2589. 完成所有任务的最少时间

你有一台电脑，它可以 **同时** 运行无数个任务。给你一个二维整数数组 `tasks` ，其中 `tasks[i] = [starti, endi, durationi]` 表示第 `i` 个任务需要在 **闭区间** 时间段 `[starti, endi]` 内运行 `durationi` 个整数时间点（但不需要连续）。

当电脑需要运行任务时，你可以打开电脑，如果空闲时，你可以将电脑关闭。

请你返回完成所有任务的情况下，电脑最少需要运行多少秒。

示例 1：

```
输入：tasks = [[2,3,1],[4,5,1],[1,5,2]]
输出：2
解释：
- 第一个任务在闭区间 [2, 2] 运行。
- 第二个任务在闭区间 [5, 5] 运行。
- 第三个任务在闭区间 [2, 2] 和 [5, 5] 运行。
电脑总共运行 2 个整数时间点。
```

示例 2：

```
输入：tasks = [[1,3,2],[2,5,3],[5,6,2]]
输出：4
解释：
- 第一个任务在闭区间 [2, 3] 运行
- 第二个任务在闭区间 [2, 3] 和 [5, 5] 运行。
- 第三个任务在闭区间 [5, 6] 运行。
电脑总共运行 4 个整数时间点。
```

提示：

- `1 <= tasks.length <= 2000`
- `tasks[i].length == 3`
- `1 <= starti, endi <= 2000`
- `1 <= durationi <= endi - starti + 1`

思路：贪心+暴力

提示 1

按照区间右端点从小到大进行排序

提示 2

排序后，对于区间 $task[i]$ 来说，它右侧的任务区间要么和它没有交集，要么包含它的一部分**后缀**。

例如排序后的区间为 $[1,5],[3,7],[6,8]$，对于 $[1,5]$ 来说，它右边的区间要么和它没有交集，例如 $[6,8]$；要么交集是 $[1,5]$ 的后缀，例如 $[1,5]$ 和 $[3,7]$ 的交集是 $[3,5]$，这是 $[1,5]$ 的后缀（$3,4,5$ 是 $1,2,3,4,5$ 的后缀）。

提示 3

遍历排序后的任务，先统计区间内的已运行的电脑运行时间点，如果个数小于 $duration$，则需要新增时间点。根据提示 2，尽量把新增的时间点安排在区间 $[start, end]$ 的后缀上，这样下一个区间就能统计到更多已运行的时间点。

1. Java

```java
class Solution {
    public int findMinimumTime(int[][] tasks) {
        Arrays.sort(tasks, (a, b) -> a[1] - b[1]);
        int ans = 0;
        int mx = tasks[tasks.length - 1][1];
        boolean[] run = new boolean[mx + 1];
        for (int[] t : tasks) {
            int start = t[0];
            int end = t[1];
            int d = t[2];
            for (int i = start; i <= end; i++) {
                if (run[i]) {
                    d--; // 去掉运行中的时间点
                }
            }
            for (int i = end; d > 0; i--) { // 剩余的 d 填充区间后缀
                if (!run[i]) {
                    run[i] = true; // 运行
                    d--;
                    ans++;
                }
            }
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def findMinimumTime(self, tasks: List[List[int]]) -> int:
        tasks.sort(key=lambda t: t[1])
        run = [False] * (tasks[-1][1] + 1)
        for start, end, d in tasks:
            d -= sum(run[start: end + 1])  # 去掉运行中的时间点
            if d <= 0:  # 该任务已完成
                continue
            # 该任务尚未完成，从后往前找没有运行的时间点
            for i in range(end, start - 1, -1):
                if run[i]:  # 已运行
                    continue
                run[i] = True  # 运行
                d -= 1
                if d == 0:
                    break
        return sum(run)
```