Problem 2403: Minimum Time to Kill All Monsters

Problem Information

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

Problem Description

You are given an integer array

power

where

power[i]

is the power of the

i

th

monster.

You start with

0

mana points, and each day you increase your mana points by

gain

where

gain

initially is equal to

1

.

Each day, after gaining

gain

mana, you can defeat a monster if your mana points are greater than or equal to the power of that monster. When you defeat a monster:

your mana points will be reset to

0

, and

the value of

gain

increases by

1

.

Return

the

minimum

number of days needed to defeat all the monsters.

Example 1:

Input:

power = [3,1,4]

Output:

4

Explanation:

The optimal way to beat all the monsters is to: - Day 1: Gain 1 mana point to get a total of 1 mana point. Spend all mana points to kill the 2

nd

monster. - Day 2: Gain 2 mana points to get a total of 2 mana points. - Day 3: Gain 2 mana points to get a total of 4 mana points. Spend all mana points to kill the 3

rd

monster. - Day 4: Gain 3 mana points to get a total of 3 mana points. Spend all mana points to kill the 1

st

monster. It can be proven that 4 is the minimum number of days needed.

Example 2:

Input:

power = [1,1,4]

Output:

4

Explanation:

The optimal way to beat all the monsters is to: - Day 1: Gain 1 mana point to get a total of 1 mana point. Spend all mana points to kill the 1

st

monster. - Day 2: Gain 2 mana points to get a total of 2 mana points. Spend all mana points to kill the 2

nd

monster. - Day 3: Gain 3 mana points to get a total of 3 mana points. - Day 4: Gain 3 mana points to get a total of 6 mana points. Spend all mana points to kill the 3

rd

monster. It can be proven that 4 is the minimum number of days needed.

Example 3:

Input:

power = [1,2,4,9]

Output:

6

Explanation:

The optimal way to beat all the monsters is to: - Day 1: Gain 1 mana point to get a total of 1 mana point. Spend all mana points to kill the 1st monster. - Day 2: Gain 2 mana points to get a total of 2 mana points. Spend all mana points to kill the 2nd monster. - Day 3: Gain 3 mana points to get a total of 3 mana points. - Day 4: Gain 3 mana points to get a total of 6 mana points. - Day 5: Gain 3 mana points to get a total of 9 mana points. Spend all mana points to kill the 4th monster. - Day 6: Gain 4 mana points to get a total of 4 mana points. Spend all mana points to kill the 3rd monster. It can be proven that 6 is the minimum number of days needed.

Constraints:

1 <= power.length <= 17

1 <= power[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long minimumTime(vector<int>& power) {

}
};
```

Java:

```
class Solution {
public long minimumTime(int[] power) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumTime(power: number[]): number {

};
```

C#:

```
public class Solution {
public long MinimumTime(int[] power) {

}
}
```

C:

```
long long minimumTime(int* power, int powerSize) {

}
```

Go:

```
func minimumTime(power []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumTime(power: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minimumTime(_ power: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_time(power: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} power
# @return {Integer}
def minimum_time(power)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $power
* @return Integer
*/
function minimumTime($power) {

}
}
```

Dart:

```
class Solution {
int minimumTime(List<int> power) {

}
```

```
}
```

Scala:

```
object Solution {
def minimumTime(power: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time(power :: [integer]) :: integer
def minimum_time(power) do

end
end
```

Erlang:

```
-spec minimum_time(Power :: [integer()]) -> integer().
minimum_time(Power) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Kill All Monsters
* 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:
long long minimumTime(vector<int>& power) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Kill All Monsters
* 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 long minimumTime(int[] power) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Kill All Monsters
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 minimumTime(self, power: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time to Kill All Monsters
* 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[]} power
* @return {number}
*/
var minimumTime = function(power) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Kill All Monsters
* 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 minimumTime(power: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Kill All Monsters
* 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 long MinimumTime(int[] power) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Kill All Monsters
* 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
*/

long long minimumTime(int* power, int powerSize) {
```

```
}
```

Go Solution:

```
// Problem: Minimum Time to Kill All Monsters
// 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 minimumTime(power []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTime(power: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumTime(_ power: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Kill All Monsters
// 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 minimum_time(power: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} power
# @return {Integer}
def minimum_time(power)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $power
* @return Integer
*/
function minimumTime($power) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTime(List<int> power) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTime(power: Array[Int]): Long = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time(power :: [integer]) :: integer
def minimum_time(power) do

end
end
```

Erlang Solution:

```
-spec minimum_time(Power :: [integer()]) -> integer().
minimum_time(Power) ->
.
```

Racket Solution:

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