Problem 56: Merge Intervals

Problem Information

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

Problem Description

Given an array of

intervals

where

intervals[i] = [start

i

, end

i

]

, merge all overlapping intervals, and return

an array of the non-overlapping intervals that cover all the intervals in the input

.

Example 1:

Input:

intervals = [[1,3],[2,6],[8,10],[15,18]]

Output:

[[1,6],[8,10],[15,18]]

Explanation:

Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Example 2:

Input:

intervals = [[1,4],[4,5]]

Output:

[[1,5]]

Explanation:

Intervals [1,4] and [4,5] are considered overlapping.

Example 3:

Input:

intervals = [[4,7],[1,4]]

Output:

[[1,7]]

Explanation:

Intervals [1,4] and [4,7] are considered overlapping.

Constraints:

1 <= intervals.length <= 10

4

intervals[i].length == 2

0 <= start

i

<= end

i

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> merge(vector<vector<int>>& intervals) {

}
};
```

Java:

```
class Solution {
public int[][] merge(int[][] intervals) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function merge(intervals: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] Merge(int[][] intervals) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** merge(int** intervals, int intervalsSize, int* intervalsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func merge(intervals [][]int) [][]int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> merge(List<List<int>> intervals) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (merge intervals)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Intervals
* 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<vector<int>> merge(vector<vector<int>>& intervals) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Intervals
* 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 int[][] merge(int[][] intervals) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Intervals
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 merge(self, intervals: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Merge Intervals
* 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[][]} intervals
* @return {number[][]}
*/
var merge = function(intervals) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Intervals
* 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 merge(intervals: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Merge Intervals
* 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 int[][] Merge(int[][] intervals) {

}
}
```

C Solution:

```
/*
* Problem: Merge Intervals
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** merge(int** intervals, int intervalsSize, int* intervalsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Merge Intervals
// 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 merge(intervals [][]int) [][]int {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Merge Intervals
// 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 merge(intervals: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> merge(List<List<int>> intervals) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

```
(define/contract (merge intervals)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```