Problem 2064: Minimized Maximum of Products Distributed to Any Store

Problem Information

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

Problem Description

You are given an integer

n

indicating there are

n

specialty retail stores. There are

m

product types of varying amounts, which are given as a

0-indexed

integer array

quantities

, where

quantities[i]

represents the number of products of the i

th

product type.

You need to distribute

all products

to the retail stores following these rules:

A store can only be given

at most one product type

but can be given

any

amount of it.

After distribution, each store will have been given some number of products (possibly

0

). Let

x

represent the maximum number of products given to any store. You want

x

to be as small as possible, i.e., you want to

minimize

the

maximum

number of products that are given to any store.

Return

the minimum possible

x

.

Example 1:

Input:

n = 6, quantities = [11,6]

Output:

3

Explanation:

One optimal way is: - The 11 products of type 0 are distributed to the first four stores in these amounts: 2, 3, 3, 3 - The 6 products of type 1 are distributed to the other two stores in these amounts: 3, 3 The maximum number of products given to any store is max(2, 3, 3, 3, 3, 3) = 3.

Example 2:

Input:

n = 7, quantities = [15,10,10]

Output:

5

Explanation:

One optimal way is: - The 15 products of type 0 are distributed to the first three stores in these amounts: 5, 5, 5 - The 10 products of type 1 are distributed to the next two stores in these amounts: 5, 5 - The 10 products of type 2 are distributed to the last two stores in these amounts: 5, 5 The maximum number of products given to any store is max(5, 5, 5, 5, 5, 5, 5) = 5.

Example 3:

Input:

n = 1, quantities = [100000]

Output:

100000

Explanation:

The only optimal way is: - The 100000 products of type 0 are distributed to the only store. The maximum number of products given to any store is max(100000) = 100000.

Constraints:

m == quantities.length

1 <= m <= n <= 10

5

1 <= quantities[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minimizedMaximum(int n, vector<int>& quantities) {

}
};
```

Java:

```
class Solution {
public int minimizedMaximum(int n, int[] quantities) {

}
}
```

Python3:

```
class Solution:
def minimizedMaximum(self, n: int, quantities: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimizedMaximum(self, n, quantities):
"""
:type n: int
:type quantities: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} quantities
* @return {number}
*/
var minimizedMaximum = function(n, quantities) {

};
```

TypeScript:

```
function minimizedMaximum(n: number, quantities: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimizedMaximum(int n, int[] quantities) {

}
}
```

C:

```
int minimizedMaximum(int n, int* quantities, int quantitiesSize) {

}
```

Go:

```
func minimizedMaximum(n int, quantities []int) int {

}
```

Kotlin:

```
class Solution {
fun minimizedMaximum(n: Int, quantities: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimizedMaximum(_ n: Int, _ quantities: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimized_maximum(n: i32, quantities: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} quantities
# @return {Integer}
def minimized_maximum(n, quantities)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $quantities
* @return Integer
*/
function minimizedMaximum($n, $quantities) {

}
}
```

Dart:

```
class Solution {
int minimizedMaximum(int n, List<int> quantities) {

}
}
```

Scala:

```
object Solution {
def minimizedMaximum(n: Int, quantities: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimized_maximum(n :: integer, quantities :: [integer]) :: integer
def minimized_maximum(n, quantities) do

end
end
```

Erlang:

```
-spec minimized_maximum(N :: integer(), Quantities :: [integer()]) ->
integer().
minimized_maximum(N, Quantities) ->
.
```

Racket:

```
(define/contract (minimized-maximum n quantities)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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 minimizedMaximum(int n, vector<int>& quantities) {

}
};
```

Java Solution:

```
/**
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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 minimizedMaximum(int n, int[] quantities) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimized Maximum of Products Distributed to Any Store
Difficulty: Medium
Tags: array, greedy, search

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 minimizedMaximum(self, n: int, quantities: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizedMaximum(self, n, quantities):
"""
:type n: int
:type quantities: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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} n
* @param {number[]} quantities
* @return {number}
*/
var minimizedMaximum = function(n, quantities) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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 minimizedMaximum(n: number, quantities: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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 MinimizedMaximum(int n, int[] quantities) {

}
}
```

C Solution:

```
/*
* Problem: Minimized Maximum of Products Distributed to Any Store
* Difficulty: Medium
* Tags: array, greedy, search
*
* 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 minimizedMaximum(int n, int* quantities, int quantitiesSize) {

}
```

Go Solution:

```
// Problem: Minimized Maximum of Products Distributed to Any Store
// Difficulty: Medium
// Tags: array, greedy, search
//
// 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 minimizedMaximum(n int, quantities []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimizedMaximum(n: Int, quantities: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimizedMaximum(_ n: Int, _ quantities: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimized Maximum of Products Distributed to Any Store
// Difficulty: Medium
// Tags: array, greedy, search
//
// 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 minimized_maximum(n: i32, quantities: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} quantities
# @return {Integer}
def minimized_maximum(n, quantities)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $quantities
* @return Integer
*/
function minimizedMaximum($n, $quantities) {

}
}
```

Dart Solution:

```
class Solution {
int minimizedMaximum(int n, List<int> quantities) {

}
}
```

Scala Solution:

```
object Solution {
def minimizedMaximum(n: Int, quantities: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimized_maximum(n :: integer, quantities :: [integer]) :: integer
def minimized_maximum(n, quantities) do

end
end
```

Erlang Solution:

```
-spec minimized_maximum(N :: integer(), Quantities :: [integer()]) ->
integer().
minimized_maximum(N, Quantities) ->
.
```

Racket Solution:

```
(define/contract (minimized-maximum n quantities)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```