Problem 2589: Minimum Time to Complete All Tasks

Problem Information

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

Problem Description

There is a computer that can run an unlimited number of tasks

at the same time

. You are given a 2D integer array

tasks

where

tasks[i] = [start

i

, end

i

, duration

i

]

indicates that the

i

th

task should run for a total of

duration

i

seconds (not necessarily continuous) within the

inclusive

time range

[start

i

, end

i

]

.

You may turn on the computer only when it needs to run a task. You can also turn it off if it is idle.

Return

the minimum time during which the computer should be turned on to complete all tasks

.

Example 1:

Input:

tasks = [[2,3,1],[4,5,1],[1,5,2]]

Output:

2

Explanation:

- The first task can be run in the inclusive time range [2, 2]. - The second task can be run in the inclusive time range [5, 5]. - The third task can be run in the two inclusive time ranges [2, 2] and [5, 5]. The computer will be on for a total of 2 seconds.

Example 2:

Input:

tasks = [[1,3,2],[2,5,3],[5,6,2]]

Output:

4

Explanation:

- The first task can be run in the inclusive time range [2, 3]. - The second task can be run in the inclusive time ranges [2, 3] and [5, 5]. - The third task can be run in the two inclusive time range [5, 6]. The computer will be on for a total of 4 seconds.

Constraints:

1 <= tasks.length <= 2000

tasks[i].length == 3

1 <= start

i

, end

i

<= 2000

1 <= duration

i

<= end

i

- start

i

+ 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 findMinimumTime(vector<vector<int>>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 findMinimumTime(int[][] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Complete All Tasks
Difficulty: Hard
Tags: array, greedy, sort, search, stack

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 findMinimumTime(self, tasks: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 findMinimumTime = function(tasks) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 findMinimumTime(tasks: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 FindMinimumTime(int[][] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Complete All Tasks
* Difficulty: Hard
* Tags: array, greedy, sort, search, stack
*
* 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 findMinimumTime(int** tasks, int tasksSize, int* tasksColSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Complete All Tasks
// Difficulty: Hard
// Tags: array, greedy, sort, search, stack
//
// 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 findMinimumTime(tasks [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Minimum Time to Complete All Tasks
// Difficulty: Hard
// Tags: array, greedy, sort, search, stack
//
// 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 find_minimum_time(tasks: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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