

Intervals Intersection (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

Problem Statement

Given two lists of intervals, find the **intersection of these two lists**. Each list consists of **disjoint intervals sorted on their start time**.

Example 1:


```
Input: arr1=[[1, 3], [5, 6], [7, 9]], arr2=[[2, 3], [5, 7]]
Output: [2, 3], [5, 6], [7, 7]
Explanation: The output list contains the common intervals between the two lists.
```


Example 2:


```
Input: arr1=[[1, 3], [5, 7], [9, 12]], arr2=[[5, 10]]
Output: [5, 7], [9, 10]
Explanation: The output list contains the common intervals between the two lists.
```


Try it yourself

Try solving this question here:

 Java

 Python3

 JS


 C++

```
1 import java.util.*;
2
3 class Interval {
4     int start;
5     int end;
6
7     public Interval(int start, int end) {
8         this.start = start;
9         this.end = end;
10    }
11 };
12
13 class IntervalsIntersection {
14
15     public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
16         List<Interval> intervalsIntersection = new ArrayList<Interval>();
17         // TODO: Write your code here
18         return intervalsIntersection.toArray(new Interval[intervalsIntersection.size()]);
19     }
20
21     public static void main(String[] args) {
22         Interval[] input1 = new Interval[] { new Interval(1, 3), new Interval(5, 6), new Interval(7, 9) };
23         Interval[] input2 = new Interval[] { new Interval(2, 3), new Interval(5, 7) };
24         Interval[] result = IntervalsIntersection.merge(input1, input2);
25         System.out.print("Intervals Intersection: ");
26         for (Interval interval : result)
27             System.out.print "[" + interval.start + "," + interval.end + " ";
28         System.out.println();
29     }
30 }
```

Run

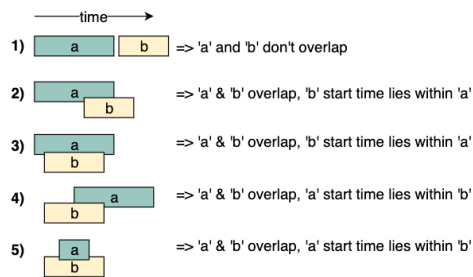
Save

Reset



Solution

This problem follows the [Merge Intervals](#) pattern. As we have discussed under [Insert Interval](#), there are five overlapping possibilities between two intervals 'a' and 'b'. A close observation will tell us that whenever the two intervals overlap, one of the interval's start time lies within the other interval. This rule can help us identify if any two intervals overlap or not.



Now, if we have found that the two intervals overlap, how can we find the overlapped part?

Again from the above diagram, the overlapping interval will be equal to:

```
start = max(a.start, b.start)
end = min(a.end, b.end)
```

That is, the highest start time and the lowest end time will be the overlapping interval.

So our algorithm will be to iterate through both the lists together to see if any two intervals overlap. If two intervals overlap, we will insert the overlapped part into a result list and move on to the next interval which is finishing early.

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS JS

```

1  import java.util.*;
2
3  class Interval {
4      int start;
5      int end;
6
7      public Interval(int start, int end) {
8          this.start = start;
9          this.end = end;
10     }
11 }
12
13 class IntervalsIntersection {
14
15     public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
16         List<Interval> result = new ArrayList<Interval>();
17         int i = 0, j = 0;
18         while (i < arr1.length && j < arr2.length) {
19             // check if the interval arr1[i] intersects with arr2[j]
20             // check if one of the interval's start time lies within the other interval
21             if ((arr1[i].start >= arr2[j].start && arr1[i].start <= arr2[j].end)
22                 || (arr2[j].start >= arr1[i].start && arr2[j].start <= arr1[i].end)) {
23                 // store the intersection part
24                 result.add(new Interval(Math.max(arr1[i].start, arr2[j].start), Math.min(arr1[i].end, arr2[j].end)));
25             }
26
27             // move next from the interval which is finishing first
28             if (arr1[i].end < arr2[j].end)

```

Run

Save

Reset

Time complexity

As we are iterating through both the lists once, the time complexity of the above algorithm is $O(N + M)$, where 'N' and 'M' are the total number of intervals in the input arrays respectively.

Space complexity

Ignoring the space needed for the result list, the algorithm runs in constant space $O(1)$.

← Back

Insert Interval (medium)

Next →

Conflicting Appointments (medium)

✓ Completed

Report an Issue Ask a Question

