Problem 2323: Find Minimum Time to Finish All Jobs II

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

jobs

and

workers

of

equal

length, where

jobs[i]

is the amount of time needed to complete the

i

th

job, and

workers[j]

is the amount of time the

j

th

worker can work each day.

Each job should be assigned to

exactly

one worker, such that each worker completes

exactly

one job.

Return

the

minimum

number of days needed to complete all the jobs after assignment.

Example 1:

Input:

jobs = [5,2,4], workers = [1,7,5]

Output:

2

Explanation:

- Assign the 2

nd

worker to the 0

th

job. It takes them 1 day to finish the job. - Assign the 0

th

worker to the 1

st

job. It takes them 2 days to finish the job. - Assign the 1

st

worker to the 2

nd

job. It takes them 1 day to finish the job. It takes 2 days for all the jobs to be completed, so return 2. It can be proven that 2 days is the minimum number of days needed.

Example 2:

Input:

jobs = [3,18,15,9], workers = [6,5,1,3]

Output:

3

Explanation:

- Assign the 2

nd

worker to the 0

th

job. It takes them 3 days to finish the job. - Assign the 0

th

worker to the 1

st

job. It takes them 3 days to finish the job. - Assign the 1

st

worker to the 2

nd

job. It takes them 3 days to finish the job. - Assign the 3

rd

worker to the 3

rd

job. It takes them 3 days to finish the job. It takes 3 days for all the jobs to be completed, so return 3. It can be proven that 3 days is the minimum number of days needed.

Constraints:

n == jobs.length == workers.length

1 <= n <= 10

5

1 <= jobs[i], workers[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minimumTime(vector<int>& jobs, vector<int>& workers) {

}
};
```

Java:

```
class Solution {
public int minimumTime(int[] jobs, int[] workers) {

}
}
```

Python3:

```
class Solution:
def minimumTime(self, jobs: List[int], workers: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumTime(self, jobs, workers):
```

```
"""
:type jobs: List[int]
:type workers: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} jobs
* @param {number[]} workers
* @return {number}
*/
var minimumTime = function(jobs, workers) {

};
```

TypeScript:

```
function minimumTime(jobs: number[], workers: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumTime(int[] jobs, int[] workers) {

}
}
```

C:

```
int minimumTime(int* jobs, int jobsSize, int* workers, int workersSize) {

}
```

Go:

```
func minimumTime(jobs []int, workers []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumTime(jobs: IntArray, workers: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumTime(_ jobs: [Int], _ workers: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_time(jobs: Vec<i32>, workers: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} jobs
# @param {Integer[]} workers
# @return {Integer}
def minimum_time(jobs, workers)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $jobs
* @param Integer[] $workers
* @return Integer
*/
function minimumTime($jobs, $workers) {
```

```
}
}
```

Dart:

```
class Solution {
int minimumTime(List<int> jobs, List<int> workers) {

}
}
```

Scala:

```
object Solution {
def minimumTime(jobs: Array[Int], workers: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time(jobs :: [integer], workers :: [integer]) :: integer
def minimum_time(jobs, workers) do

end
end
```

Erlang:

```
-spec minimum_time(Jobs :: [integer()], Workers :: [integer()]) -> integer().
minimum_time(Jobs, Workers) ->
.
```

Racket:

```
(define/contract (minimum-time jobs workers)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs II
* Difficulty: Medium
* 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 minimumTime(vector<int>& jobs, vector<int>& workers) {

}
};
```

Java Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs II
* Difficulty: Medium
* 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 minimumTime(int[] jobs, int[] workers) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Minimum Time to Finish All Jobs II
```

```
Difficulty: Medium
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 minimumTime(self, jobs: List[int], workers: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumTime(self, jobs, workers):
"""
:type jobs: List[int]
:type workers: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs II
* Difficulty: Medium
* 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[]} jobs
* @param {number[]} workers
* @return {number}
*/
var minimumTime = function(jobs, workers) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Find Minimum Time to Finish All Jobs II
* Difficulty: Medium
* 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 minimumTime(jobs: number[], workers: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs II
* Difficulty: Medium
* 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 MinimumTime(int[] jobs, int[] workers) {

}
}
```

C Solution:

```
/*
* Problem: Find Minimum Time to Finish All Jobs II
```

```
* Difficulty: Medium
* 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 minimumTime(int* jobs, int jobsSize, int* workers, int workersSize) {

}
```

Go Solution:

```
// Problem: Find Minimum Time to Finish All Jobs II
// Difficulty: Medium
// 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 minimumTime(jobs []int, workers []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTime(jobs: IntArray, workers: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumTime(_ jobs: [Int], _ workers: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Minimum Time to Finish All Jobs II
// Difficulty: Medium
// 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_time(jobs: Vec<i32>, workers: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} jobs
# @param {Integer[]} workers
# @return {Integer}
def minimum_time(jobs, workers)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $jobs
* @param Integer[] $workers
* @return Integer
*/
function minimumTime($jobs, $workers) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTime(List<int> jobs, List<int> workers) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTime(jobs: Array[Int], workers: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time(jobs :: [integer], workers :: [integer]) :: integer
def minimum_time(jobs, workers) do

end
end
```

Erlang Solution:

```
-spec minimum_time(Jobs :: [integer()], Workers :: [integer()]) -> integer().
minimum_time(Jobs, Workers) ->
.
```

Racket Solution:

```
(define/contract (minimum-time jobs workers)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```