Problem 2432: The Employee That Worked on the Longest Task

Problem Information

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

Problem Description

There are

n

employees, each with a unique id from

0

to

n - 1

.

You are given a 2D integer array

logs

where

logs[i] = [id

i

, leaveTime

i

]

where:

id

i

is the id of the employee that worked on the

i

th

task, and

leaveTime

i

is the time at which the employee finished the

i

th

task. All the values

leaveTime

i

are

unique

.

Note that the

i

th

task starts the moment right after the

(i - 1)

th

task ends, and the

0

th

task starts at time

0

.

Return

the id of the employee that worked the task with the longest time.

If there is a tie between two or more employees, return

the

smallest

id among them

.

Example 1:

Input:

n = 10, logs = [[0,3],[2,5],[0,9],[1,15]]

Output:

1

Explanation:

Task 0 started at 0 and ended at 3 with 3 units of times. Task 1 started at 3 and ended at 5 with 2 units of times. Task 2 started at 5 and ended at 9 with 4 units of times. Task 3 started at 9 and ended at 15 with 6 units of times. The task with the longest time is task 3 and the employee with id 1 is the one that worked on it, so we return 1.

Example 2:

Input:

n = 26, logs = [[1,1],[3,7],[2,12],[7,17]]

Output:

3

Explanation:

Task 0 started at 0 and ended at 1 with 1 unit of times. Task 1 started at 1 and ended at 7 with 6 units of times. Task 2 started at 7 and ended at 12 with 5 units of times. Task 3 started at 12 and ended at 17 with 5 units of times. The tasks with the longest time is task 1. The employee that worked on it is 3, so we return 3.

Example 3:

Input:

n = 2, logs = [[0,10],[1,20]]

Output:

0

Explanation:

Task 0 started at 0 and ended at 10 with 10 units of times. Task 1 started at 10 and ended at 20 with 10 units of times. The tasks with the longest time are tasks 0 and 1. The employees that worked on them are 0 and 1, so we return the smallest id 0.

Constraints:

2 <= n <= 500

1 <= logs.length <= 500

logs[i].length == 2

0 <= id

i

<= n - 1

1 <= leaveTime

i

<= 500

id

i

!= id

i+1

leaveTime

i

are sorted in a strictly increasing order.

Code Snippets

C++:

```
class Solution {
public:
int hardestWorker(int n, vector<vector<int>>& logs) {

}
};
```

Java:

```
class Solution {
public int hardestWorker(int n, int[][] logs) {

}
}
```

Python3:

```
class Solution:
def hardestWorker(self, n: int, logs: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def hardestWorker(self, n, logs):
"""
:type n: int
:type logs: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} logs
* @return {number}
*/
var hardestWorker = function(n, logs) {

};
```

TypeScript:

```
function hardestWorker(n: number, logs: number[][]): number {

};
```

C#:

```
public class Solution {
public int HardestWorker(int n, int[][] logs) {

}
}
```

C:

```
int hardestWorker(int n, int** logs, int logsSize, int* logsColSize) {

}
```

Go:

```
func hardestWorker(n int, logs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun hardestWorker(n: Int, logs: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func hardestWorker(_ n: Int, _ logs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn hardest_worker(n: i32, logs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} logs
# @return {Integer}
def hardest_worker(n, logs)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $logs
* @return Integer
*/
function hardestWorker($n, $logs) {

}
}
```

Dart:

```
class Solution {
int hardestWorker(int n, List<List<int>> logs) {
```

```
}
}
```

Scala:

```
object Solution {
def hardestWorker(n: Int, logs: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec hardest_worker(n :: integer, logs :: [[integer]]) :: integer
def hardest_worker(n, logs) do

end
end
```

Erlang:

```
-spec hardest_worker(N :: integer(), Logs :: [[integer()]]) -> integer().
hardest_worker(N, Logs) ->
.
```

Racket:

```
(define/contract (hardest-worker n logs)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
```

```
* Tags: array, 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 hardestWorker(int n, vector<vector<int>>& logs) {

}
};
```

Java Solution:

```
/**
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
* Tags: array, 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 hardestWorker(int n, int[][] logs) {

}
}
```

Python3 Solution:

```
"""
Problem: The Employee That Worked on the Longest Task
Difficulty: Easy
Tags: array, 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 hardestWorker(self, n: int, logs: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def hardestWorker(self, n, logs):
"""
:type n: int
:type logs: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
* Tags: array, 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} n
* @param {number[][]} logs
* @return {number}
*/
var hardestWorker = function(n, logs) {

};
```

TypeScript Solution:

```
/**
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
* Tags: array, 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 hardestWorker(n: number, logs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
* Tags: array, 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 HardestWorker(int n, int[][] logs) {

}
}
```

C Solution:

```
/*
* Problem: The Employee That Worked on the Longest Task
* Difficulty: Easy
* Tags: array, 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 hardestWorker(int n, int** logs, int logsSize, int* logsColSize) {

}
```

Go Solution:

```
// Problem: The Employee That Worked on the Longest Task
// Difficulty: Easy
// Tags: array, 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 hardestWorker(n int, logs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun hardestWorker(n: Int, logs: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func hardestWorker(_ n: Int, _ logs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: The Employee That Worked on the Longest Task
// Difficulty: Easy
// Tags: array, 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 hardest_worker(n: i32, logs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} logs
# @return {Integer}
def hardest_worker(n, logs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $logs
* @return Integer
*/
function hardestWorker($n, $logs) {

}
}
```

Dart Solution:

```
class Solution {
int hardestWorker(int n, List<List<int>> logs) {

}
}
```

Scala Solution:

```
object Solution {
def hardestWorker(n: Int, logs: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec hardest_worker(n :: integer, logs :: [[integer]]) :: integer
def hardest_worker(n, logs) do

end
end
```

Erlang Solution:

```
-spec hardest_worker(N :: integer(), Logs :: [[integer()]]) -> integer().
hardest_worker(N, Logs) ->
.
```

Racket Solution:

```
(define/contract (hardest-worker n logs)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```