Problem 1815: Maximum Number of Groups Getting Fresh Donuts

Problem Information

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

Problem Description

There is a donuts shop that bakes donuts in batches of

batchSize

. They have a rule where they must serve

all

of the donuts of a batch before serving any donuts of the next batch. You are given an integer

batchSize

and an integer array

groups

, where

groups[i]

denotes that there is a group of

groups[i]

customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return

the

maximum

possible number of happy groups after rearranging the groups.

Example 1:

Input:

batchSize = 3, groups = [1,2,3,4,5,6]

Output:

4

Explanation:

You can arrange the groups as [6,2,4,5,1,3]. Then the 1

st

, 2

nd

, 4

th

, and 6

th

groups will be happy.

Example 2:

Input:

batchSize = 4, groups = [1,3,2,5,2,2,1,6]

Output:

4

Constraints:

1 <= batchSize <= 9

1 <= groups.length <= 30

1 <= groups[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxHappyGroups(int batchSize, vector<int>& groups) {

}
};
```

Java:

```
class Solution {
public int maxHappyGroups(int batchSize, int[] groups) {

}
```

```
}
```

Python3:

```
class Solution:
def maxHappyGroups(self, batchSize: int, groups: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxHappyGroups(self, batchSize, groups):
"""
:type batchSize: int
:type groups: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} batchSize
* @param {number[]} groups
* @return {number}
*/
var maxHappyGroups = function(batchSize, groups) {

};
```

TypeScript:

```
function maxHappyGroups(batchSize: number, groups: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxHappyGroups(int batchSize, int[] groups) {

}
}
```

C:

```
int maxHappyGroups(int batchSize, int* groups, int groupsSize) {

}
```

Go:

```
func maxHappyGroups(batchSize int, groups []int) int {

}
```

Kotlin:

```
class Solution {
fun maxHappyGroups(batchSize: Int, groups: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxHappyGroups(_ batchSize: Int, _ groups: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_happy_groups(batch_size: i32, groups: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} batch_size
# @param {Integer[]} groups
# @return {Integer}
def max_happy_groups(batch_size, groups)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $batchSize
* @param Integer[] $groups
* @return Integer
*/
function maxHappyGroups($batchSize, $groups) {

}
}
```

Dart:

```
class Solution {
int maxHappyGroups(int batchSize, List<int> groups) {

}
}
```

Scala:

```
object Solution {
def maxHappyGroups(batchSize: Int, groups: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_happy_groups(batch_size :: integer, groups :: [integer]) :: integer
def max_happy_groups(batch_size, groups) do

end
end
```

Erlang:

```
-spec max_happy_groups(BatchSize :: integer(), Groups :: [integer()]) ->
integer().
max_happy_groups(BatchSize, Groups) ->
.
```

Racket:

```
(define/contract (max-happy-groups batchSize groups)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxHappyGroups(int batchSize, vector<int>& groups) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxHappyGroups(int batchSize, int[] groups) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Groups Getting Fresh Donuts
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxHappyGroups(self, batchSize: int, groups: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxHappyGroups(self, batchSize, groups):
"""
:type batchSize: int
:type groups: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} batchSize
* @param {number[]} groups
* @return {number}
*/
var maxHappyGroups = function(batchSize, groups) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxHappyGroups(batchSize: number, groups: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxHappyGroups(int batchSize, int[] groups) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Groups Getting Fresh Donuts
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxHappyGroups(int batchSize, int* groups, int groupsSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Groups Getting Fresh Donuts
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxHappyGroups(batchSize int, groups []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxHappyGroups(batchSize: Int, groups: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxHappyGroups(_ batchSize: Int, _ groups: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Groups Getting Fresh Donuts
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_happy_groups(batch_size: i32, groups: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} batch_size
# @param {Integer[]} groups
# @return {Integer}
def max_happy_groups(batch_size, groups)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $batchSize
* @param Integer[] $groups
* @return Integer
*/
function maxHappyGroups($batchSize, $groups) {

}
}
```

Dart Solution:

```
class Solution {
int maxHappyGroups(int batchSize, List<int> groups) {

}
}
```

Scala Solution:

```
object Solution {
def maxHappyGroups(batchSize: Int, groups: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_happy_groups(batch_size :: integer, groups :: [integer]) :: integer
def max_happy_groups(batch_size, groups) do

end
end
```

Erlang Solution:

```
-spec max_happy_groups(BatchSize :: integer(), Groups :: [integer()]) ->
integer().
max_happy_groups(BatchSize, Groups) ->
```

```
.
```

Racket Solution:

```
(define/contract (max-happy-groups batchSize groups)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```