Problem 1851: Minimum Interval to Include Each Query

Problem Information

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

Problem Description

You are given a 2D integer array

intervals

, where

intervals[i] = [left

i

, right

i

]

describes the

i

th

interval starting at

left

i

and ending at

right

i

(inclusive)

. The

size

of an interval is defined as the number of integers it contains, or more formally

right

i

- left

i

+ 1

.

You are also given an integer array

queries

. The answer to the

j

th

query is the

size of the smallest interval

i

such that

left

i

<= queries[j] <= right

i

. If no such interval exists, the answer is

-1

.

Return

an array containing the answers to the queries

.

Example 1:

Input:

intervals = [[1,4],[2,4],[3,6],[4,4]], queries = [2,3,4,5]

Output:

[3,3,1,4]

Explanation:

The queries are processed as follows: - Query = 2: The interval [2,4] is the smallest interval containing 2. The answer is 4 - 2 + 1 = 3. - Query = 3: The interval [2,4] is the smallest interval containing 3. The answer is 4 - 2 + 1 = 3. - Query = 4: The interval [4,4] is the smallest interval containing 4. The answer is 4 - 4 + 1 = 1. - Query = 5: The interval [3,6] is the smallest interval containing 5. The answer is 6 - 3 + 1 = 4.

Example 2:

Input:

intervals = [[2,3],[2,5],[1,8],[20,25]], queries = [2,19,5,22]

Output:

[2,-1,4,6]

Explanation:

The queries are processed as follows: - Query = 2: The interval [2,3] is the smallest interval containing 2. The answer is 3 - 2 + 1 = 2. - Query = 19: None of the intervals contain 19. The answer is -1. - Query = 5: The interval [2,5] is the smallest interval containing 5. The answer is 5 - 2 + 1 = 4. - Query = 22: The interval [20,25] is the smallest interval containing 22. The answer is 25 - 20 + 1 = 6.

Constraints:

1 <= intervals.length <= 10

5

1 <= queries.length <= 10

5

intervals[i].length == 2

1 <= left

i

<= right

i

<= 10

7

1 <= queries[j] <= 10

7

Code Snippets

C++:

```
class Solution {
public:
vector<int> minInterval(vector<vector<int>>& intervals, vector<int>& queries)
{

}
};
```

Java:

```
class Solution {
public int[] minInterval(int[][] intervals, int[] queries) {

}
}
```

Python3:

```
class Solution:
def minInterval(self, intervals: List[List[int]], queries: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
def minInterval(self, intervals, queries):
"""
:type intervals: List[List[int]]
:type queries: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} intervals
* @param {number[]} queries
* @return {number[]}
*/
var minInterval = function(intervals, queries) {

};
```

TypeScript:

```
function minInterval(intervals: number[][], queries: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] MinInterval(int[][] intervals, int[] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minInterval(int** intervals, int intervalsSize, int* intervalsColSize,
int* queries, int queriesSize, int* returnSize) {

}
```

Go:

```
func minInterval(intervals [][]int, queries []int) []int {

}
```

Kotlin:

```
class Solution {
fun minInterval(intervals: Array<IntArray>, queries: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func minInterval(_ intervals: [[Int]], _ queries: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn min_interval(intervals: Vec<Vec<i32>>, queries: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @param {Integer[]} queries
# @return {Integer[]}
def min_interval(intervals, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer[] $queries
* @return Integer[]
*/
function minInterval($intervals, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> minInterval(List<List<int>> intervals, List<int> queries) {

}
}
```

Scala:

```
object Solution {
def minInterval(intervals: Array[Array[Int]], queries: Array[Int]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_interval(intervals :: [[integer]], queries :: [integer]) ::
[integer]
def min_interval(intervals, queries) do

end
end
```

Erlang:

```
-spec min_interval(Intervals :: [[integer()]], Queries :: [integer()]) ->
[integer()].
```

```
min_interval(Intervals, Queries) ->
.
```

Racket:

```
(define/contract (min-interval intervals queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
* 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:
vector<int> minInterval(vector<vector<int>>& intervals, vector<int>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
* 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[] minInterval(int[][] intervals, int[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Interval to Include Each Query
Difficulty: Hard
Tags: array, sort, search, queue, heap

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 minInterval(self, intervals: List[List[int]], queries: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minInterval(self, intervals, queries):
"""
:type intervals: List[List[int]]
:type queries: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
* 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[][]} intervals
* @param {number[]} queries
* @return {number[]}
*/
var minInterval = function(intervals, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
* 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 minInterval(intervals: number[][], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
```

```
* 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[] MinInterval(int[][] intervals, int[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Interval to Include Each Query
* Difficulty: Hard
* Tags: array, sort, search, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minInterval(int** intervals, int intervalsSize, int* intervalsColSize,
int* queries, int queriesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Interval to Include Each Query
// Difficulty: Hard
// Tags: array, sort, search, queue, heap
//
// 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 minInterval(intervals [][]int, queries []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun minInterval(intervals: Array<IntArray>, queries: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func minInterval(_ intervals: [[Int]], _ queries: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Minimum Interval to Include Each Query
// Difficulty: Hard
// Tags: array, sort, search, queue, heap
//
// 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 min_interval(intervals: Vec<Vec<i32>>, queries: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @param {Integer[]} queries
# @return {Integer[]}
def min_interval(intervals, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer[] $queries
* @return Integer[]
*/
function minInterval($intervals, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minInterval(List<List<int>> intervals, List<int> queries) {

}
}
```

Scala Solution:

```
object Solution {
def minInterval(intervals: Array[Array[Int]], queries: Array[Int]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_interval(intervals :: [[integer]], queries :: [integer]) ::
[integer]
```

```
def min_interval(intervals, queries) do

end
end
```

Erlang Solution:

```
-spec min_interval(Intervals :: [[integer()]], Queries :: [integer()]) ->
[integer()].
min_interval(Intervals, Queries) ->
.
```

Racket Solution:

```
(define/contract (min-interval intervals queries)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```