# Problem 252: Meeting Rooms

## Problem Information

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

## Problem Description

Given an array of meeting time

intervals

where

intervals[i] = [start

i

, end

i

]

, determine if a person could attend all meetings.

Example 1:

Input:

intervals = [[0,30],[5,10],[15,20]]

Output:

false

Example 2:

Input:

intervals = [[7,10],[2,4]]

Output:

true

Constraints:

0 <= intervals.length <= 10

4

intervals[i].length == 2

0 <= start

i

< end

i

<= 10

6

## Code Snippets

**C++:**

```
class Solution {
public:
```

```
bool canAttendMeetings(vector<vector<int>>& intervals) {

}
};
```

**Java:**

```
class Solution {
public boolean canAttendMeetings(int[][] intervals) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

```
/**
* @param {number[][]} intervals
* @return {boolean}
*/
var canAttendMeetings = function(intervals) {

};
```

**TypeScript:**

```
function canAttendMeetings(intervals: number[][]): boolean {

};
```

**C#:**

```
public class Solution {
public bool CanAttendMeetings(int[][] intervals) {

}
}
```

**C:**

```
bool canAttendMeetings(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

**Go:**

```
func canAttendMeetings(intervals [][]int) bool {

}
```

**Kotlin:**

```
class Solution {
fun canAttendMeetings(intervals: Array<IntArray>): Boolean {

}
}
```

**Swift:**

```
class Solution {
func canAttendMeetings(_ intervals: [[Int]]) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn can_attend_meetings(intervals: Vec<Vec<i32>>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[][]} intervals
# @return {Boolean}
def can_attend_meetings(intervals)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Boolean
*/
function canAttendMeetings($intervals) {

}
}
```

**Dart:**

```
class Solution {
bool canAttendMeetings(List<List<int>> intervals) {

}
}
```

**Scala:**

```
object Solution {
def canAttendMeetings(intervals: Array[Array[Int]]): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec can_attend_meetings(intervals :: [[integer]]) :: boolean
def can_attend_meetings(intervals) do

end
end
```

### Erlang:

```
-spec can_attend_meetings(Intervals :: [[integer()]]) -> boolean().
can_attend_meetings(Intervals) ->
.
```

### Racket:

```
(define/contract (can-attend-meetings intervals)
(-> (listof (listof exact-integer?)) boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Meeting Rooms
* 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:
bool canAttendMeetings(vector<vector<int>>& intervals) {

}
};
```

### Java Solution:

```
/**
* Problem: Meeting Rooms
* 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 boolean canAttendMeetings(int[][] intervals) {

}
}
```

### Python3 Solution:

```
"""
Problem: Meeting Rooms
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 canAttendMeetings(self, intervals: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Meeting Rooms
* 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[][]} intervals
* @return {boolean}
*/
var canAttendMeetings = function(intervals) {

};
```

### TypeScript Solution:

```
/**
* Problem: Meeting Rooms
* 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 canAttendMeetings(intervals: number[][]): boolean {

};
```

### C# Solution:

```
/*
* Problem: Meeting Rooms
* 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 bool CanAttendMeetings(int[][] intervals) {

}
}
```

### C Solution:

```
/*
* Problem: Meeting Rooms
* 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
*/

bool canAttendMeetings(int** intervals, int intervalsSize, int*
intervalsColSize) {

}
```

### Go Solution:

```
// Problem: Meeting Rooms
// 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 canAttendMeetings(intervals [][]int) bool {

}
```

### Kotlin Solution:

```
class Solution {
fun canAttendMeetings(intervals: Array<IntArray>): Boolean {

}
}
```

### Swift Solution:

```
class Solution {
func canAttendMeetings(_ intervals: [[Int]]) -> Bool {

}
}
```

### Rust Solution:

```
// Problem: Meeting Rooms
// 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 can_attend_meetings(intervals: Vec<Vec<i32>>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} intervals
# @return {Boolean}
def can_attend_meetings(intervals)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @return Boolean
*/
function canAttendMeetings($intervals) {

}
}
```

### Dart Solution:

```
class Solution {
bool canAttendMeetings(List<List<int>> intervals) {

}
}
```

### Scala Solution:

```
object Solution {
def canAttendMeetings(intervals: Array[Array[Int]]): Boolean = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec can_attend_meetings(intervals :: [[integer]]) :: boolean
def can_attend_meetings(intervals) do

end
end
```

### Erlang Solution:

```
-spec can_attend_meetings(Intervals :: [[integer()]]) -> boolean().
can_attend_meetings(Intervals) ->
.
```

## Racket Solution:

```
(define/contract (can-attend-meetings intervals)
(-> (listof (listof exact-integer?)) boolean?)
)
```