Schedule-slopping Friendslop. Recently, *Big Walk*, a new hit Friendslop game, released to critically aclaimed reviews[1]. Justin would like to play *Big Walk* with his friends, but they are all old adults with jobs and prior commitments or something. Your objective is to find the number of times which Justin and his friends can all play (assuming there is such a time). They only want to play *Big Walk* if they can all meet at least 6 times (or 7 if they're bad[2]).

Formally, there are m friends. Each friend i has a list of times they are busy represented by disjoint, sorted ranges $L_i = \{[a_1^i, b_1^i], \ldots [a_t^i, b_t^i]\}$. Each range $[a_j^i, b_j^i]$ represents the *inclusive* start a_j^i and end b_j^i of their busy times. Assume time (measured in some arbitrary unit) starts at 0 and ends at the largest number given in the input. In total, there are n busy times. Give an algorithm that takes $(L_1, L_2, \ldots, L_m)$ as input and outputs the number of times they can meet. In other words, output the number of occurences where there is a gap between busy times. [3]

Example ($m = 2, n = 5$):

Input: $L_1 = [(1, 2), (4, 6), (8, 12)]$
$L_2 = [(1, 3), (5, 9)]$
Output: 2 because the number of free times are $[(0, 1), (3, 4)]$

Solution. The algorithm is as follows:

Algorithm $\mathcal{A}(L = [L_1, \ldots, L_m])$

Recall that each input list L_1 contains tuples (s, e) that represent start time s and end time e.

1. Let L' be the list of time ranges by concatenating all input lists $L_1, \ldots, L_m$ together.
2. Let L_s be the list L' sorted by start times (Break ties by end times).
3. Let `res` $\leftarrow 0$ and `busyUntil` $\leftarrow 0$.
4. For $i = 1, \ldots, n$:
   (a) Let $(s_i, e_i) \leftarrow L_s[i]$.
   (b) If $s_i >$ `busyUntil`: Set `res` $\leftarrow$ `res` $+ 1$.
   (c) Set `busyUntil` $\leftarrow \max($`busyUntil`$, e_i)$.
5. Output `res`.

Time complexity. This algorithm takes $O(n \log n)$ time overall. Step 2 takes $O(n \log n)$ time to sort. All other outer steps take linear $O(n)$ time. Note that step 2 and the overall

[1] As of writing this problem, it has a 10/10 on Steam, 4.6/5 on the PlayStation store, and a 91% on Metacritic.

[2] _-- --^

[3] Hint: Can you give a $O(n^2)$ algorithm? Can you give a faster algorithm?

algorithm can be done in $O(n \log m)$ time using a m-wise merge sort of sorted arrays (via a binary heap – the general idea is to keep at most m items in the heap, using the assumed sorted individual arrays).

Correctness. We show that the following two invariant lemmas hold for the step 4 loop.

Lemma 1. *After the i'th loop iteration, `busyUntil` is set as the maximum end time of the first i intervals of L_s.*

Lemma 2. *After the i'th loop iteration, `res` is set to be the number of free times of the first i intervals of L_s.*

Observe that correctness follows from Lemma 2 for $i = n$.

We start with proving Lemma 1 via induction on i.

Proof. For $i = 0$, `busyUntil` $= 0$, and the claim is vacuously true for 0 intervals of L_s.

Assume that for some $0 < i < n$, that Lemma 1 is true. We want to show that the first claim is true for $i + 1$ iterations, that is, after the $(i + 1)$'th loop iteration, `busyUntil` is set as the maximum end time of the first $(i + 1)$ intervals of L_s. By line 4c in the algorithm, we set `busyUntil` $\leftarrow \max($`busyUntil`$, e_{i+1})$. Since `busyUntil` is equal to the first maximum end time of the first i intervals of L_s by the IH, line 4c correctly sets `busyUntil` to be the maximum end time of the first $i + 1$ intervals. □

We proceed with proving Lemma 2 about `res`.

Proof. For $i = 0$, `res` is vacuously correct for 0 intervals.

Assume that for some $0 < i < n$, that the Lemma 2 is true. We want to show that the first claim is true for $i + 1$ iterations, that is, after the $(i + 1)$'th loop iteration, `res` is set to be the number of free times of the first $i+1$ intervals of L_s. On line 4b, we set `res` $\leftarrow$ `res` $+1$ if $s_{i+1} >$ `busyUntil`. By Lemma 1, we know that `busyUntil` is equal to the maximum end time of the first i intervals. If $s_{i+1} >$ `busyUntil`, then all previous intervals ended prior to `bustUntil` and there exists a free time in the time range between $($`busyUntil`$, s_{i+1})$. Thus, we increment `res` correctly.

Otherwise, suppose $s_{i+1} \leq$ `busyUntil`. Then, there exists some $j < i + 1$ such that for interval $(s_j, e_j) \in L_s$, we have $e_j =$ `busyUntil`. Since the intervals are sorted by start time,

$$s_j \leq s_{i+1} \leq e_j,$$

so s_{i+1} lies within a previous interval. Thus, leaving `res` unchanged is correct. □