Consider the following variation on the *Interval Scheduling Problem* from lecture. You have a processor that can operate 24 hours a day, every day. People submit requests to run *daily jobs* on the processor. Each such job comes with a *start time* and an *end time*; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. (Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we saw in the Interval Scheduling Problem.)

Given a list of n such jobs, your goal is to accept *as many jobs as possible* (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n, the number of jobs. You may assume for simplicity that no two jobs have the same start or end times.

Example: Consider the following four jobs, specified by *(start-time, end-time)* pairs.

(6 pm, 6 am), (9 pm, 4 am), (3 am, 2 pm), (1 pm, 7 pm).

The unique solution would be to pick the two jobs (9 pm, 4 am) and (1 pm, 7 pm), which can be scheduled without overlapping.

Let $I_1, \ldots, I_n$ denote the n intervals. We say that an *I_j-restricted solution* is one that contains the interval I_j.

Here is an algorithm, for fixed j, to compute an I_j-restricted solution of maximum size. Let x be a point contained in I_j. First delete I_j and all intervals that overlap it. The remaining intervals do not contain the point x, so we can "cut" the time-line at x and produce an instance of the Interval Scheduling Problem from class. We solve this in $O(n)$ time, assuming that the intervals are ordered by ending time.

Now, the algorithm for the full problem is to compute an I_j-restricted solution of maximum size for each $j = 1, \ldots, n$. This takes a total time of $O(n^2)$. We then pick the largest of these solutions, and claim that it is an optimal solution. To see this, consider the optimal solution to the full problem, consisting of a set of intervals S. Since $n > 0$, there is some interval $I_j \in S$; but then S is an optimal I_j-restricted solution, and so our algorithm will produce a solution at least as large as S.