Problem 3683: Earliest Time to Finish One Task

Problem Information

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

Problem Description

You are given a 2D integer array

tasks

where

tasks[i] = [s

i

, t

i

]

.

Each

[s

i

, t

i

]

in

tasks

represents a task with start time

s

i

that takes

t

i

units of time to finish.

Return the earliest time at which at least one task is finished.

Example 1:

Input:

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

Output:

5

Explanation:

The first task starts at time

t = 1

and finishes at time

1 + 6 = 7

. The second task finishes at time

2 + 3 = 5

. You can finish one task at time 5.

Example 2:

Input:

tasks = [[100,100],[100,100],[100,100]]

Output:

200

Explanation:

All three tasks finish at time

100 + 100 = 200

.

Constraints:

1 <= tasks.length <= 100

tasks[i] = [s

i

, t

i

]

1 <= s

i

, t

i

<= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 earliestTime(vector<vector<int>>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 earliestTime(int[][] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Earliest Time to Finish One Task
Difficulty: Easy
Tags: array
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 earliestTime = function(tasks) {

};
```

TypeScript Solution:

```
/**
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 earliestTime(tasks: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 EarliestTime(int[][] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Earliest Time to Finish One Task
* Difficulty: Easy
* Tags: array
*
* 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 earliestTime(int** tasks, int tasksSize, int* tasksColSize) {

}
```

Go Solution:

```
// Problem: Earliest Time to Finish One Task
// Difficulty: Easy
// Tags: array
//
// 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 earliestTime(tasks [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Earliest Time to Finish One Task
// Difficulty: Easy
// Tags: array
```

```
//
// 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 earliest_time(tasks: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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