Problem 1921: Eliminate Maximum Number of Monsters

Problem Information

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

Problem Description

You are playing a video game where you are defending your city from a group of

n

monsters. You are given a

0-indexed

integer array

dist

of size

n

, where

dist[i]

is the

initial distance

in kilometers of the

i

th

monster from the city.

The monsters walk toward the city at a

constant

speed. The speed of each monster is given to you in an integer array

speed

of size

n

, where

speed[i]

is the speed of the

i

th

monster in kilometers per minute.

You have a weapon that, once fully charged, can eliminate a

single

monster. However, the weapon takes

one minute

to charge. The weapon is fully charged at the very start.

You lose when any monster reaches your city. If a monster reaches the city at the exact moment the weapon is fully charged, it counts as a

loss

, and the game ends before you can use your weapon.

Return

the

maximum

number of monsters that you can eliminate before you lose, or

n

if you can eliminate all the monsters before they reach the city.

Example 1:

Input:

dist = [1,3,4], speed = [1,1,1]

Output:

3

Explanation:

In the beginning, the distances of the monsters are [1,3,4]. You eliminate the first monster. After a minute, the distances of the monsters are [X,2,3]. You eliminate the second monster. After a minute, the distances of the monsters are [X,X,2]. You eliminate the third monster. All 3 monsters can be eliminated.

Example 2:

Input:

dist = [1,1,2,3], speed = [1,1,1,1]

Output:

1

Explanation:

In the beginning, the distances of the monsters are [1,1,2,3]. You eliminate the first monster. After a minute, the distances of the monsters are [X,0,1,2], so you lose. You can only eliminate 1 monster.

Example 3:

Input:

dist = [3,2,4], speed = [5,3,2]

Output:

1

Explanation:

In the beginning, the distances of the monsters are [3,2,4]. You eliminate the first monster. After a minute, the distances of the monsters are [X,0,2], so you lose. You can only eliminate 1 monster.

Constraints:

n == dist.length == speed.length

1 <= n <= 10

5

1 <= dist[i], speed[i] <= 10

Code Snippets

C++:

```
class Solution {
public:
int eliminateMaximum(vector<int>& dist, vector<int>& speed) {

}
};
```

Java:

```
class Solution {
public int eliminateMaximum(int[] dist, int[] speed) {

}
}
```

Python3:

```
class Solution:
def eliminateMaximum(self, dist: List[int], speed: List[int]) -> int:
```

Python:

```
class Solution(object):
def eliminateMaximum(self, dist, speed):
"""
:type dist: List[int]
:type speed: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} dist
* @param {number[]} speed
```

```
 * @return {number}
 */
var eliminateMaximum = function(dist, speed) {

};
```

TypeScript:

```
function eliminateMaximum(dist: number[], speed: number[]): number {

};
```

C#:

```
public class Solution {
public int EliminateMaximum(int[] dist, int[] speed) {

}
}
```

C:

```
int eliminateMaximum(int* dist, int distSize, int* speed, int speedSize) {

}
```

Go:

```
func eliminateMaximum(dist []int, speed []int) int {

}
```

Kotlin:

```
class Solution {
fun eliminateMaximum(dist: IntArray, speed: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func eliminateMaximum(_ dist: [Int], _ speed: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn eliminate_maximum(dist: Vec<i32>, speed: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} dist
# @param {Integer[]} speed
# @return {Integer}
def eliminate_maximum(dist, speed)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $dist
* @param Integer[] $speed
* @return Integer
*/
function eliminateMaximum($dist, $speed) {

}
}
```

Dart:

```
class Solution {
int eliminateMaximum(List<int> dist, List<int> speed) {

}
```

```
}
```

Scala:

```
object Solution {
def eliminateMaximum(dist: Array[Int], speed: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec eliminate_maximum(dist :: [integer], speed :: [integer]) :: integer
def eliminate_maximum(dist, speed) do

end
end
```

Erlang:

```
-spec eliminate_maximum(Dist :: [integer()], Speed :: [integer()]) ->
integer().
eliminate_maximum(Dist, Speed) ->
.
```

Racket:

```
(define/contract (eliminate-maximum dist speed)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
```

```
*
* 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 eliminateMaximum(vector<int>& dist, vector<int>& speed) {

}
};
```

Java Solution:

```
/**
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 eliminateMaximum(int[] dist, int[] speed) {

}
}
```

Python3 Solution:

```
"""
Problem: Eliminate Maximum Number of Monsters
Difficulty: Medium
Tags: array, greedy, sort

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 eliminateMaximum(self, dist: List[int], speed: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def eliminateMaximum(self, dist, speed):
"""
:type dist: List[int]
:type speed: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[]} dist
* @param {number[]} speed
* @return {number}
*/
var eliminateMaximum = function(dist, speed) {

};
```

TypeScript Solution:

```
/**
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 eliminateMaximum(dist: number[], speed: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 EliminateMaximum(int[] dist, int[] speed) {

}
}
```

C Solution:

```
/*
* Problem: Eliminate Maximum Number of Monsters
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 eliminateMaximum(int* dist, int distSize, int* speed, int speedSize) {

}
```

Go Solution:

```
// Problem: Eliminate Maximum Number of Monsters
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 eliminateMaximum(dist []int, speed []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun eliminateMaximum(dist: IntArray, speed: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func eliminateMaximum(_ dist: [Int], _ speed: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Eliminate Maximum Number of Monsters
// Difficulty: Medium
// Tags: array, greedy, sort
```

```
//
// 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 eliminate_maximum(dist: Vec<i32>, speed: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} dist
# @param {Integer[]} speed
# @return {Integer}
def eliminate_maximum(dist, speed)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $dist
* @param Integer[] $speed
* @return Integer
*/
function eliminateMaximum($dist, $speed) {

}
}
```

Dart Solution:

```
class Solution {
int eliminateMaximum(List<int> dist, List<int> speed) {

}
}
```

Scala Solution:

```
object Solution {
def eliminateMaximum(dist: Array[Int], speed: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec eliminate_maximum(dist :: [integer], speed :: [integer]) :: integer
def eliminate_maximum(dist, speed) do

end
end
```

Erlang Solution:

```
-spec eliminate_maximum(Dist :: [integer()], Speed :: [integer()]) ->
integer().
eliminate_maximum(Dist, Speed) ->
.
```

Racket Solution:

```
(define/contract (eliminate-maximum dist speed)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```