

Problem Description

There is a bookstore owner that has a store open for

n

minutes. You are given an integer array

customers

of length

n

where

customers[i]

is the number of the customers that enter the store at the start of the

i

th

minute and all those customers leave after the end of that minute.

During certain minutes, the bookstore owner is grumpy. You are given a binary array grumpy where

grumpy[i]

is

1

if the bookstore owner is grumpy during the

i

th

minute, and is

0

otherwise.

When the bookstore owner is grumpy, the customers entering during that minute are not

satisfied

. Otherwise, they are satisfied.

The bookstore owner knows a secret technique to remain

not grumpy

for

minutes

consecutive minutes, but this technique can only be used

once

.

Return the

maximum

number of customers that can be

satisfied

throughout the day.

Example 1:

Input:

customers = [1,0,1,2,1,1,7,5], grumpy = [0,1,0,1,0,1,0,1], minutes = 3

Output:

16

Explanation:

The bookstore owner keeps themselves not grumpy for the last 3 minutes.

The maximum number of customers that can be satisfied = 1 + 1 + 1 + 1 + 7 + 5 = 16.

Example 2:

Input:

customers = [1], grumpy = [0], minutes = 1

Output:

1

Constraints:

n == customers.length == grumpy.length

1 <= minutes <= n <= 2 * 10

4

0 <= customers[i] <= 1000

grumpy[i]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int maxSatisfied(vector<int>& customers, vector<int>& grumpy, int minutes) {

}
};
```

Java:

```
class Solution {
public int maxSatisfied(int[] customers, int[] grumpy, int minutes) {

}
}
```

Python3:

```
class Solution:
def maxSatisfied(self, customers: List[int], grumpy: List[int], minutes: int)
-> int:
```

Python:

```
class Solution(object):
def maxSatisfied(self, customers, grumpy, minutes):
"""
:type customers: List[int]
:type grumpy: List[int]
:type minutes: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} customers
* @param {number[]} grumpy
* @param {number} minutes
* @return {number}
*/
var maxSatisfied = function(customers, grumpy, minutes) {

};
```

TypeScript:

```
function maxSatisfied(customers: number[], grumpy: number[], minutes:
number): number {

};
```

C#:

```
public class Solution {
public int MaxSatisfied(int[] customers, int[] grumpy, int minutes) {

}
}
```

C:

```
int maxSatisfied(int* customers, int customersSize, int* grumpy, int
grumpySize, int minutes) {

}
```

Go:

```
func maxSatisfied(customers []int, grumpy []int, minutes int) int {

}
```

Kotlin:

```
class Solution {
fun maxSatisfied(customers: IntArray, grumpy: IntArray, minutes: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxSatisfied(_ customers: [Int], _ grumpy: [Int], _ minutes: Int) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn max_satisfied(customers: Vec<i32>, grumpy: Vec<i32>, minutes: i32) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer[]} customers
# @param {Integer[]} grumpy
# @param {Integer} minutes
# @return {Integer}
```

```
def max_satisfied(customers, grumpy, minutes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $customers
* @param Integer[] $grumpy
* @param Integer $minutes
* @return Integer
*/
function maxSatisfied($customers, $grumpy, $minutes) {

}
}
```

Dart:

```
class Solution {
int maxSatisfied(List<int> customers, List<int> grumpy, int minutes) {

}
}
```

Scala:

```
object Solution {
def maxSatisfied(customers: Array[Int], grumpy: Array[Int], minutes: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_satisfied(customers :: [integer], grumpy :: [integer], minutes ::
integer) :: integer
def max_satisfied(customers, grumpy, minutes) do
```

```
end
end
```

Erlang:

```
-spec max_satisfied(Customers :: [integer()], Grumpy :: [integer()], Minutes
:: integer()) -> integer().
max_satisfied(Customers, Grumpy, Minutes) ->
.
```

Racket:

```
(define/contract (max-satisfied customers grumpy minutes)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Grumpy Bookstore Owner
* 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 maxSatisfied(vector<int>& customers, vector<int>& grumpy, int minutes) {

}
};
```

Java Solution:

```
/**
* Problem: Grumpy Bookstore Owner
* 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 maxSatisfied(int[] customers, int[] grumpy, int minutes) {

}
}
```

Python3 Solution:

```
"""
Problem: Grumpy Bookstore Owner
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 maxSatisfied(self, customers: List[int], grumpy: List[int], minutes: int)
-> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSatisfied(self, customers, grumpy, minutes):
"""
:type customers: List[int]
:type grumpy: List[int]
:type minutes: int
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Grumpy Bookstore Owner
* 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[]} customers
* @param {number[]} grumpy
* @param {number} minutes
* @return {number}
*/
var maxSatisfied = function(customers, grumpy, minutes) {

};
```

TypeScript Solution:

```
/**
* Problem: Grumpy Bookstore Owner
* 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 maxSatisfied(customers: number[], grumpy: number[], minutes:
number): number {

};
```

C# Solution:

```
/*
* Problem: Grumpy Bookstore Owner
* 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 MaxSatisfied(int[] customers, int[] grumpy, int minutes) {

}
}
```

C Solution:

```
/*
* Problem: Grumpy Bookstore Owner
* 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 maxSatisfied(int* customers, int customersSize, int* grumpy, int
grumpySize, int minutes) {

}
```

Go Solution:

```
// Problem: Grumpy Bookstore Owner
// 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 maxSatisfied(customers []int, grumpy []int, minutes int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSatisfied(customers: IntArray, grumpy: IntArray, minutes: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSatisfied(_ customers: [Int], _ grumpy: [Int], _ minutes: Int) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Grumpy Bookstore Owner
// 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 max_satisfied(customers: Vec<i32>, grumpy: Vec<i32>, minutes: i32) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} customers
# @param {Integer[]} grumpy
# @param {Integer} minutes
# @return {Integer}
def max_satisfied(customers, grumpy, minutes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $customers
* @param Integer[] $grumpy
* @param Integer $minutes
* @return Integer
*/
function maxSatisfied($customers, $grumpy, $minutes) {

}
}
```

Dart Solution:

```
class Solution {
int maxSatisfied(List<int> customers, List<int> grumpy, int minutes) {

}
}
```

Scala Solution:

```
object Solution {
def maxSatisfied(customers: Array[Int], grumpy: Array[Int], minutes: Int):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_satisfied(customers :: [integer], grumpy :: [integer], minutes ::
integer) :: integer
def max_satisfied(customers, grumpy, minutes) do

end
end
```

Erlang Solution:

```
-spec max_satisfied(Customers :: [integer()], Grumpy :: [integer()], Minutes
:: integer()) -> integer().
max_satisfied(Customers, Grumpy, Minutes) ->
.
```

Racket Solution:

```
(define/contract (max-satisfied customers grumpy minutes)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```