Problem 3273: Minimum Amount of Damage Dealt to Bob

Problem Information

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

Problem Description

You are given an integer

power

and two integer arrays

damage

and

health

, both having length

n

.

Bob has

n

enemies, where enemy

i

will deal Bob

damage[i]

points

of damage per second while they are

alive

(i.e.

health[i] > 0

).

Every second,

after

the enemies deal damage to Bob, he chooses

one

of the enemies that is still

alive

and deals

power

points of damage to them.

Determine the

minimum

total amount of damage points that will be dealt to Bob before

all

n

enemies are

dead

.

Example 1:

Input:

power = 4, damage = [1,2,3,4], health = [4,5,6,8]

Output:

39

Explanation:

Attack enemy 3 in the first two seconds, after which enemy 3 will go down, the number of damage points dealt to Bob is

10 + 10 = 20

points.

Attack enemy 2 in the next two seconds, after which enemy 2 will go down, the number of damage points dealt to Bob is

6 + 6 = 12

points.

Attack enemy 0 in the next second, after which enemy 0 will go down, the number of damage points dealt to Bob is

3

points.

Attack enemy 1 in the next two seconds, after which enemy 1 will go down, the number of damage points dealt to Bob is

2 + 2 = 4

points.

Example 2:

Input:

power = 1, damage = [1,1,1,1], health = [1,2,3,4]

Output:

20

Explanation:

Attack enemy 0 in the first second, after which enemy 0 will go down, the number of damage points dealt to Bob is

4

points.

Attack enemy 1 in the next two seconds, after which enemy 1 will go down, the number of damage points dealt to Bob is

3 + 3 = 6

points.

Attack enemy 2 in the next three seconds, after which enemy 2 will go down, the number of damage points dealt to Bob is

2 + 2 + 2 = 6

points.

Attack enemy 3 in the next four seconds, after which enemy 3 will go down, the number of damage points dealt to Bob is

1 + 1 + 1 + 1 = 4

points.

Example 3:

Input:

power = 8, damage = [40], health = [59]

Output:

320

Constraints:

1 <= power <= 10

4

1 <= n == damage.length == health.length <= 10

5

1 <= damage[i], health[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
long long minDamage(int power, vector<int>& damage, vector<int>& health) {

}
};
```

Java:

```
class Solution {
public long minDamage(int power, int[] damage, int[] health) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} power
* @param {number[]} damage
* @param {number[]} health
* @return {number}
*/
```

```
var minDamage = function(power, damage, health) {

};
```

TypeScript:

```
function minDamage(power: number, damage: number[], health: number[]): number
{

};
```

C#:

```
public class Solution {
public long MinDamage(int power, int[] damage, int[] health) {

}
}
```

C:

```
long long minDamage(int power, int* damage, int damageSize, int* health, int
healthSize) {

}
```

Go:

```
func minDamage(power int, damage []int, health []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minDamage(power: Int, damage: IntArray, health: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minDamage(_ power: Int, _ damage: [Int], _ health: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_damage(power: i32, damage: Vec<i32>, health: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} power
# @param {Integer[]} damage
# @param {Integer[]} health
# @return {Integer}
def min_damage(power, damage, health)

end
```

PHP:

```
class Solution {

/**
* @param Integer $power
* @param Integer[] $damage
* @param Integer[] $health
* @return Integer
*/
function minDamage($power, $damage, $health) {

}
}
```

Dart:

```
class Solution {
int minDamage(int power, List<int> damage, List<int> health) {


}
}
```

Scala:

```
object Solution {
def minDamage(power: Int, damage: Array[Int], health: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_damage(power :: integer, damage :: [integer], health :: [integer])
:: integer
def min_damage(power, damage, health) do

end
end
```

Erlang:

```
-spec min_damage(Power :: integer(), Damage :: [integer()], Health ::
[integer()]) -> integer().
min_damage(Power, Damage, Health) ->
.
```

Racket:

```
(define/contract (min-damage power damage health)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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:
long long minDamage(int power, vector<int>& damage, vector<int>& health) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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 long minDamage(int power, int[] damage, int[] health) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Amount of Damage Dealt to Bob
Difficulty: Hard
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 minDamage(self, power: int, damage: List[int], health: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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} power
* @param {number[]} damage
* @param {number[]} health
* @return {number}
*/
var minDamage = function(power, damage, health) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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 minDamage(power: number, damage: number[], health: number[]): number
{

};
```

C# Solution:

```
/*
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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 long MinDamage(int power, int[] damage, int[] health) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Amount of Damage Dealt to Bob
* Difficulty: Hard
* 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
*/

long long minDamage(int power, int* damage, int damageSize, int* health, int
healthSize) {

}
```

Go Solution:

```
// Problem: Minimum Amount of Damage Dealt to Bob
// Difficulty: Hard
// 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 minDamage(power int, damage []int, health []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minDamage(power: Int, damage: IntArray, health: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minDamage(_ power: Int, _ damage: [Int], _ health: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Amount of Damage Dealt to Bob
// Difficulty: Hard
// 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 min_damage(power: i32, damage: Vec<i32>, health: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} power
# @param {Integer[]} damage
# @param {Integer[]} health
# @return {Integer}
def min_damage(power, damage, health)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $power
* @param Integer[] $damage
* @param Integer[] $health
* @return Integer
*/
function minDamage($power, $damage, $health) {

}

}
```

Dart Solution:

```
class Solution {
int minDamage(int power, List<int> damage, List<int> health) {

}
}
```

Scala Solution:

```
object Solution {
def minDamage(power: Int, damage: Array[Int], health: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_damage(power :: integer, damage :: [integer], health :: [integer])
:: integer
def min_damage(power, damage, health) do

end
end
```

Erlang Solution:

```
-spec min_damage(Power :: integer(), Damage :: [integer()], Health ::
[integer()]) -> integer().
min_damage(Power, Damage, Health) ->
.
```

Racket Solution:

```
(define/contract (min-damage power damage health)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```