

Problem 2214: Minimum Health to Beat Game

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are playing a game that has

n

levels numbered from

0

to

$n - 1$

. You are given a

0-indexed

integer array

damage

where

$\text{damage}[i]$

is the amount of health you will lose to complete the

i

th

level.

You are also given an integer

armor

. You may use your armor ability

at most once

during the game on

any

level which will protect you from

at most

armor

damage.

You must complete the levels in order and your health must be

greater than

0

at all times to beat the game.

Return

the

minimum

health you need to start with to beat the game.

Example 1:

Input:

damage = [2,7,4,3], armor = 4

Output:

13

Explanation:

One optimal way to beat the game starting at 13 health is: On round 1, take 2 damage. You have $13 - 2 = 11$ health. On round 2, take 7 damage. You have $11 - 7 = 4$ health. On round 3, use your armor to protect you from 4 damage. You have $4 - 0 = 4$ health. On round 4, take 3 damage. You have $4 - 3 = 1$ health. Note that 13 is the minimum health you need to start with to beat the game.

Example 2:

Input:

damage = [2,5,3,4], armor = 7

Output:

10

Explanation:

One optimal way to beat the game starting at 10 health is: On round 1, take 2 damage. You have $10 - 2 = 8$ health. On round 2, use your armor to protect you from 5 damage. You have $8 - 0 = 8$ health. On round 3, take 3 damage. You have $8 - 3 = 5$ health. On round 4, take 4 damage. You have $5 - 4 = 1$ health. Note that 10 is the minimum health you need to start with to beat the game.

Example 3:

Input:

damage = [3,3,3], armor = 0

Output:

10

Explanation:

One optimal way to beat the game starting at 10 health is: On round 1, take 3 damage. You have $10 - 3 = 7$ health. On round 2, take 3 damage. You have $7 - 3 = 4$ health. On round 3, take 3 damage. You have $4 - 3 = 1$ health. Note that you did not use your armor ability.

Constraints:

$n == \text{damage.length}$

$1 \leq n \leq 10$

5

$0 \leq \text{damage}[i] \leq 10$

5

$0 \leq \text{armor} \leq 10$

5

Code Snippets

C++:

```
class Solution {
public:
    long long minimumHealth(vector<int>& damage, int armor) {
```

```
    }
};
```

Java:

```
class Solution {
public long minimumHealth(int[] damage, int armor) {

}
}
```

Python3:

```
class Solution:
def minimumHealth(self, damage: List[int], armor: int) -> int:
```

Python:

```
class Solution(object):
def minimumHealth(self, damage, armor):
"""
:type damage: List[int]
:type armor: int
:rtype: int
"""


```

JavaScript:

```
/**
 * @param {number[]} damage
 * @param {number} armor
 * @return {number}
 */
var minimumHealth = function(damage, armor) {

};
```

TypeScript:

```
function minimumHealth(damage: number[], armor: number): number {
}
```

C#:

```
public class Solution {  
    public long MinimumHealth(int[] damage, int armor) {  
  
    }  
}
```

C:

```
long long minimumHealth(int* damage, int damageSize, int armor) {  
  
}
```

Go:

```
func minimumHealth(damage []int, armor int) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun minimumHealth(damage: IntArray, armor: Int): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func minimumHealth(_ damage: [Int], _ armor: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn minimum_health(damage: Vec<i32>, armor: i32) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} damage
# @param {Integer} armor
# @return {Integer}
def minimum_health(damage, armor)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $damage
     * @param Integer $armor
     * @return Integer
     */
    function minimumHealth($damage, $armor) {

    }
}
```

Dart:

```
class Solution {
    int minimumHealth(List<int> damage, int armor) {
        }
}
```

Scala:

```
object Solution {
    def minimumHealth(damage: Array[Int], armor: Int): Long = {
        }
}
```

Elixir:

```
defmodule Solution do
    @spec minimum_health(integer(), integer()) :: integer()
```

```
def minimum_health(damage, armor) do
  end
end
```

Erlang:

```
-spec minimum_health(Damage :: [integer()], Armor :: integer()) -> integer().
minimum_health(Damage, Armor) ->
  .
```

Racket:

```
(define/contract (minimum-health damage armor)
  (-> (listof exact-integer?) exact-integer? exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimum Health to Beat Game
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumHealth(vector<int>& damage, int armor) {

    }
};
```

Java Solution:

```

/**
 * Problem: Minimum Health to Beat Game
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumHealth(int[] damage, int armor) {
        return 0;
    }
}

```

Python3 Solution:

```

"""
Problem: Minimum Health to Beat Game
Difficulty: Medium
Tags: array, greedy

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 minimumHealth(self, damage: List[int], armor: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def minimumHealth(self, damage, armor):
        """
:type damage: List[int]
:type armor: int
:rtype: int
"""

```

JavaScript Solution:

```
/**  
 * Problem: Minimum Health to Beat Game  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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[]} damage  
 * @param {number} armor  
 * @return {number}  
 */  
var minimumHealth = function(damage, armor) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Minimum Health to Beat Game  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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 minimumHealth(damage: number[], armor: number): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Minimum Health to Beat Game  
 * Difficulty: Medium
```

```

* Tags: array, greedy
*
* 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 MinimumHealth(int[] damage, int armor) {
        }
    }

```

C Solution:

```

/*
 * Problem: Minimum Health to Beat Game
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 minimumHealth(int* damage, int damageSize, int armor) {
}

```

Go Solution:

```

// Problem: Minimum Health to Beat Game
// Difficulty: Medium
// Tags: array, greedy
//
// 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 minimumHealth(damage []int, armor int) int64 {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun minimumHealth(damage: IntArray, armor: Int): Long {  
        //  
        //  
        return 0L  
    }  
}
```

Swift Solution:

```
class Solution {  
    func minimumHealth(_ damage: [Int], _ armor: Int) -> Int {  
        //  
        //  
        return 0  
    }  
}
```

Rust Solution:

```
// Problem: Minimum Health to Beat Game  
// Difficulty: Medium  
// Tags: array, greedy  
//  
// 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_health(damage: Vec<i32>, armor: i32) -> i64 {  
        //  
        //  
        return 0  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} damage  
# @param {Integer} armor  
# @return {Integer}  
def minimum_health(damage, armor)
```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $damage  
     * @param Integer $armor  
     * @return Integer  
     */  
    function minimumHealth($damage, $armor) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
  int minimumHealth(List<int> damage, int armor) {  
  
  }  
}
```

Scala Solution:

```
object Solution {  
  def minimumHealth(damage: Array[Int], armor: Int): Long = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec minimum_health([integer], integer) :: integer  
  def minimum_health(damage, armor) do  
  
  end  
end
```

Erlang Solution:

```
-spec minimum_health(Damage :: [integer()], Armor :: integer()) -> integer().  
minimum_health(Damage, Armor) ->  
. 
```

Racket Solution:

```
(define/contract (minimum-health damage armor)  
(-> (listof exact-integer?) exact-integer? exact-integer?)  
) 
```