

Problem 1229: Meeting Scheduler

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the availability time slots arrays

slots1

and

slots2

of two people and a meeting duration

duration

, return the

earliest time slot

that works for both of them and is of duration

duration

.

If there is no common time slot that satisfies the requirements, return an

empty array

.

The format of a time slot is an array of two elements

[start, end]

representing an inclusive time range from

start

to

end

.

It is guaranteed that no two availability slots of the same person intersect with each other.
That is, for any two time slots

[start1, end1]

and

[start2, end2]

of the same person, either

$\text{start1} > \text{end2}$

or

$\text{start2} > \text{end1}$

.

Example 1:

Input:

```
slots1 = [[10,50],[60,120],[140,210]], slots2 = [[0,15],[60,70]], duration = 8
```

Output:

```
[60,68]
```

Example 2:

Input:

```
slots1 = [[10,50],[60,120],[140,210]], slots2 = [[0,15],[60,70]], duration = 12
```

Output:

```
[]
```

Constraints:

```
1 <= slots1.length, slots2.length <= 10
```

```
4
```

```
slots1[i].length, slots2[i].length == 2
```

```
slots1[i][0] < slots1[i][1]
```

```
slots2[i][0] < slots2[i][1]
```

```
0 <= slots1[i][j], slots2[i][j] <= 10
```

```
9
```

```
1 <= duration <= 10
```

```
6
```

Code Snippets

C++:

```
class Solution {
public:
    vector<int> minAvailableDuration(vector<vector<int>>& slots1,
    vector<vector<int>>& slots2, int duration) {

    }
};
```

Java:

```
class Solution {
    public List<Integer> minAvailableDuration(int[][] slots1, int[][] slots2, int
    duration) {

    }
}
```

Python3:

```
class Solution:
    def minAvailableDuration(self, slots1: List[List[int]], slots2:
    List[List[int]], duration: int) -> List[int]:
```

Python:

```
class Solution(object):
    def minAvailableDuration(self, slots1, slots2, duration):
        """
        :type slots1: List[List[int]]
        :type slots2: List[List[int]]
        :type duration: int
        :rtype: List[int]
        """
```

JavaScript:

```
/**
 * @param {number[][]} slots1
 * @param {number[][]} slots2
 * @param {number} duration
 * @return {number[]}
```

```

*/
var minAvailableDuration = function(slots1, slots2, duration) {

};

```

TypeScript:

```

function minAvailableDuration(slots1: number[][], slots2: number[][],
duration: number): number[] {

};

```

C#:

```

public class Solution {
    public IList<int> MinAvailableDuration(int[][] slots1, int[][] slots2, int
duration) {

    }
}

```

C:

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* minAvailableDuration(int** slots1, int slots1Size, int* slots1ColSize,
int** slots2, int slots2Size, int* slots2ColSize, int duration, int*
returnSize) {

}

```

Go:

```

func minAvailableDuration(slots1 [][]int, slots2 [][]int, duration int) []int
{

}

```

Kotlin:

```

class Solution {
    fun minAvailableDuration(slots1: Array<IntArray>, slots2: Array<IntArray>,
        duration: Int): List<Int> {

    }
}

```

Swift:

```

class Solution {
    func minAvailableDuration(_ slots1: [[Int]], _ slots2: [[Int]], _ duration:
        Int) -> [Int] {

    }
}

```

Rust:

```

impl Solution {
    pub fn min_available_duration(slots1: Vec<Vec<i32>>, slots2: Vec<Vec<i32>>,
        duration: i32) -> Vec<i32> {

    }
}

```

Ruby:

```

# @param {Integer[][]} slots1
# @param {Integer[][]} slots2
# @param {Integer} duration
# @return {Integer[]}
def min_available_duration(slots1, slots2, duration)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[][] $slots1
     * @param Integer[][] $slots2
     * @param Integer $duration
     */
}

```

```

* @return Integer[]
*/
function minAvailableDuration($slots1, $slots2, $duration) {

}
}

```

Dart:

```

class Solution {
  List<int> minAvailableDuration(List<List<int>> slots1, List<List<int>>
  slots2, int duration) {

  }
}

```

Scala:

```

object Solution {
  def minAvailableDuration(slots1: Array[Array[Int]], slots2:
  Array[Array[Int]], duration: Int): List[Int] = {

  }
}

```

Elixir:

```

defmodule Solution do
  @spec min_available_duration(slots1 :: [[integer]], slots2 :: [[integer]],
  duration :: integer) :: [integer]
  def min_available_duration(slots1, slots2, duration) do

  end
end

```

Erlang:

```

-spec min_available_duration(Slots1 :: [[integer()]], Slots2 ::
[[integer()]], Duration :: integer()) -> [integer()].
min_available_duration(Slots1, Slots2, Duration) ->
.

```

Racket:

```
(define/contract (min-available-duration slots1 slots2 duration)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
      exact-integer? (listof exact-integer?))
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Meeting Scheduler
 * Difficulty: Medium
 * 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:
    vector<int> minAvailableDuration(vector<vector<int>>& slots1,
    vector<vector<int>>& slots2, int duration) {

    }

};
```

Java Solution:

```
/**
 * Problem: Meeting Scheduler
 * Difficulty: Medium
 * 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 List<Integer> minAvailableDuration(int[][] slots1, int[][] slots2, int
duration) {

}
}

```

Python3 Solution:

```

"""
Problem: Meeting Scheduler
Difficulty: Medium
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 minAvailableDuration(self, slots1: List[List[int]], slots2:
List[List[int]], duration: int) -> List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def minAvailableDuration(self, slots1, slots2, duration):
"""
:type slots1: List[List[int]]
:type slots2: List[List[int]]
:type duration: int
:rtype: List[int]
"""

```

JavaScript Solution:

```

/**
 * Problem: Meeting Scheduler
 * Difficulty: Medium

```

```

* 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[][]} slots1
* @param {number[][]} slots2
* @param {number} duration
* @return {number[]}
*/
var minAvailableDuration = function(slots1, slots2, duration) {

};

```

TypeScript Solution:

```

/**
* Problem: Meeting Scheduler
* Difficulty: Medium
* 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 minAvailableDuration(slots1: number[][], slots2: number[][],
duration: number): number[] {

};

```

C# Solution:

```

/*
* Problem: Meeting Scheduler
* Difficulty: Medium
* 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 IList<int> MinAvailableDuration(int[][] slots1, int[][] slots2, int
duration) {

}

}

```

C Solution:

```

/*
* Problem: Meeting Scheduler
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minAvailableDuration(int** slots1, int slots1Size, int* slots1ColSize,
int** slots2, int slots2Size, int* slots2ColSize, int duration, int*
returnSize) {

}

```

Go Solution:

```

// Problem: Meeting Scheduler
// Difficulty: Medium
// 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 minAvailableDuration(slots1 [[[int, slots2 [[[int, duration int) []int
{

}

}
```

Kotlin Solution:

```
class Solution {
    fun minAvailableDuration(slots1: Array<IntArray>, slots2: Array<IntArray>,
        duration: Int): List<Int> {

    }
}
```

Swift Solution:

```
class Solution {
    func minAvailableDuration(_ slots1: [[Int]], _ slots2: [[Int]], _ duration:
        Int) -> [Int] {

    }
}
```

Rust Solution:

```
// Problem: Meeting Scheduler
// Difficulty: Medium
// 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 min_available_duration(slots1: Vec<Vec<i32>>, slots2: Vec<Vec<i32>>,
        duration: i32) -> Vec<i32> {

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} slots1
# @param {Integer[][]} slots2
# @param {Integer} duration
# @return {Integer[]}
def min_available_duration(slots1, slots2, duration)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $slots1
     * @param Integer[][] $slots2
     * @param Integer $duration
     * @return Integer[]
     */
    function minAvailableDuration($slots1, $slots2, $duration) {

    }

}
```

Dart Solution:

```
class Solution {
  List<int> minAvailableDuration(List<List<int>> slots1, List<List<int>>
    slots2, int duration) {

  }

}
```

Scala Solution:

```
object Solution {
  def minAvailableDuration(slots1: Array[Array[Int]], slots2:
    Array[Array[Int]], duration: Int): List[Int] = {

  }

}
```

Elixir Solution:

```
defmodule Solution do
  @spec min_available_duration(slots1 :: [[integer]], slots2 :: [[integer]],
    duration :: integer) :: [integer]
  def min_available_duration(slots1, slots2, duration) do

  end
end
```

Erlang Solution:

```
-spec min_available_duration(Slots1 :: [[integer()]], Slots2 ::
[[integer()]], Duration :: integer()) -> [integer()].
min_available_duration(Slots1, Slots2, Duration) ->
.
```

Racket Solution:

```
(define/contract (min-available-duration slots1 slots2 duration)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
    exact-integer? (listof exact-integer?))
  )
```