Problem 1665: Minimum Initial Energy to Finish Tasks

Problem Information

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

Problem Description

You are given an array

tasks

where

tasks[i] = [actual

i

, minimum

i

]

:

actual

i

is the actual amount of energy you

spend to finish

the

i

th

task.

minimum

i

is the minimum amount of energy you

require to begin

the

i

th

task.

For example, if the task is

[10, 12]

and your current energy is

11

, you cannot start this task. However, if your current energy is

13

, you can complete this task, and your energy will be

3

after finishing it.

You can finish the tasks in

any order

you like.

Return

the

minimum

initial amount of energy you will need

to finish all the tasks

.

Example 1:

Input:

tasks = [[1,2],[2,4],[4,8]]

Output:

8

Explanation:

Starting with 8 energy, we finish the tasks in the following order: - 3rd task. Now energy = 8 - 4 = 4. - 2nd task. Now energy = 4 - 2 = 2. - 1st task. Now energy = 2 - 1 = 1. Notice that even though we have leftover energy, starting with 7 energy does not work because we cannot do the 3rd task.

Example 2:

Input:

tasks = [[1,3],[2,4],[10,11],[10,12],[8,9]]

Output:

32

Explanation:

Starting with 32 energy, we finish the tasks in the following order: - 1st task. Now energy = 32 - 1 = 31. - 2nd task. Now energy = 31 - 2 = 29. - 3rd task. Now energy = 29 - 10 = 19. - 4th task. Now energy = 19 - 10 = 9. - 5th task. Now energy = 9 - 8 = 1.

Example 3:

Input:

tasks = [[1,7],[2,8],[3,9],[4,10],[5,11],[6,12]]

Output:

27

Explanation:

Starting with 27 energy, we finish the tasks in the following order: - 5th task. Now energy = 27 - 5 = 22. - 2nd task. Now energy = 22 - 2 = 20. - 3rd task. Now energy = 20 - 3 = 17. - 1st task. Now energy = 17 - 1 = 16. - 4th task. Now energy = 16 - 4 = 12. - 6th task. Now energy = 12 - 6 = 6.

Constraints:

1 <= tasks.length <= 10

5

1 <= actual

i

<= minimum

i

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
int minimumEffort(vector<vector<int>>& tasks) {

}
};
```

Java:

```
class Solution {
public int minimumEffort(int[][] tasks) {

}
}
```

Python3:

```
class Solution:
def minimumEffort(self, tasks: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumEffort(self, tasks):
```

```
"""
:type tasks: List[List[int]]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minimumEffort(tasks: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumEffort(int[][] tasks) {

}
}
```

C:

```
int minimumEffort(int** tasks, int tasksSize, int* tasksColSize) {

}
```

Go:

```
func minimumEffort(tasks [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumEffort(tasks: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumEffort(_ tasks: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_effort(tasks: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} tasks
# @return {Integer}
def minimum_effort(tasks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $tasks
* @return Integer
*/
function minimumEffort($tasks) {

}
}
```

Dart:

```
class Solution {
int minimumEffort(List<List<int>> tasks) {

}
}
```

Scala:

```
object Solution {
def minimumEffort(tasks: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_effort(tasks :: [[integer]]) :: integer
def minimum_effort(tasks) do

end
end
```

Erlang:

```
-spec minimum_effort(Tasks :: [[integer()]]) -> integer().
minimum_effort(Tasks) ->
.
```

Racket:

```
(define/contract (minimum-effort tasks)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Initial Energy to Finish Tasks
* 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:
int minimumEffort(vector<vector<int>>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Initial Energy to Finish Tasks
* 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 int minimumEffort(int[][] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Initial Energy to Finish Tasks
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 minimumEffort(self, tasks: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Initial Energy to Finish Tasks
* 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[][]} tasks
* @return {number}
*/
var minimumEffort = function(tasks) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Initial Energy to Finish Tasks
* 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 minimumEffort(tasks: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Initial Energy to Finish Tasks
* 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 int MinimumEffort(int[][] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Initial Energy to Finish Tasks
* 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
```

```
*/

int minimumEffort(int** tasks, int tasksSize, int* tasksColSize) {

}
```

Go Solution:

```
// Problem: Minimum Initial Energy to Finish Tasks
// 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 minimumEffort(tasks [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumEffort(tasks: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumEffort(_ tasks: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Initial Energy to Finish Tasks
// 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 minimum_effort(tasks: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} tasks
# @return {Integer}
def minimum_effort(tasks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $tasks
* @return Integer
*/
function minimumEffort($tasks) {

}
}
```

Dart Solution:

```
class Solution {
int minimumEffort(List<List<int>> tasks) {

}
}
```

Scala Solution:

```
object Solution {
def minimumEffort(tasks: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_effort(tasks :: [[integer]]) :: integer
def minimum_effort(tasks) do

end
end
```

Erlang Solution:

```
-spec minimum_effort(Tasks :: [[integer()]]) -> integer().
minimum_effort(Tasks) ->
.
```

Racket Solution:

```
(define/contract (minimum-effort tasks)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```