Problem 57: Insert Interval

Problem Information

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

Problem Description

You are given an array of non-overlapping intervals

intervals

where

intervals[i] = [start

i

, end

i

]

represent the start and the end of the

i

th

interval and

intervals

is sorted in ascending order by

start

i

. You are also given an interval

newInterval = [start, end]

that represents the start and end of another interval.

Insert

newInterval

into

intervals

such that

intervals

is still sorted in ascending order by

start

i

and

intervals

still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return

intervals

after the insertion

.

Note

that you don't need to modify

intervals

in-place. You can make a new array and return it.

Example 1:

Input:

intervals = [[1,3],[6,9]], newInterval = [2,5]

Output:

[[1,5],[6,9]]

Example 2:

Input:

intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

Output:

[[1,2],[3,10],[12,16]]

Explanation:

Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

Constraints:

0 <= intervals.length <= 10

4

intervals[i].length == 2

0 <= start

i

<= end

i

<= 10

5

intervals

is sorted by

start

i

in

ascending

order.

newInterval.length == 2

0 <= start <= end <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>&
newInterval) {

}
};
```

Java:

```
class Solution {
public int[][] insert(int[][] intervals, int[] newInterval) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} intervals
* @param {number[]} newInterval
* @return {number[][]}
*/
```

```
var insert = function(intervals, newInterval) {

};
```

TypeScript:

```
function insert(intervals: number[][], newInterval: number[]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] Insert(int[][] intervals, int[] newInterval) {

}
}
```

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** insert(int** intervals, int intervalsSize, int* intervalsColSize, int*
newInterval, int newIntervalSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func insert(intervals [][]int, newInterval []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun insert(intervals: Array<IntArray>, newInterval: IntArray):
Array<IntArray> {
```

```
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @param {Integer[]} new_interval
# @return {Integer[][]}
def insert(intervals, new_interval)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer[] $newInterval
* @return Integer[][]
*/
function insert($intervals, $newInterval) {

}
```

```
}
```

Dart:

```
class Solution {
List<List<int>> insert(List<List<int>> intervals, List<int> newInterval) {

}
}
```

Scala:

```
object Solution {
def insert(intervals: Array[Array[Int]], newInterval: Array[Int]):
Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec insert(intervals :: [[integer]], new_interval :: [integer]) ::
[[integer]]
def insert(intervals, new_interval) do

end
end
```

Erlang:

```
-spec insert(Intervals :: [[integer()]], NewInterval :: [integer()]) ->
[[integer()]].
insert(Intervals, NewInterval) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Insert Interval
* 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>> insert(vector<vector<int>>& intervals, vector<int>&
newInterval) {

}
};
```

Java Solution:

```
/**
* Problem: Insert Interval
* 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[][] insert(int[][] intervals, int[] newInterval) {

}
}
```

Python3 Solution:

```
"""
Problem: Insert Interval
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 insert(self, intervals: List[List[int]], newInterval: List[int]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Insert Interval
* 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
* @param {number[]} newInterval
```

```
* @return {number[][]}
*/
var insert = function(intervals, newInterval) {

};
```

TypeScript Solution:

```
/**
* Problem: Insert Interval
* 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 insert(intervals: number[][], newInterval: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Insert Interval
* 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[][] Insert(int[][] intervals, int[] newInterval) {

}
}
```

C Solution:

```
/*
* Problem: Insert Interval
* 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** insert(int** intervals, int intervalsSize, int* intervalsColSize, int*
newInterval, int newIntervalSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Insert Interval
// 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 insert(intervals [][]int, newInterval []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun insert(intervals: Array<IntArray>, newInterval: IntArray):
Array<IntArray> {

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Insert Interval
// 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 insert(intervals: Vec<Vec<i32>>, new_interval: Vec<i32>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @param {Integer[]} new_interval
# @return {Integer[][]}
def insert(intervals, new_interval)

end
```

PHP Solution:

```
class Solution {

/**
```

```
* @param Integer[][] $intervals
* @param Integer[] $newInterval
* @return Integer[][]
*/
function insert($intervals, $newInterval) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> insert(List<List<int>> intervals, List<int> newInterval) {

}
}
```

Scala Solution:

```
object Solution {
def insert(intervals: Array[Array[Int]], newInterval: Array[Int]):
Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec insert(intervals :: [[integer]], new_interval :: [integer]) ::
[[integer]]
def insert(intervals, new_interval) do

end
end
```

Erlang Solution:

```
-spec insert(Intervals :: [[integer()]], NewInterval :: [integer()]) ->
[[integer()]].
insert(Intervals, NewInterval) ->
```

```
.
```

Racket Solution:

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