1229. Meeting Scheduler

Medium 534 23 Add to List Share

Given the availability time slots arrays `slots1` and `slots2` of two people and a meeting duration `duration`, return the **earliest time slot** that works for both of them and is of duration `duration`.

If there is no common time slot that satisfies the requirements, return an **empty array**.

The format of a time slot is an array of two elements `[start, end]` representing an inclusive time range from `start` to `end`.

It is guaranteed that no two availability slots of the same person intersect with each other. That is, for any two time slots `[start1, end1]` and `[start2, end2]` of the same person, either `start1 > end2` or `start2 > end1`.

Example 1:

```
Input: slots1 = [[10,50],[60,120],[140,210]], slots2 = [[0,15],[60,70]], duration = 8
Output: [60,68]
```

Example 2:

```
Input: slots1 = [[10,50],[60,120],[140,210]], slots2 = [[0,15],[60,70]], duration = 12
Output: []
```

Constraints:

- `1 <= slots1.length, slots2.length <= 10^4`
- `slots1[i].length, slots2[i].length == 2`
- `slots1[i][0] < slots1[i][1]`
- `slots2[i][0] < slots2[i][1]`
- `0 <= slots1[i][j], slots2[i][j] <= 10^9`
- `1 <= duration <= 10^6`

Accepted 43,174 | Submissions 78,887

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Amazon | 4 DoorDash | 2 Apple | 2

Related Topics

Array Two Pointers Sorting

Hide Hint 1

Assume that in the solution, the selected slot from slotsA is bigger than the respectively selected slot from slotsB.

Hide Hint 2

Use two pointers in order to try all the possible intersections, and check the length.

Hide Hint 3

Do the same in step N° 1 but now assume that the selected slot from slotsB is bigger, return the minimum of the two options.

Java Autocomplete

```java
class Solution {
    public List<Integer> minAvailableDuration(int[][] slots1, int[][] slots2, int duration) {
        Arrays.sort(slots1, (a, b) -> a[0] - b[0]);
        Arrays.sort(slots2, (a, b) -> a[0] - b[0]);

        int pointer1 = 0, pointer2 = 0;

        while (pointer1 < slots1.length && pointer2 < slots2.length) {
            // find the boundaries of the intersection, or the common slot
            int intersectLeft = Math.max(slots1[pointer1][0], slots2[pointer2][0]);
            int intersectRight = Math.min(slots1[pointer1][1], slots2[pointer2][1]);
            if (intersectRight - intersectLeft >= duration) {
                return new ArrayList<Integer>(Arrays.asList(intersectLeft, intersectLeft + duration));
            }
            // always move the one that ends earlier
            if (slots1[pointer1][1] < slots2[pointer2][1]) {
                pointer1++;
            } else {
                pointer2++;
            }
        }
        return new ArrayList<Integer>();
    }
}
```