

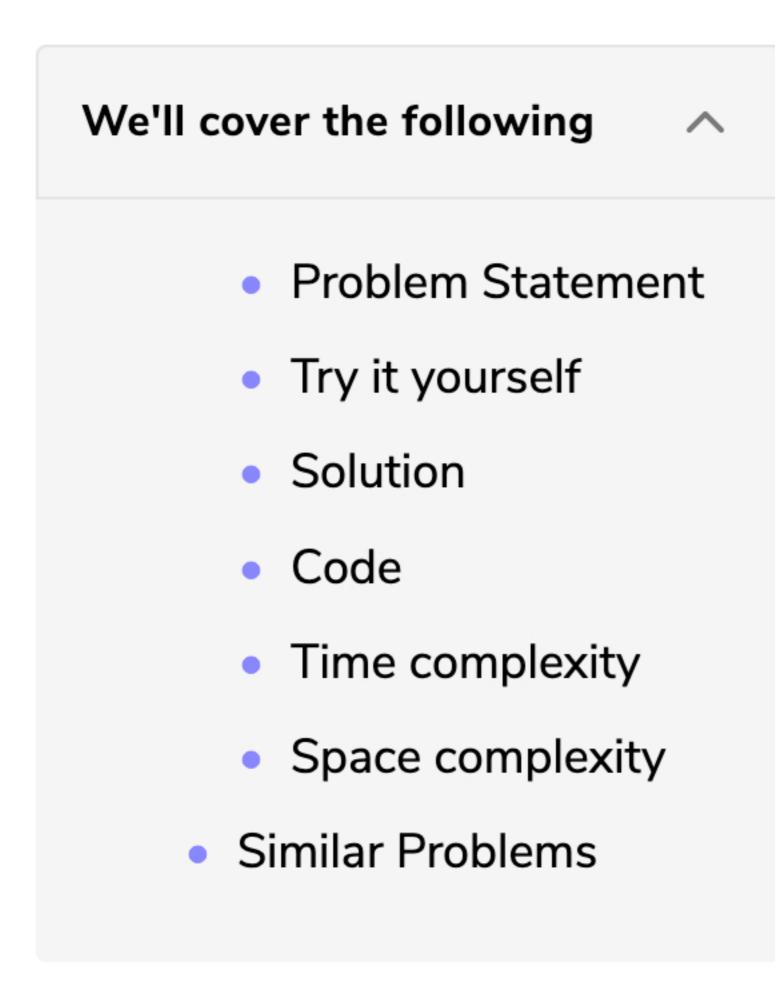
Challenge 1

Challenge 2

Problem Challenge 2

Solution Review: Problem

# Conflicting Appointments (medium)



### Problem Statement

Given an array of intervals representing 'N' appointments, find out if a person can attend all the appointments.

€

? Ask a Question

Example 1:

```
Appointments: [[1,4], [2,5], [7,9]]
Output: false
Explanation: Since [1,4] and [2,5] overlap, a person cannot attend both of these app
ointments.
```

### Example 2:

```
Appointments: [[6,7], [2,4], [8,12]]
Output: true
Explanation: None of the appointments overlap, therefore a person can attend all o
f them.
```

Example 3:

```
Appointments: [[4,5], [2,3], [3,6]]
Output: false
Explanation: Since [4,5] and [3,6] overlap, a person cannot attend both of these app
ointments.
```

# Try it yourself

Try solving this question here:

```
Python3
                        JS JS
                                    G C++
Java
        this.start = start;
 8
        this.end = end;
 9
10
11
12
    class ConflictingAppointments {
14
      public static boolean canAttendAllAppointments(Interval[] intervals) {
15
        // TODO: Write your code here
16
17
        return true;
18
19
      public static void main(String[] args) {
20
        Interval[] intervals = { new Interval(1, 4), new Interval(2, 5), new Interval(7, 9) };
21
        boolean result = ConflictingAppointments.canAttendAllAppointments(intervals);
22
        System.out.println("Can attend all appointments: " + result);
        Interval[] intervals1 = { new Interval(6, 7), new Interval(2, 4), new Interval(8, 12) };
25
        result = ConflictingAppointments.canAttendAllAppointments(intervals1);
26
        System.out.println("Can attend all appointments: " + result);
28
        Interval[] intervals2 = { new Interval(4, 5), new Interval(2, 3), new Interval(3, 6) };
29
        result = ConflictingAppointments.canAttendAllAppointments(intervals2);
30
31
        System.out.println("Can attend all appointments: " + result);
32
33
34
Run
                                                                                    Reset
                                                                           Save
```

# The problem follows the Merge Intervals pattern. We can sort all the intervals by start time and

Solution

then check if any two intervals overlap. A person will not be able to attend all appointments if any two appointments overlap. Code

```
Here is what our algorithm will look like:
```

**⊗** C++ Python3 JS JS 👙 Java

```
import java.util.*;
      class Interval {
        int start;
        int end;
   6
        public Interval(int start, int end) {
          this.start = start;
   8
          this.end = end;
   9
  10
  12
      class ConflictingAppointments {
  14
        public static boolean canAttendAllAppointments(Interval[] intervals) {
  15
          // sort the intervals by start time
  16
          Arrays.sort(intervals, (a, b) -> Integer.compare(a.start, b.start));
  18
          // find any overlapping appointment
  19
          for (int i = 1; i < intervals.length; i++) {</pre>
  20
            if (intervals[i].start < intervals[i - 1].end) {</pre>
  21
              // please note the comparison above, it is "<" and not "<="</pre>
              // while merging we needed "<=" comparison, as we will be merging the two
  23
              // intervals having condition "intervals[i].start == intervals[i - 1].end" but
  24
              // such intervals don't represent conflicting appointments as one starts right
  25
              // after the other
  26
              return false;
  28
  Run
                                                                                       Reset
                                                                              Save
Time complexity
```

# The time complexity of the above algorithm is O(N \* log N), where 'N' is the total number of

appointments. Though we are iterating the intervals only once, our algorithm will take O(N \* log N) since we need to sort them in the beginning. Space complexity

# The space complexity of the above algorithm will be O(N), which we need for sorting. For Java,

Arrays.sort() uses Timsort, which needs O(N) space.

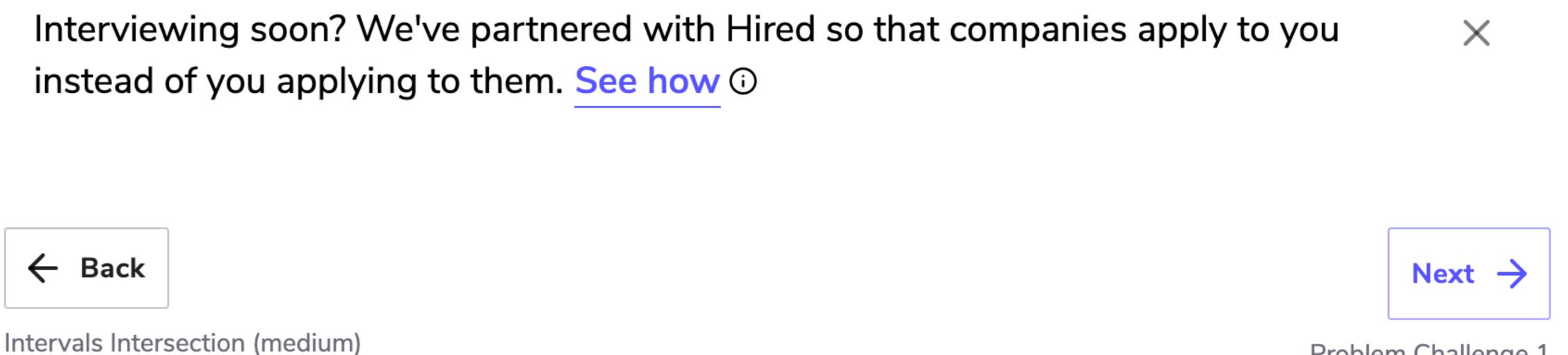
# **Problem 1:** Given a list of appointments, find all the conflicting appointments.

**Example:** 

Similar Problems

```
Appointments: [[4,5], [2,3], [3,6], [5,7], [7,8]]
Output:
[4,5] and [3,6] conflict.
```

[3,6] and [5,7] conflict.



Problem Challenge 1

✓ Mark as Completed