Problem 1854: Maximum Population Year

Problem Information

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

Problem Description

You are given a 2D integer array

logs

where each

logs[i] = [birth

i

, death

i

]

indicates the birth and death years of the

i

th

person.

The

population of some year x is the number of people alive during that year. The i^{th} person is counted in year x's population if x is in the inclusive range $[birth_i, death_i - 1]$. Note that the person is not counted in the year that they die.

Return

the

earliest

year with the

maximum population

.

Example 1:

Input:

logs = [[1993,1999],[2000,2010]]

Output:

1993

Explanation:

The maximum population is 1, and 1993 is the earliest year with this population.

Example 2:

Input:

logs = [[1950,1961],[1960,1971],[1970,1981]]

Output:

1960

Explanation:

The maximum population is 2, and it had happened in years 1960 and 1970. The earlier year between them is 1960.

Constraints:

1 <= logs.length <= 100

1950 <= birth

i

< death

i

<= 2050

Code Snippets

C++:

```
class Solution {
public:
int maximumPopulation(vector<vector<int>>& logs) {

}
};
```

Java:

```
class Solution {
public int maximumPopulation(int[][] logs) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumPopulation(logs: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximumPopulation(int[][] logs) {

}
}
```

C:

```
int maximumPopulation(int** logs, int logsSize, int* logsColSize) {

}
```

Go:

```
func maximumPopulation(logs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximumPopulation(logs: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maximumPopulation(_ logs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_population(logs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} logs
# @return {Integer}
def maximum_population(logs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $logs
* @return Integer
*/
function maximumPopulation($logs) {

}
}
```

Dart:

```
class Solution {
int maximumPopulation(List<List<int>> logs) {

}
}
```

Scala:

```
object Solution {
def maximumPopulation(logs: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_population(logs :: [[integer]]) :: integer
def maximum_population(logs) do

end
end
```

Erlang:

```
-spec maximum_population(Logs :: [[integer()]]) -> integer().
maximum_population(Logs) ->
.
```

Racket:

```
(define/contract (maximum-population logs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Population Year
* 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 maximumPopulation(vector<vector<int>>& logs) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Population Year
* 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 maximumPopulation(int[][] logs) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Population Year
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 maximumPopulation(self, logs: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Population Year
* 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[][]} logs
* @return {number}
*/
var maximumPopulation = function(logs) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Population Year
* 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 maximumPopulation(logs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Population Year
* 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 MaximumPopulation(int[][] logs) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Population Year
* 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 maximumPopulation(int** logs, int logsSize, int* logsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Population Year
// 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 maximumPopulation(logs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumPopulation(logs: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumPopulation(_ logs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Population Year
// 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 maximum_population(logs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} logs
# @return {Integer}
def maximum_population(logs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $logs
* @return Integer
*/
function maximumPopulation($logs) {

}
}
```

Dart Solution:

```
class Solution {
int maximumPopulation(List<List<int>> logs) {
}
}
```

Scala Solution:

```
object Solution {
def maximumPopulation(logs: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_population(logs :: [[integer]]) :: integer
def maximum_population(logs) do

end
end
```

Erlang Solution:

```
-spec maximum_population(Logs :: [[integer()]]) -> integer().
maximum_population(Logs) ->
.
```

Racket Solution:

```
(define/contract (maximum-population logs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```