Problem 1272: Remove Interval

Problem Information

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

Problem Description

A set of real numbers can be represented as the union of several disjoint intervals, where each interval is in the form

[a, b)

. A real number

x

is in the set if one of its intervals

[a, b)

contains

x

(i.e.

a <= x < b

).

You are given a

sorted

list of disjoint intervals

intervals

representing a set of real numbers as described above, where

intervals[i] = [a

i

, b

i

]

represents the interval

[a

i

, b

i

)

. You are also given another interval

toBeRemoved

.

Return

the set of real numbers with the interval

toBeRemoved

removed

from

intervals

. In other words, return the set of real numbers such that every

x

in the set is in

intervals

but

not

in

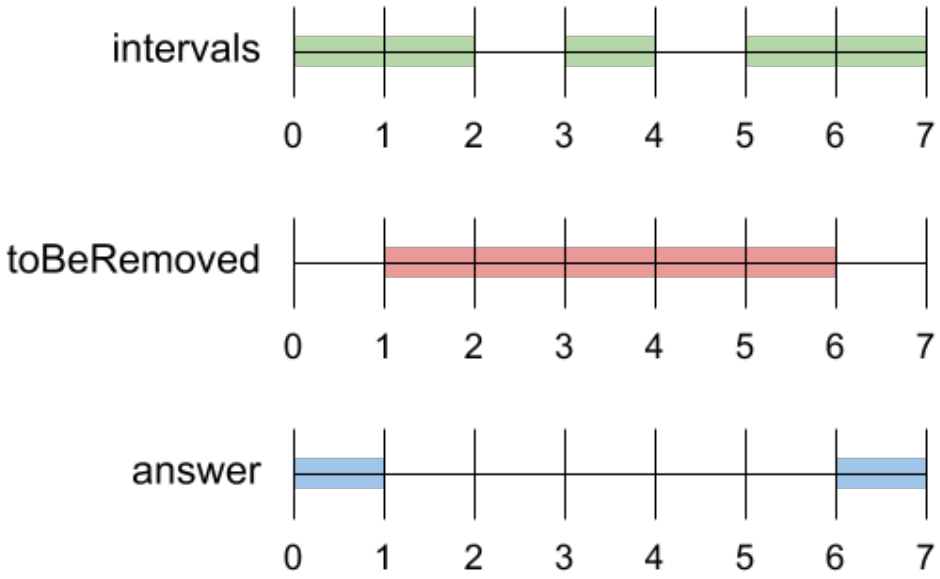
toBeRemoved

. Your answer should be a

sorted

list of disjoint intervals as described above.

Example 1:



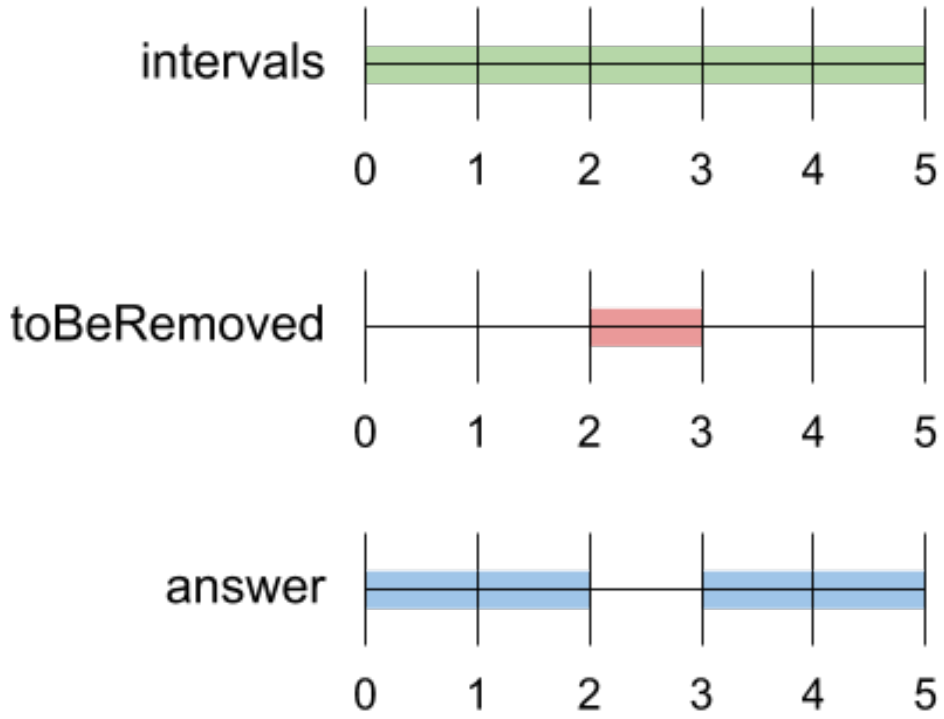
Input:

intervals = [[0,2],[3,4],[5,7]], toBeRemoved = [1,6]

Output:

[[0,1],[6,7]]

Example 2:



Input:

intervals = [[0,5]], toBeRemoved = [2,3]

Output:

[[0,2],[3,5]]

Example 3:

Input:

intervals = [[-5,-4],[-3,-2],[1,2],[3,5],[8,9]], toBeRemoved = [-1,4]

Output:

[[-5,-4],[-3,-2],[4,5],[8,9]]

Constraints:

1 <= intervals.length <= 10

4

-10

9

<= a

i

< b

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> removeInterval(vector<vector<int>>& intervals,
vector<int>& toBeRemoved) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> removeInterval(int[][] intervals, int[]
toBeRemoved) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeInterval(intervals: number[][], toBeRemoved: number[]):
number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> RemoveInterval(int[][] intervals, int[] toBeRemoved)
{

}
}
```

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** removeInterval(int** intervals, int intervalsSize, int*
intervalsColSize, int* toBeRemoved, int toBeRemovedSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func removeInterval(intervals [][]int, toBeRemoved []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun removeInterval(intervals: Array<IntArray>, toBeRemoved: IntArray):
List<List<Int>> {

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @param {Integer[]} to_be_removed
# @return {Integer[][]}
def remove_interval(intervals, to_be_removed)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer[] $toBeRemoved
* @return Integer[][]
*/
function removeInterval($intervals, $toBeRemoved) {

}
}
```

Dart:

```
class Solution {
List<List<int>> removeInterval(List<List<int>> intervals, List<int>
toBeRemoved) {

}
}
```

Scala:

```
object Solution {
def removeInterval(intervals: Array[Array[Int]], toBeRemoved: Array[Int]):
List[List[Int]] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec remove_interval(intervals :: [[integer]], to_be_removed :: [integer])
:: [[integer]]
def remove_interval(intervals, to_be_removed) do

end
end
```

Erlang:

```
-spec remove_interval(Intervals :: [[integer()]], ToBeRemoved :: [integer()])
-> [[integer()]].
remove_interval(Intervals, ToBeRemoved) ->
.
```

Racket:

```
(define/contract (remove-interval intervals toBeRemoved)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove 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>> removeInterval(vector<vector<int>>& intervals,
vector<int>& toBeRemoved) {

}
};
```

Java Solution:

```
/**
* Problem: Remove 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 List<List<Integer>> removeInterval(int[][] intervals, int[]
toBeRemoved) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove 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 removeInterval(self, intervals: List[List[int]], toBeRemoved: List[int])
```

```
-> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove 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[]} toBeRemoved
* @return {number[][]}
*/
var removeInterval = function(intervals, toBeRemoved) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove 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 removeInterval(intervals: number[][], toBeRemoved: number[]):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Remove 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 IList<IList<int>> RemoveInterval(int[][] intervals, int[] toBeRemoved)
{

}
}
```

C Solution:

```
/*
* Problem: Remove 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** removeInterval(int** intervals, int intervalsSize, int*
intervalsColSize, int* toBeRemoved, int toBeRemovedSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Remove 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 removeInterval(intervals [][]int, toBeRemoved []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun removeInterval(intervals: Array<IntArray>, toBeRemoved: IntArray):
List<List<Int>> {

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Remove 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 remove_interval(intervals: Vec<Vec<i32>>, to_be_removed: Vec<i32>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @param {Integer[]} to_be_removed
# @return {Integer[][]}
def remove_interval(intervals, to_be_removed)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer[] $toBeRemoved
* @return Integer[][]
*/
function removeInterval($intervals, $toBeRemoved) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> removeInterval(List<List<int>> intervals, List<int>
toBeRemoved) {

}
}
```

Scala Solution:

```
object Solution {
def removeInterval(intervals: Array[Array[Int]], toBeRemoved: Array[Int]):
List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_interval(intervals :: [[integer]], to_be_removed :: [integer])
:: [[integer]]
def remove_interval(intervals, to_be_removed) do

end
end
```

Erlang Solution:

```
-spec remove_interval(Intervals :: [[integer()]], ToBeRemoved :: [integer()])
-> [[integer()]].
remove_interval(Intervals, ToBeRemoved) ->
.
```

Racket Solution:

```
(define/contract (remove-interval intervals toBeRemoved)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```