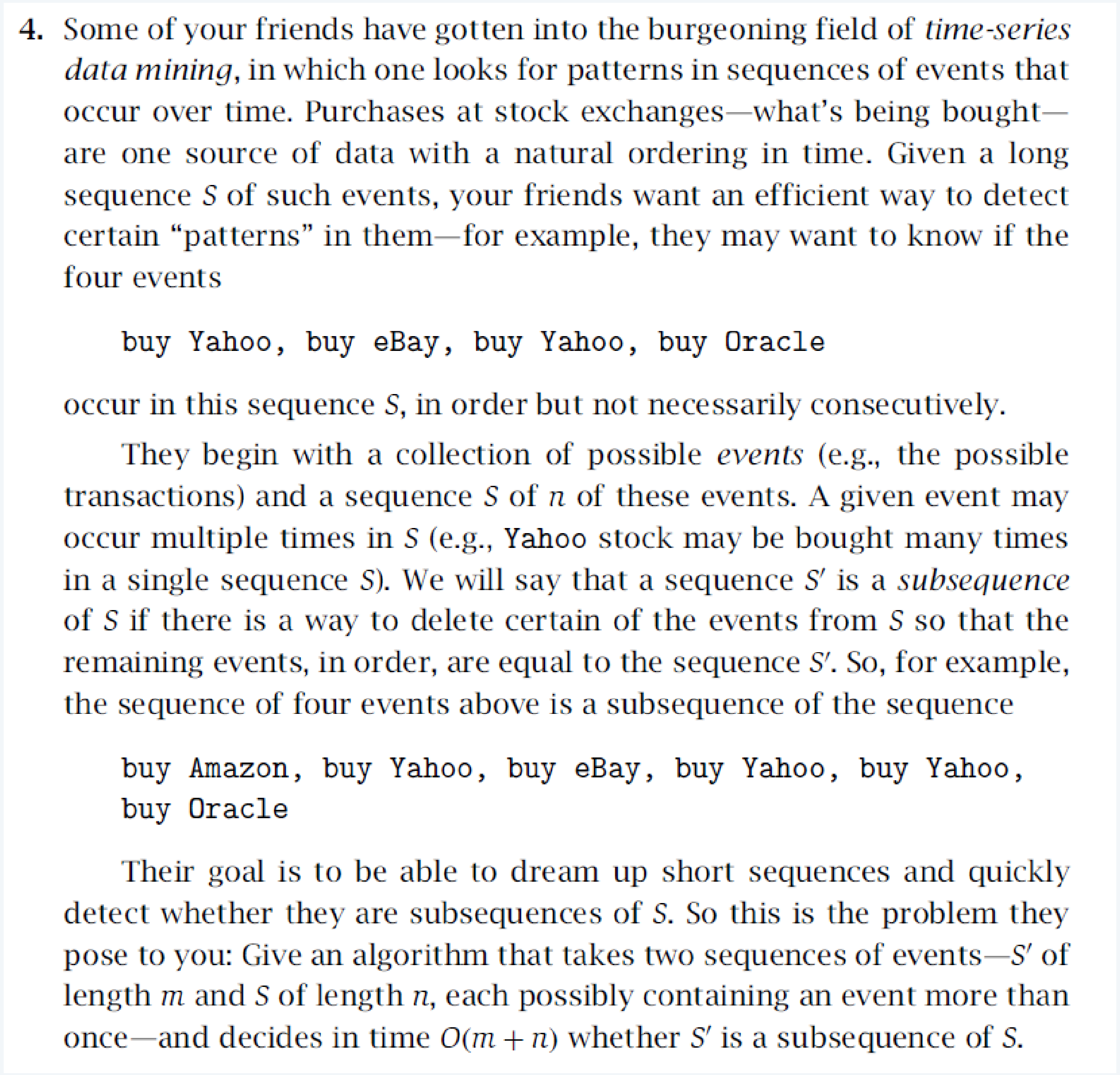
CS 3330 Algorithms

3. Greedy Algorithm

- *greedy algorithms*

Greedy algorithms



4. Some of your friends have gotten into the burgeoning field of *time-series data mining*, in which one looks for patterns in sequences of events that occur over time. Purchases at stock exchanges—what's being bought—are one source of data with a natural ordering in time. Given a long sequence S of such events, your friends want an efficient way to detect certain "patterns" in them—for example, they may want to know if the four events

```
buy Yahoo, buy eBay, buy Yahoo, buy Oracle
```

occur in this sequence S, in order but not necessarily consecutively.

They begin with a collection of possible *events* (e.g., the possible transactions) and a sequence S of n of these events. A given event may occur multiple times in S (e.g., `Yahoo` stock may be bought many times in a single sequence S). We will say that a sequence S' is a *subsequence* of S if there is a way to delete certain of the events from S so that the remaining events, in order, are equal to the sequence S'. So, for example, the sequence of four events above is a subsequence of the sequence

```
buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo,
buy Oracle
```

Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of S. So this is the problem they pose to you: Give an algorithm that takes two sequences of events—S' of length m and S of length n, each possibly containing an event more than once—and decides in time $O(m + n)$ whether S' is a subsequence of S.

Greedy algorithms

$S = (S_1, S_2, \dots, S_n)$ is a $\underset{\text{order matters}}{\underline{\text{sequence}}}$ of strings

$S' = (S'_1, S'_2, \dots, S'_m)$ is a $\underset{\text{order matters}}{\underline{\text{sequence}}}$ of strings

We want to see whether S' can be obtained from S by removing some its members:

$S = (\cancel{a}, \cancel{a}, b, \cancel{c}, b, \cancel{a})$ & $S' = (b, b) \longrightarrow$ Yes

Complexity: $O(m+n) \longrightarrow$ Greedy

have two pointers, pointing at elements of S & S'

Greedy algorithms

p: points at starting location of S, $(\overset{\overset{p}{\downarrow}}{s_1}, s_2, \dots, s_n)$

p': points at starting location of S', $(\underset{\underset{p'}{\uparrow}}{s'_1}, s'_2, \dots, s'_m)$

```
While ($p' \neq m+1$ & $p \neq n+1$)
    if ( $S[p] == S[p']$ )
        $p' \longleftarrow p'+1$
    end
    $p \longleftarrow p+1$
end
```

If $p' = m+1$, the S' is subsequence of S.

Greedy algorithms

① Complexity: the loop cannot run for more than n iter. at most we do $m+n$ increments.

② Correctness: we want to argue that

(i) Whenever algorithm claims "S' is subsequence of S", the algorithm is correct: this part is obvious.

(ii) Whenever S' is subsequence of S, the algorithm will figure it out as well: suppose that $i_1, i_2, \dots, i_m$ are such that

$$(s_{i_1}, s_{i_2}, \dots, s_{i_m}) = (s'_1, s'_2, \dots, s'_m)$$

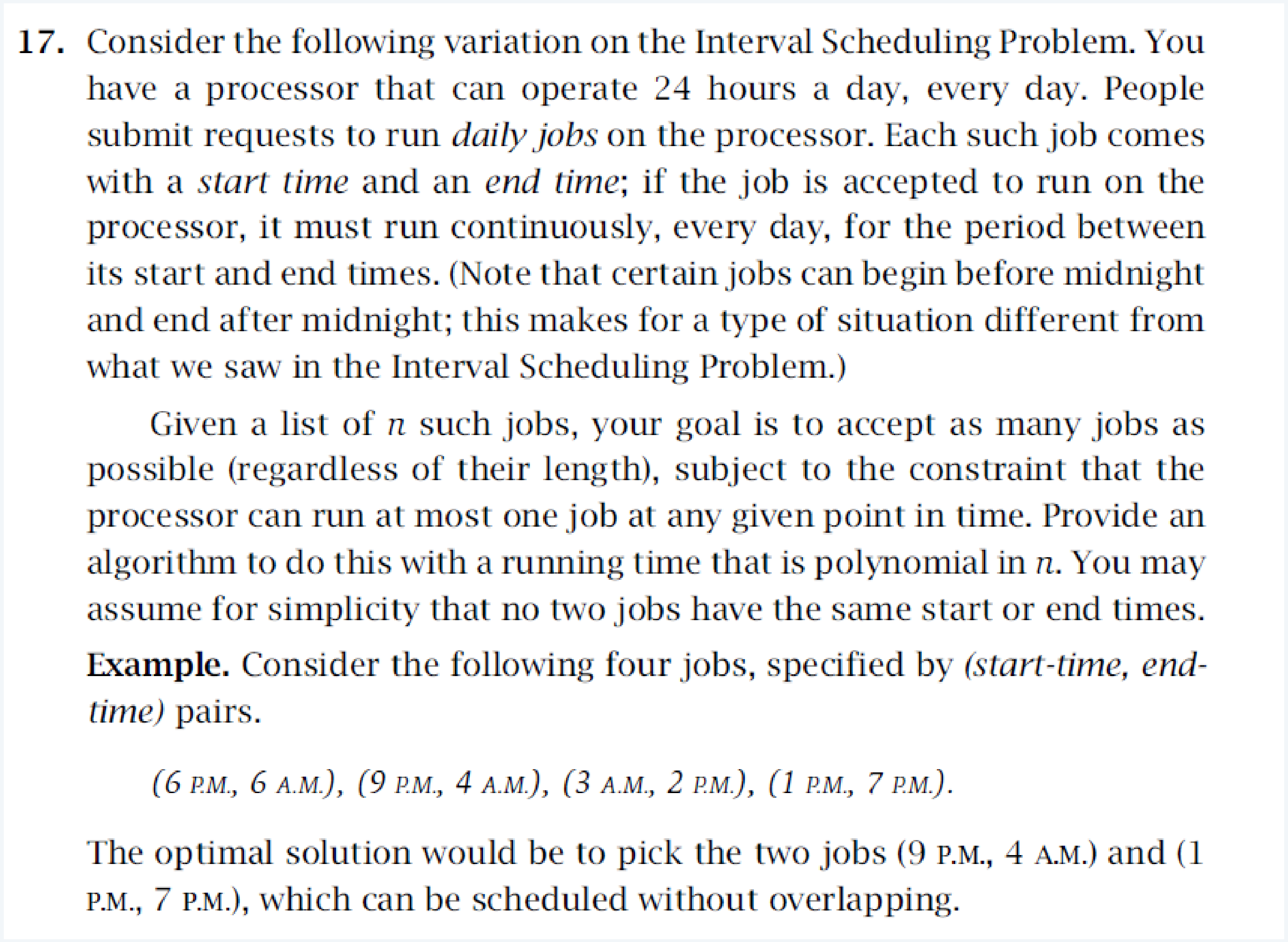
Greedy algorithms

Use induction to prove the following:

"In the while loop, when $p = i_\ell$ then $p' \geq \ell$."

Hence, when the while loop terminates, if $p = n+1$, then $p' = m+1$ as well.

Greedy algorithms



17. Consider the following variation on the Interval Scheduling Problem. 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. 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.

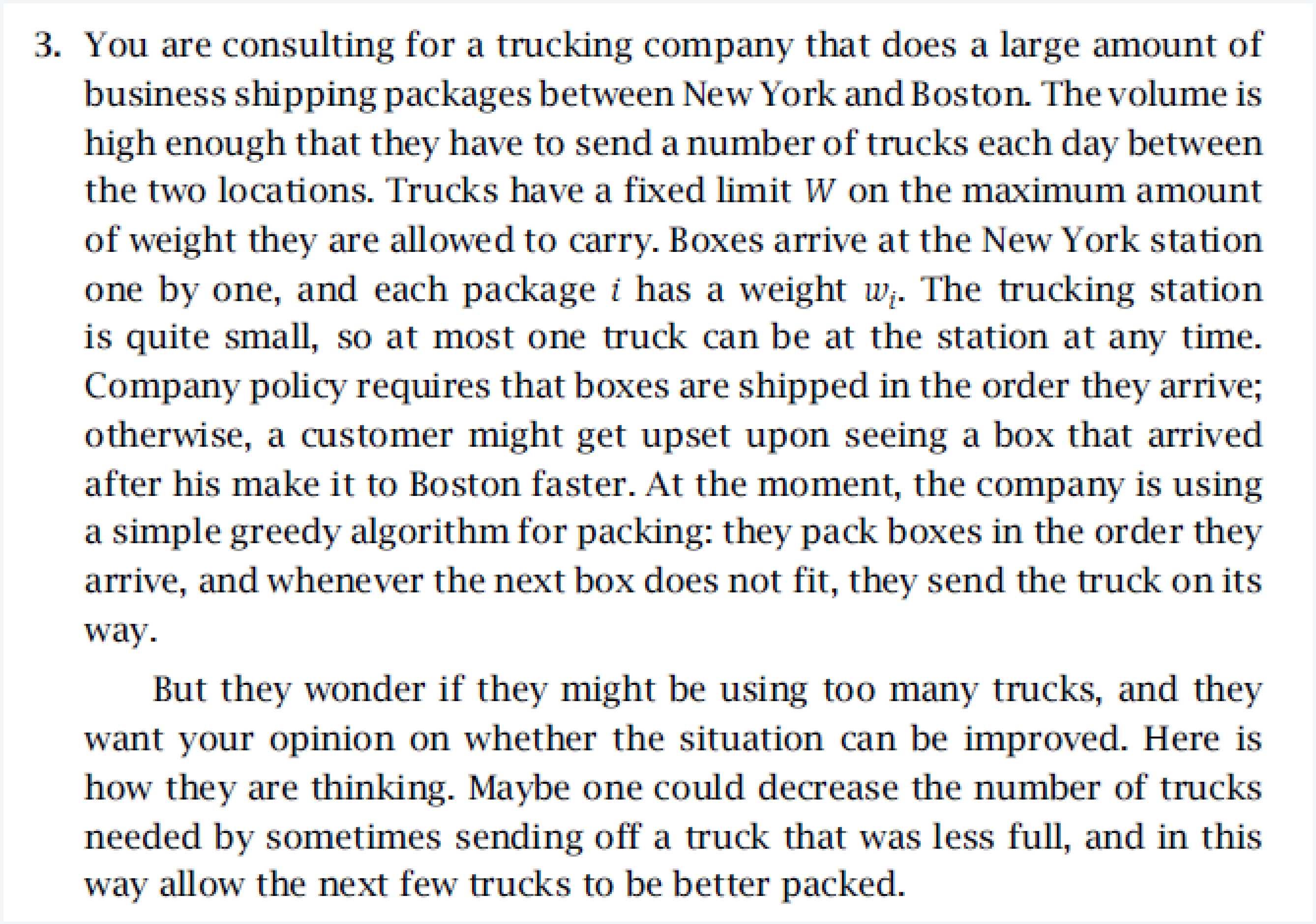
(6 P.M., 6 A.M.), (9 P.M., 4 A.M.), (3 A.M., 2 P.M.), (1 P.M., 7 P.M.).

The optimal solution would be to pick the two jobs (9 P.M., 4 A.M.) and (1 P.M., 7 P.M.), which can be scheduled without overlapping.

Greedy algorithms

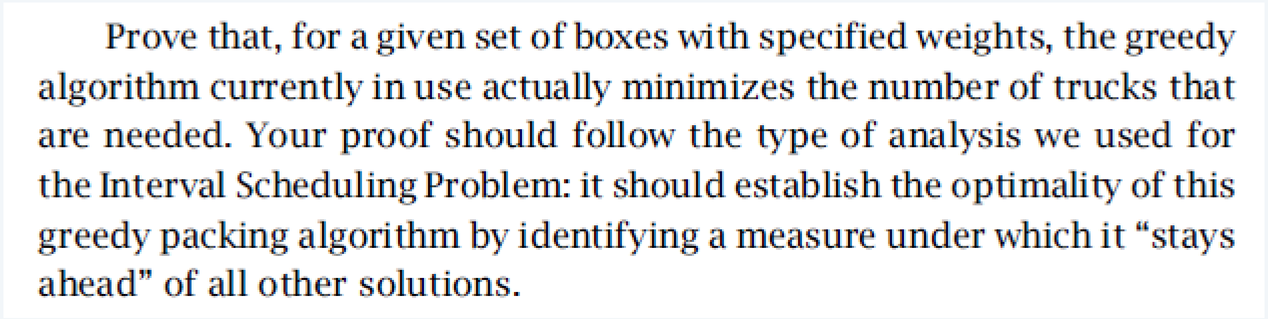
Notice that the jobs will be run continuously, there is no time 0 or end time. On the other hand, if you pick a job as part of schedule and remove all overlapping jobs, you will end up with interval scheduling problem. Hence, the idea is to pick each interval once as your starting point, and then run interval scheduling algorithm.

Greedy algorithms



3. You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box that arrived after his make it to Boston faster. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

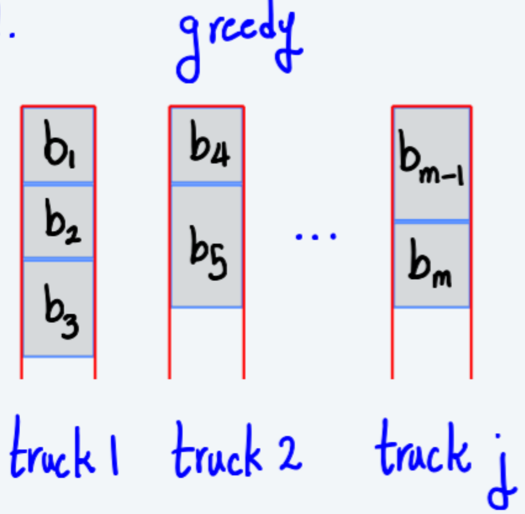
   But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking. Maybe one could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed.



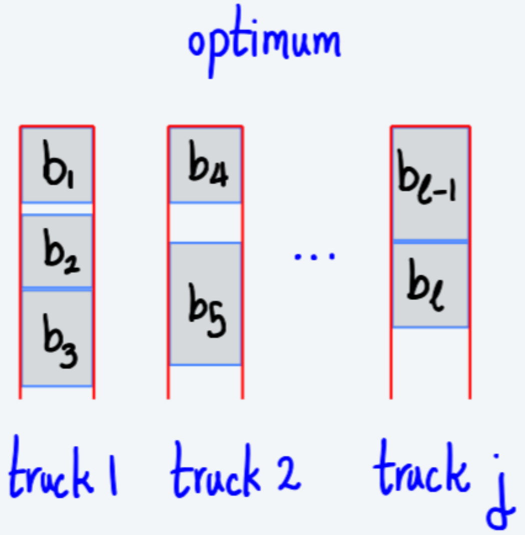
Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed. Your proof should follow the type of analysis we used for the Interval Scheduling Problem: it should establish the optimality of this greedy packing algorithm by identifying a measure under which it "stays ahead" of all other solutions.

Greedy algorithms

Suppose we have n boxes $b_1, b_2, \ldots, b_n$. Suppose that the optimum algorithm uses k trucks. Argue that the largest index of a box in jth truck in the optimum algorithm is always smaller than or equal to the same index for the greedy algorithm.



Claim:
$m > \ell$


optimum
b_1
b_2
b_3
b_4
b_5
$\cdots$
$b_{\ell-1}$
b_ℓ
truck 1
truck 2
truck j