Problem 3386: Button with Longest Push Time

Problem Information

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

Problem Description

You are given a 2D array

events

which represents a sequence of events where a child pushes a series of buttons on a keyboard.

Each

events[i] = [index

i

, time

i

]

indicates that the button at index

index

i

was pressed at time

time

i

.

The array is

sorted

in increasing order of

time

.

The time taken to press a button is the difference in time between consecutive button presses. The time for the first button is simply the time at which it was pressed.

Return the

index

of the button that took the

longest

time to push. If multiple buttons have the same longest time, return the button with the

smallest

index

.

Example 1:

Input:

events = [[1,2],[2,5],[3,9],[1,15]]

Output:

1

Explanation:

Button with index 1 is pressed at time 2.

Button with index 2 is pressed at time 5, so it took

5 - 2 = 3

units of time.

Button with index 3 is pressed at time 9, so it took

9 - 5 = 4

units of time.

Button with index 1 is pressed again at time 15, so it took

15 - 9 = 6

units of time.

Example 2:

Input:

events = [[10,5],[1,7]]

Output:

10

Explanation:

Button with index 10 is pressed at time 5.

Button with index 1 is pressed at time 7, so it took

7 - 5 = 2

units of time.

Constraints:

1 <= events.length <= 1000

events[i] == [index

i

, time

i

]

1 <= index

i

, time

i

<= 10

5

The input is generated such that

events

is sorted in increasing order of

time

i

.

Code Snippets

C++:

```
class Solution {
public:
int buttonWithLongestTime(vector<vector<int>>& events) {

}
};
```

Java:

```
class Solution {
public int buttonWithLongestTime(int[][] events) {

}
}
```

Python3:

```
class Solution:
def buttonWithLongestTime(self, events: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function buttonWithLongestTime(events: number[][]): number {

};
```

C#:

```
public class Solution {
public int ButtonWithLongestTime(int[][] events) {

}
}
```

C:

```
int buttonWithLongestTime(int** events, int eventsSize, int* eventsColSize) {

}
```

Go:

```
func buttonWithLongestTime(events [][]int) int {

}
```

Kotlin:

```
class Solution {
fun buttonWithLongestTime(events: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func buttonWithLongestTime(_ events: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn button_with_longest_time(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} events
# @return {Integer}
def button_with_longest_time(events)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $events
* @return Integer
*/
function buttonWithLongestTime($events) {

}
}
```

Dart:

```
class Solution {
int buttonWithLongestTime(List<List<int>> events) {

}
}
```

Scala:

```
object Solution {
def buttonWithLongestTime(events: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec button_with_longest_time(events :: [[integer]]) :: integer
def button_with_longest_time(events) do

end
end
```

Erlang:

```
-spec button_with_longest_time(Events :: [[integer()]]) -> integer().
button_with_longest_time(Events) ->
.
```

Racket:

```
(define/contract (button-with-longest-time events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Button with Longest Push Time
* 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 buttonWithLongestTime(vector<vector<int>>& events) {

}
};
```

Java Solution:

```
/**
* Problem: Button with Longest Push Time
* 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 buttonWithLongestTime(int[][] events) {

}
}
```

Python3 Solution:

```
"""
Problem: Button with Longest Push Time
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 buttonWithLongestTime(self, events: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Button with Longest Push Time
* 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[][]} events
* @return {number}
*/
var buttonWithLongestTime = function(events) {

};
```

TypeScript Solution:

```
/**
* Problem: Button with Longest Push Time
* 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 buttonWithLongestTime(events: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Button with Longest Push Time
* 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 ButtonWithLongestTime(int[][] events) {

}
}
```

C Solution:

```
/*
* Problem: Button with Longest Push Time
* 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 buttonWithLongestTime(int** events, int eventsSize, int* eventsColSize) {
}
```

Go Solution:

```go
// Problem: Button with Longest Push Time
// 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 buttonWithLongestTime(events [][]int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun buttonWithLongestTime(events: Array<IntArray>): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func buttonWithLongestTime(_ events: [[Int]]) -> Int {

}
}
```

Rust Solution:

```rust
// Problem: Button with Longest Push Time
// 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 button_with_longest_time(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} events
# @return {Integer}
def button_with_longest_time(events)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $events
* @return Integer
*/
function buttonWithLongestTime($events) {

}
}
```

Dart Solution:

```
class Solution {
int buttonWithLongestTime(List<List<int>> events) {

}
}
```

Scala Solution:

```
object Solution {
def buttonWithLongestTime(events: Array[Array[Int]]): Int = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec button_with_longest_time(events :: [[integer]]) :: integer
def button_with_longest_time(events) do

end
end
```

Erlang Solution:

```
-spec button_with_longest_time(Events :: [[integer()]]) -> integer().
button_with_longest_time(Events) ->
.
```

Racket Solution:

```
(define/contract (button-with-longest-time events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```