Problem 986: Interval List Intersections

Problem Information

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

Problem Description

You are given two lists of closed intervals,

firstList

and

secondList

, where

firstList[i] = [start

i

, end

i

]

and

secondList[j] = [start

j

, end

j

]

. Each list of intervals is pairwise

disjoint

and in

sorted order

.

Return

the intersection of these two interval lists

.

A

closed interval

[a, b]

(with

a <= b

) denotes the set of real numbers

x

with

a <= x <= b

.

The

intersection

of two closed intervals is a set of real numbers that are either empty or represented as a closed interval. For example, the intersection of

[1, 3]

and

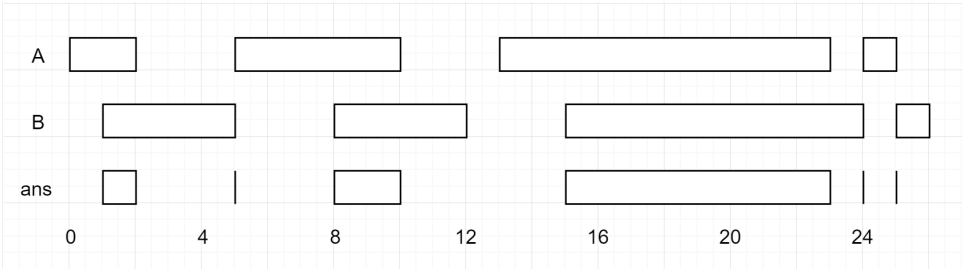
[2, 4]

is

[2, 3]

.

Example 1:



Input:

firstList = [[0,2],[5,10],[13,23],[24,25]], secondList = [[1,5],[8,12],[15,24],[25,26]]

Output:

[[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]

Example 2:

Input:

firstList = [[1,3],[5,9]], secondList = []

Output:

[]

Constraints:

0 <= firstList.length, secondList.length <= 1000

firstList.length + secondList.length >= 1

0 <= start

i

< end

i

<= 10

9

end

i

< start

i+1

0 <= start

j

< end

j

<= 10

9

end

j

< start

j+1

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> intervalIntersection(vector<vector<int>>& firstList,
vector<vector<int>>& secondList) {

}
};
```

Java:

```
class Solution {
public int[][] intervalIntersection(int[][] firstList, int[][] secondList) {

}
}
```

Python3:

```
class Solution:
def intervalIntersection(self, firstList: List[List[int]], secondList:
```

```
List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def intervalIntersection(self, firstList, secondList):
"""
:type firstList: List[List[int]]
:type secondList: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} firstList
* @param {number[][]} secondList
* @return {number[][]}
*/
var intervalIntersection = function(firstList, secondList) {

};
```

TypeScript:

```
function intervalIntersection(firstList: number[][], secondList: number[][]):
number[][] {

};
```

C#:

```
public class Solution {
public int[][] IntervalIntersection(int[][] firstList, int[][] secondList) {

}
}
```

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** intervalIntersection(int** firstList, int firstListSize, int*
firstListColSize, int** secondList, int secondListSize, int*
secondListColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func intervalIntersection(firstList [][]int, secondList [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun intervalIntersection(firstList: Array<IntArray>, secondList:
Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func intervalIntersection(_ firstList: [[Int]], _ secondList: [[Int]]) ->
[[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn interval_intersection(first_list: Vec<Vec<i32>>, second_list:
Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} first_list
# @param {Integer[][]} second_list
# @return {Integer[][]}
def interval_intersection(first_list, second_list)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $firstList
* @param Integer[][] $secondList
* @return Integer[][]
*/
function intervalIntersection($firstList, $secondList) {

}
}
```

Dart:

```
class Solution {
List<List<int>> intervalIntersection(List<List<int>> firstList,
List<List<int>> secondList) {

}
}
```

Scala:

```
object Solution {
def intervalIntersection(firstList: Array[Array[Int]], secondList:
Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec interval_intersection(first_list :: [[integer]], second_list ::
[[integer]]) :: [[integer]]
def interval_intersection(first_list, second_list) do

end
end
```

Erlang:

```
-spec interval_intersection(FirstList :: [[integer()]], SecondList ::
[[integer()]]) -> [[integer()]].
interval_intersection(FirstList, SecondList) ->
.
```

Racket:

```
(define/contract (interval-intersection firstList secondList)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Interval List Intersections
* 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>> intervalIntersection(vector<vector<int>>& firstList,
vector<vector<int>>& secondList) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Interval List Intersections
* 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[][] intervalIntersection(int[][] firstList, int[][] secondList) {

}
}
```

Python3 Solution:

```
"""
Problem: Interval List Intersections
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 intervalIntersection(self, firstList: List[List[int]], secondList:
List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def intervalIntersection(self, firstList, secondList):
"""
:type firstList: List[List[int]]
:type secondList: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Interval List Intersections
* 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[][]} firstList
* @param {number[][]} secondList
* @return {number[][]}
*/
var intervalIntersection = function(firstList, secondList) {

};
```

TypeScript Solution:

```
/**
* Problem: Interval List Intersections
* 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 intervalIntersection(firstList: number[][], secondList: number[][]):
```

```
number[][] {

};
```

C# Solution:

```
/*
* Problem: Interval List Intersections
* 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[][] IntervalIntersection(int[][] firstList, int[][] secondList) {

}
}
```

C Solution:

```
/*
* Problem: Interval List Intersections
* 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** intervalIntersection(int** firstList, int firstListSize, int*
firstListColSize, int** secondList, int secondListSize, int*
```

```
secondListColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Interval List Intersections
// 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 intervalIntersection(firstList [][]int, secondList [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun intervalIntersection(firstList: Array<IntArray>, secondList:
Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func intervalIntersection(_ firstList: [[Int]], _ secondList: [[Int]]) ->
[[Int]] {

}
}
```

Rust Solution:

```
// Problem: Interval List Intersections
// 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 interval_intersection(first_list: Vec<Vec<i32>>, second_list:
Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} first_list
# @param {Integer[][]} second_list
# @return {Integer[][]}
def interval_intersection(first_list, second_list)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $firstList
* @param Integer[][] $secondList
* @return Integer[][]
*/
function intervalIntersection($firstList, $secondList) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> intervalIntersection(List<List<int>> firstList,
List<List<int>> secondList) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def intervalIntersection(firstList: Array[Array[Int]], secondList:
Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec interval_intersection(first_list :: [[integer]], second_list ::
[[integer]]) :: [[integer]]
def interval_intersection(first_list, second_list) do

end
end
```

Erlang Solution:

```
-spec interval_intersection(FirstList :: [[integer()]], SecondList ::
[[integer()]]) -> [[integer()]].
interval_intersection(FirstList, SecondList) ->
.
```

Racket Solution:

```
(define/contract (interval-intersection firstList secondList)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
(listof exact-integer?)))
)
```