Problem 2604: Minimum Time to Eat All Grains

Problem Information

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

Problem Description

There are

n

hens and

m

grains on a line. You are given the initial positions of the hens and the grains in two integer arrays

hens

and

grains

of size

n

and

m

respectively.

Any hen can eat a grain if they are on the same position. The time taken for this is negligible. One hen can also eat multiple grains.

In

1

second, a hen can move right or left by

1

unit. The hens can move simultaneously and independently of each other.

Return

the

minimum

time to eat all grains if the hens act optimally.

Example 1:

Input:

hens = [3,6,7], grains = [2,4,7,9]

Output:

2

Explanation:

One of the ways hens eat all grains in 2 seconds is described below: - The first hen eats the grain at position 2 in 1 second. - The second hen eats the grain at position 4 in 2 seconds. - The third hen eats the grains at positions 7 and 9 in 2 seconds. So, the maximum time needed is 2. It can be proven that the hens cannot eat all grains before 2 seconds.

Example 2:

Input:

hens = [4,6,109,111,213,215], grains = [5,110,214]

Output:

1

Explanation:

One of the ways hens eat all grains in 1 second is described below: - The first hen eats the grain at position 5 in 1 second. - The fourth hen eats the grain at position 110 in 1 second. - The sixth hen eats the grain at position 214 in 1 second. - The other hens do not move. So, the maximum time needed is 1.

Constraints:

1 <= hens.length, grains.length <= 2*10

4

0 <= hens[i], grains[j] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimumTime(vector<int>& hens, vector<int>& grains) {

}
};
```

Java:

```
class Solution {
public int minimumTime(int[] hens, int[] grains) {

}
}
```

Python3:

```
class Solution:
def minimumTime(self, hens: List[int], grains: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumTime(self, hens, grains):
"""
:type hens: List[int]
:type grains: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} hens
* @param {number[]} grains
* @return {number}
*/
var minimumTime = function(hens, grains) {

};
```

TypeScript:

```
function minimumTime(hens: number[], grains: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumTime(int[] hens, int[] grains) {
```

```
}
}
```

C:

```
int minimumTime(int* hens, int hensSize, int* grains, int grainsSize) {

}
```

Go:

```
func minimumTime(hens []int, grains []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumTime(hens: IntArray, grains: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumTime(_ hens: [Int], _ grains: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_time(hens: Vec<i32>, grains: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} hens
# @param {Integer[]} grains
# @return {Integer}
def minimum_time(hens, grains)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $hens
* @param Integer[] $grains
* @return Integer
*/
function minimumTime($hens, $grains) {

}
}
```

Dart:

```
class Solution {
int minimumTime(List<int> hens, List<int> grains) {

}
}
```

Scala:

```
object Solution {
def minimumTime(hens: Array[Int], grains: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time(hens :: [integer], grains :: [integer]) :: integer
def minimum_time(hens, grains) do
```

```
end
end
```

Erlang:

```
-spec minimum_time(Hens :: [integer()], Grains :: [integer()]) -> integer().
minimum_time(Hens, Grains) ->
.
```

Racket:

```
(define/contract (minimum-time hens grains)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Eat All Grains
* Difficulty: Hard
* Tags: array, sort, 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 minimumTime(vector<int>& hens, vector<int>& grains) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Eat All Grains
```

```
* Difficulty: Hard
* Tags: array, sort, 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 minimumTime(int[] hens, int[] grains) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Eat All Grains
Difficulty: Hard
Tags: array, sort, 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 minimumTime(self, hens: List[int], grains: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumTime(self, hens, grains):
"""
:type hens: List[int]
:type grains: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Eat All Grains
* Difficulty: Hard
* Tags: array, sort, 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[]} hens
* @param {number[]} grains
* @return {number}
*/
var minimumTime = function(hens, grains) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Eat All Grains
* Difficulty: Hard
* Tags: array, sort, 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 minimumTime(hens: number[], grains: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Eat All Grains
* Difficulty: Hard
* Tags: array, sort, 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 MinimumTime(int[] hens, int[] grains) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Eat All Grains
* Difficulty: Hard
* Tags: array, sort, 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 minimumTime(int* hens, int hensSize, int* grains, int grainsSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Eat All Grains
// Difficulty: Hard
// Tags: array, sort, 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 minimumTime(hens []int, grains []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTime(hens: IntArray, grains: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumTime(_ hens: [Int], _ grains: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Eat All Grains
// Difficulty: Hard
// Tags: array, sort, 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 minimum_time(hens: Vec<i32>, grains: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} hens
# @param {Integer[]} grains
# @return {Integer}
def minimum_time(hens, grains)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $hens
* @param Integer[] $grains
* @return Integer
*/
function minimumTime($hens, $grains) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTime(List<int> hens, List<int> grains) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTime(hens: Array[Int], grains: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time(hens :: [integer], grains :: [integer]) :: integer
def minimum_time(hens, grains) do

end
end
```

Erlang Solution:

```
-spec minimum_time(Hens :: [integer()], Grains :: [integer()]) -> integer().
minimum_time(Hens, Grains) ->
.
```

Racket Solution:

```
(define/contract (minimum-time hens grains)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```