Problem 2383: Minimum Hours of Training to Win a Competition

Problem Information

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

Problem Description

You are entering a competition, and are given two

positive

integers

initialEnergy

and

initialExperience

denoting your initial energy and initial experience respectively.

You are also given two

0-indexed

integer arrays

energy

and

experience

, both of length

n

.

You will face

n

opponents

in order

. The energy and experience of the

i

th

opponent is denoted by

energy[i]

and

experience[i]

respectively. When you face an opponent, you need to have both

strictly

greater experience and energy to defeat them and move to the next opponent if available.

Defeating the

i

th

opponent

increases

your experience by

experience[i]

, but

decreases

your energy by

energy[i]

.

Before starting the competition, you can train for some number of hours. After each hour of training, you can

either

choose to increase your initial experience by one, or increase your initial energy by one.

Return

the

minimum

number of training hours required to defeat all

n

opponents

.

Example 1:

Input:

initialEnergy = 5, initialExperience = 3, energy = [1,4,3,2], experience = [2,6,3,1]

Output:

8

Explanation:

You can increase your energy to 11 after 6 hours of training, and your experience to 5 after 2 hours of training. You face the opponents in the following order: - You have more energy and experience than the 0

th

opponent so you win. Your energy becomes 11 - 1 = 10, and your experience becomes 5 + 2 = 7. - You have more energy and experience than the 1

st

opponent so you win. Your energy becomes 10 - 4 = 6, and your experience becomes 7 + 6 = 13. - You have more energy and experience than the 2

nd

opponent so you win. Your energy becomes 6 - 3 = 3, and your experience becomes 13 + 3 = 16. - You have more energy and experience than the 3

rd

opponent so you win. Your energy becomes 3 - 2 = 1, and your experience becomes 16 + 1 = 17. You did a total of 6 + 2 = 8 hours of training before the competition, so we return 8. It can be proven that no smaller answer exists.

Example 2:

Input:

initialEnergy = 2, initialExperience = 4, energy = [1], experience = [3]

Output:

0

Explanation:

You do not need any additional energy or experience to win the competition, so we return 0.

Constraints:

n == energy.length == experience.length

1 <= n <= 100

1 <= initialEnergy, initialExperience, energy[i], experience[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int minNumberOfHours(int initialEnergy, int initialExperience, vector<int>&
energy, vector<int>& experience) {

}
};
```

Java:

```
class Solution {
public int minNumberOfHours(int initialEnergy, int initialExperience, int[]
energy, int[] experience) {
```

```
}
}
```

Python3:

```
class Solution:
def minNumberOfHours(self, initialEnergy: int, initialExperience: int,
energy: List[int], experience: List[int]) -> int:
```

Python:

```
class Solution(object):
def minNumberOfHours(self, initialEnergy, initialExperience, energy,
experience):
"""
:type initialEnergy: int
:type initialExperience: int
:type energy: List[int]
:type experience: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} initialEnergy
* @param {number} initialExperience
* @param {number[]} energy
* @param {number[]} experience
* @return {number}
*/
var minNumberOfHours = function(initialEnergy, initialExperience, energy,
experience) {

};
```

TypeScript:

```
function minNumberOfHours(initialEnergy: number, initialExperience: number,
energy: number[], experience: number[]): number {
```

```
};
```

C#:

```
public class Solution {
public int MinNumberOfHours(int initialEnergy, int initialExperience, int[]
energy, int[] experience) {

}
}
```

C:

```
int minNumberOfHours(int initialEnergy, int initialExperience, int* energy,
int energySize, int* experience, int experienceSize) {

}
```

Go:

```
func minNumberOfHours(initialEnergy int, initialExperience int, energy []int,
experience []int) int {

}
```

Kotlin:

```
class Solution {
fun minNumberOfHours(initialEnergy: Int, initialExperience: Int, energy:
IntArray, experience: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minNumberOfHours(_ initialEnergy: Int, _ initialExperience: Int, _
energy: [Int], _ experience: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_number_of_hours(initial_energy: i32, initial_experience: i32,
energy: Vec<i32>, experience: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} initial_energy
# @param {Integer} initial_experience
# @param {Integer[]} energy
# @param {Integer[]} experience
# @return {Integer}
def min_number_of_hours(initial_energy, initial_experience, energy,
experience)

end
```

PHP:

```
class Solution {

/**
* @param Integer $initialEnergy
* @param Integer $initialExperience
* @param Integer[] $energy
* @param Integer[] $experience
* @return Integer
*/
function minNumberOfHours($initialEnergy, $initialExperience, $energy,
$experience) {

}
}
```

Dart:

```
class Solution {
int minNumberOfHours(int initialEnergy, int initialExperience, List<int>
energy, List<int> experience) {
```

```
}
}
```

Scala:

```
object Solution {
def minNumberOfHours(initialEnergy: Int, initialExperience: Int, energy:
Array[Int], experience: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_number_of_hours(initial_energy :: integer, initial_experience ::
integer, energy :: [integer], experience :: [integer]) :: integer
def min_number_of_hours(initial_energy, initial_experience, energy,
experience) do

end
end
```

Erlang:

```
-spec min_number_of_hours(InitialEnergy :: integer(), InitialExperience ::
integer(), Energy :: [integer()], Experience :: [integer()]) -> integer().
min_number_of_hours(InitialEnergy, InitialExperience, Energy, Experience) ->
.
```

Racket:

```
(define/contract (min-number-of-hours initialEnergy initialExperience energy
experience)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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:
int minNumberOfHours(int initialEnergy, int initialExperience, vector<int>&
energy, vector<int>& experience) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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 int minNumberOfHours(int initialEnergy, int initialExperience, int[]
energy, int[] experience) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Hours of Training to Win a Competition
```

```
Difficulty: Easy
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 minNumberOfHours(self, initialEnergy: int, initialExperience: int,
energy: List[int], experience: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minNumberOfHours(self, initialEnergy, initialExperience, energy,
experience):
"""
:type initialEnergy: int
:type initialExperience: int
:type energy: List[int]
:type experience: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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} initialEnergy
```

```
* @param {number} initialExperience
* @param {number[]} energy
* @param {number[]} experience
* @return {number}
*/
var minNumberOfHours = function(initialEnergy, initialExperience, energy,
experience) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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 minNumberOfHours(initialEnergy: number, initialExperience: number,
energy: number[], experience: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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 int MinNumberOfHours(int initialEnergy, int initialExperience, int[]
```

```
energy, int[] experience) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Hours of Training to Win a Competition
* Difficulty: Easy
* 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
*/

int minNumberOfHours(int initialEnergy, int initialExperience, int* energy,
int energySize, int* experience, int experienceSize) {

}
```

Go Solution:

```
// Problem: Minimum Hours of Training to Win a Competition
// Difficulty: Easy
// 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 minNumberOfHours(initialEnergy int, initialExperience int, energy []int,
experience []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minNumberOfHours(initialEnergy: Int, initialExperience: Int, energy:
IntArray, experience: IntArray): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func minNumberOfHours(_ initialEnergy: Int, _ initialExperience: Int, _
energy: [Int], _ experience: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Hours of Training to Win a Competition
// Difficulty: Easy
// 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 min_number_of_hours(initial_energy: i32, initial_experience: i32,
energy: Vec<i32>, experience: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} initial_energy
# @param {Integer} initial_experience
# @param {Integer[]} energy
# @param {Integer[]} experience
# @return {Integer}
def min_number_of_hours(initial_energy, initial_experience, energy,
experience)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $initialEnergy
* @param Integer $initialExperience
* @param Integer[] $energy
* @param Integer[] $experience
* @return Integer
*/
function minNumberOfHours($initialEnergy, $initialExperience, $energy,
$experience) {

}
}
```

Dart Solution:

```
class Solution {
int minNumberOfHours(int initialEnergy, int initialExperience, List<int>
energy, List<int> experience) {

}
}
```

Scala Solution:

```
object Solution {
def minNumberOfHours(initialEnergy: Int, initialExperience: Int, energy:
Array[Int], experience: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_number_of_hours(initial_energy :: integer, initial_experience ::
integer, energy :: [integer], experience :: [integer]) :: integer
def min_number_of_hours(initial_energy, initial_experience, energy,
experience) do
```

```
end
end
```

Erlang Solution:

```
-spec min_number_of_hours(InitialEnergy :: integer(), InitialExperience ::
integer(), Energy :: [integer()], Experience :: [integer()]) -> integer().
min_number_of_hours(InitialEnergy, InitialExperience, Energy, Experience) ->
.
```

Racket Solution:

```
(define/contract (min-number-of-hours initialEnergy initialExperience energy
experience)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```