≡ >_ educative   ⚙ 📋

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 betwee
n 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 betwee
n the two lists.
```

 educative

Try solving this question here:

| Java | Python3 | JS | C++ |

```java
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.siz
19    }
20
21    public static void main(String[] args) {
22      Interval[] input1 = new Interval[] { new Interval(1, 3), new Interval(5, 6)
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      input1 = new Interval[] { new Interval(1, 3), new Interval(5, 7), new Inter
31      input2 = new Interval[] { new Interval(5, 10) };
32      result = IntervalsIntersection.merge(input1, input2);
33      System.out.print("Intervals Intersection: ");
34      for (Interval interval : result)
35        System.out.print("[" + interval.start + "," + interval.end + "] ");
36    }
37  }
38
```

educative

```
Output                                                2.648s

  Intervals Intersection:
  Intervals Intersection:
```

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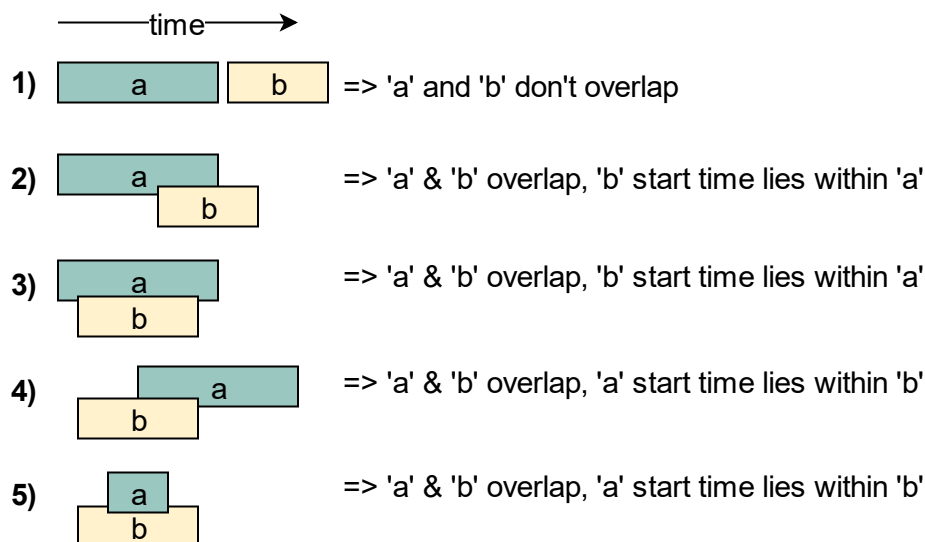
Now, if we have found that the two intervals overlap, how can we find the overlapped part?

```
    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

```java
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 arr[i] intersects with arr2[j]
20        // check if one of the interval's start time lies within the other interv
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))
```

```
26
27         // move next from the interval which is finishing first
28         if (arr1[i].end < arr2[j].end)
29            i++;
30         else
31            j++;
32      }
33
34      return result.toArray(new Interval[result.size()]);
35    }
36
37    public static void main(String[] args) {
38      Interval[] input1 = new Interval[] { new Interval(1, 3), new Interval(5, 6)
39      Interval[] input2 = new Interval[] { new Interval(2, 3), new Interval(5, 7)
40      Interval[] result = IntervalsIntersection.merge(input1, input2);
41      System.out.print("Intervals Intersection: ");
42      for (Interval interval : result)
43        System.out.print("[" + interval.start + "," + interval.end + "] ");
44      System.out.println();
45
46      input1 = new Interval[] { new Interval(1, 3), new Interval(5, 7), new Inter
47      input2 = new Interval[] { new Interval(5, 10) };
48      result = IntervalsIntersection.merge(input1, input2);
49      System.out.print("Intervals Intersection: ");
50      for (Interval interval : result)
51        System.out.print("[" + interval.start + "," + interval.end + "] ");
52    }
53  }
54
```

Output   2.287s

```
Intervals Intersection: [2,3] [5,6] [7,7]
Intervals Intersection: [5,7] [9,10]
```

Time complexity

number of intervals in the input arrays respectively.

Space complexity #

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

☑ **Mark as Completed**

← Back

Insert Interval (medium)

Next →

Conflicting Appointments (medium)

Stuck? Get help on   **DISCUSS**   Send feedback   5 Recommendations