Problem 2071: Maximum Number of Tasks You Can Assign

Problem Information

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

Problem Description

You have

n

tasks and

m

workers. Each task has a strength requirement stored in a

0-indexed

integer array

tasks

, with the

i

th

task requiring

tasks[i]

strength to complete. The strength of each worker is stored in a

0-indexed

integer array

workers

, with the

j

th

worker having

workers[j]

strength. Each worker can only be assigned to a

single

task and must have a strength

greater than or equal

to the task's strength requirement (i.e.,

workers[j] >= tasks[i]

).

Additionally, you have

pills

magical pills that will

increase a worker's strength

by

strength

. You can decide which workers receive the magical pills, however, you may only give each worker

at most one

magical pill.

Given the

0-indexed

integer arrays

tasks

and

workers

and the integers

pills

and

strength

, return

the

maximum

number of tasks that can be completed.

Example 1:

Input:

tasks = [

3

,

2

,

1

], workers = [

0

,

3

,

3

], pills = 1, strength = 1

Output:

3

Explanation:

We can assign the magical pill and tasks as follows: - Give the magical pill to worker 0. - Assign worker 0 to task 2 (0 + 1 >= 1) - Assign worker 1 to task 1 (3 >= 2) - Assign worker 2 to task 0 (3 >= 3)

Example 2:

Input:

tasks = [

5

,4], workers = [

0

,0,0], pills = 1, strength = 5

Output:

1

Explanation:

We can assign the magical pill and tasks as follows: - Give the magical pill to worker 0. - Assign worker 0 to task 0 (0 + 5 >= 5)

Example 3:

Input:

tasks = [

10

,

15

,30], workers = [

0

,

10

,10,10,10], pills = 3, strength = 10

Output:

2

Explanation:

We can assign the magical pills and tasks as follows: - Give the magical pill to worker 0 and worker 1. - Assign worker 0 to task 0 (0 + 10 >= 10) - Assign worker 1 to task 1 (10 + 10 >= 15) The last pill is not given because it will not make any worker strong enough for the last task.

Constraints:

n == tasks.length

m == workers.length

1 <= n, m <= 5 * 10

4

0 <= pills <= m

0 <= tasks[i], workers[j], strength <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int
strength) {

}
};
```

Java:

```
class Solution {
public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength)
{

}
}
```

Python3:

```
class Solution:
def maxTaskAssign(self, tasks: List[int], workers: List[int], pills: int,
strength: int) -> int:
```

Python:

```
class Solution(object):
def maxTaskAssign(self, tasks, workers, pills, strength):
"""
:type tasks: List[int]
:type workers: List[int]
:type pills: int
:type strength: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} tasks
* @param {number[]} workers
* @param {number} pills
* @param {number} strength
* @return {number}
*/
var maxTaskAssign = function(tasks, workers, pills, strength) {

};
```

TypeScript:

```
function maxTaskAssign(tasks: number[], workers: number[], pills: number,
strength: number): number {

};
```

C#:

```
public class Solution {
public int MaxTaskAssign(int[] tasks, int[] workers, int pills, int strength)
{

}
}
```

C:

```
int maxTaskAssign(int* tasks, int tasksSize, int* workers, int workersSize,
int pills, int strength) {

}
```

Go:

```
func maxTaskAssign(tasks []int, workers []int, pills int, strength int) int {

}
```

Kotlin:

```
class Solution {
fun maxTaskAssign(tasks: IntArray, workers: IntArray, pills: Int, strength:
```

```
Int): Int {

}
}
```

Swift:

```
class Solution {
func maxTaskAssign(_ tasks: [Int], _ workers: [Int], _ pills: Int, _
strength: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_task_assign(tasks: Vec<i32>, workers: Vec<i32>, pills: i32,
strength: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tasks
# @param {Integer[]} workers
# @param {Integer} pills
# @param {Integer} strength
# @return {Integer}
def max_task_assign(tasks, workers, pills, strength)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $tasks
* @param Integer[] $workers
* @param Integer $pills
* @param Integer $strength
* @return Integer
*/
function maxTaskAssign($tasks, $workers, $pills, $strength) {

}
}
```

Dart:

```
class Solution {
int maxTaskAssign(List<int> tasks, List<int> workers, int pills, int
strength) {

}
}
```

Scala:

```
object Solution {
def maxTaskAssign(tasks: Array[Int], workers: Array[Int], pills: Int,
strength: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_task_assign(tasks :: [integer], workers :: [integer], pills ::
integer, strength :: integer) :: integer
def max_task_assign(tasks, workers, pills, strength) do

end
end
```

Erlang:

```
-spec max_task_assign(Tasks :: [integer()], Workers :: [integer()], Pills ::
integer(), Strength :: integer()) -> integer().
max_task_assign(Tasks, Workers, Pills, Strength) ->
.
```

Racket:

```
(define/contract (max-task-assign tasks workers pills strength)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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 maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int
strength) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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 maxTaskAssign(int[] tasks, int[] workers, int pills, int strength)
{

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Tasks You Can Assign
Difficulty: Hard
Tags: array, greedy, sort, search, queue

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 maxTaskAssign(self, tasks: List[int], workers: List[int], pills: int,
strength: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxTaskAssign(self, tasks, workers, pills, strength):
"""
:type tasks: List[int]
:type workers: List[int]
:type pills: int
:type strength: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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
* @param {number[]} workers
* @param {number} pills
* @param {number} strength
* @return {number}
*/
var maxTaskAssign = function(tasks, workers, pills, strength) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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 maxTaskAssign(tasks: number[], workers: number[], pills: number,
strength: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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 MaxTaskAssign(int[] tasks, int[] workers, int pills, int strength)
{

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Tasks You Can Assign
* Difficulty: Hard
* Tags: array, greedy, sort, search, queue
*
* 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 maxTaskAssign(int* tasks, int tasksSize, int* workers, int workersSize,
int pills, int strength) {

}
```

Go Solution:

```
// Problem: Maximum Number of Tasks You Can Assign
// Difficulty: Hard
// Tags: array, greedy, sort, search, queue
//
// 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 maxTaskAssign(tasks []int, workers []int, pills int, strength int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxTaskAssign(tasks: IntArray, workers: IntArray, pills: Int, strength:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxTaskAssign(_ tasks: [Int], _ workers: [Int], _ pills: Int, _
strength: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Tasks You Can Assign
// Difficulty: Hard
// Tags: array, greedy, sort, search, queue
//
// 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 max_task_assign(tasks: Vec<i32>, workers: Vec<i32>, pills: i32,
strength: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} tasks
# @param {Integer[]} workers
# @param {Integer} pills
# @param {Integer} strength
# @return {Integer}
def max_task_assign(tasks, workers, pills, strength)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tasks
* @param Integer[] $workers
* @param Integer $pills
* @param Integer $strength
* @return Integer
*/
function maxTaskAssign($tasks, $workers, $pills, $strength) {

}
}
```

Dart Solution:

```
class Solution {
int maxTaskAssign(List<int> tasks, List<int> workers, int pills, int
strength) {

}
}
```

Scala Solution:

```
object Solution {
def maxTaskAssign(tasks: Array[Int], workers: Array[Int], pills: Int,
strength: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_task_assign(tasks :: [integer], workers :: [integer], pills ::
integer, strength :: integer) :: integer
def max_task_assign(tasks, workers, pills, strength) do

end
end
```

Erlang Solution:

```
-spec max_task_assign(Tasks :: [integer()], Workers :: [integer()], Pills ::
integer(), Strength :: integer()) -> integer().
max_task_assign(Tasks, Workers, Pills, Strength) ->
.
```

Racket Solution:

```
(define/contract (max-task-assign tasks workers pills strength)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer? exact-integer?)
)
```