4.3:

Answer:

We can prove that the greedy algorithm currently in use actually minimizes the number of trucks that are needed using a similar analysis to that used for the *Interval Scheduling Problem*. We will establish the optimality of this greedy packing algorithm by identifying a measure under which it "stays ahead" of all other solutions. Namely, it "stays ahead" by packing more boxes into fewer trucks.

Suppose the greedy algorithm fits boxes $b_1, b_2, ..., b_j$ into the first k trucks. Now assume the other solution fits $b_1, b_2, ..., b_i$ into the first k trucks. We will induct on the number of trucks k to prove that $i \leqslant j$. Note this establishes the optimality of the greedy algorithm because it packs the most boxes into k trucks, thereby minimizing the number of trucks.

Basis Step:
k = 1. By definition, the greedy algorithm fits as many boxes as possible into the first truck. $\Rightarrow$ basis step true.

Inductive Step:
Inductive Hypothesis: Assume the greedy algorithm packs j' boxes into the first $k-1$ trucks, the other algorithm packs i' boxes into the first $k-1$ trucks, and that $i' \leqslant j'$.
For the kth truck, the other solution packs in the additional boxes $b_{i'+1}, b_{i'+2}, ..., b_i$ into the truck. But since $i' \leqslant j'$, we know that the greedy algorithm can fit at least $b_{j'+1}, b_{j'+2}, ..., b_i$ additional boxes into the kth truck.

The result follows by the principle of mathematical induction and we have proved the optimality of the greedy algorithm currently in use by the trucking company.

4.7:

Answer:

We can prove using an exchange argument that the following polynomial-time algorithm finds a *schedule* with as small a *completion time* as possible:

Run the jobs $J_1, J_2, ..., J_n$ *in order of decreasing finishing time* f_i.

Let G be the resulting schedule. The exchange argument will prove that G is an optimal schedule. Note that regardless of the order, the last job is handed off to a high-end PC at the same time. Hence, this algorithm puts the job with the shortest finishing time as the last job in the resulting schedule G.

We want to prove that for any schedule $O \neq G$ that we can convert O into G without increasing the completion time, by swapping adjacent jobs to gradually modify O. Assume O has a different order than G. By the definition of G, O must contain an inversion. Namely, O must contain two jobs J_i and J_j such that J_i runs directly before J_j but $f_i < f_j$. We can optimize O by swapping J_i and J_j. Suppose O' is a schedule in which we swap J_i and J_j. Note that the finishing times of all jobs except J_i and J_j stay the same.

In O', J_j now schedules earlier and will finish earlier than in O. The job J_i schedules later, but the supercomputer hands off J_i to a high-end PC in O' at the same time it would've handed off J_j in O. But the job J_i will finish earlier in O' than J_j would've finished in O. Hence O' does not have a greater completion time than O. By repeatedly performing such swaps on inversions and hence decreasing the number of inversions, we can therefore convert O into G without increasing the completion time. Hence the completion time for G is not greater than the completion time for any arbitrary schedule O, and G is an optimal schedule.

4.12:

Answer:

(a)

This claim is false. We can prove this using a counterexample. Suppose we have stream 1 with $t_1 = 1$ and $b_1 = 50$, stream 2 with $t_2 = 1$ and $b_2 = 10$, and link parameter $r = 40$. Then the claim is false for stream 1. But there exists a valid schedule that runs the streams in the order stream 2, stream 1.

(b)

The algorithm to determine whether there exists a valid schedule can be described as follows. We assume that the streams are given and sent in increasing order of rate $r_i = \frac{b_i}{t_i}$. We claim that if the inequality $\sum_{i=1}^{n} b_i \leqslant r \sum_{i=1}^{n} t_i$ succeeds, the order gives a valid schedule. But if the inequality fails, no ordering produces a valid schedule. This is because in a total time of $\sum_{i=1}^{n} t_i$ we need to send $\sum_{i=1}^{n} b_i$ bits regardless of the order. We claim that the inequality also fails if the order sends too many bits for any initial time period $[0, t]$. Consider a time t and suppose i is the stream sent during the last time period. If $r_i \leqslant r$ then all streams sent so far have a rate at most r and the total sent is at most rt. But this contradicts the assumption that we sent too much in t time. So $r_i > r$. In any time period after t we also will send at least r bits, and therefore the total rate at the end of all streams also will break the rule of having an average rate of at most r. The algorithm runs in $O(n)$ time because we only need to check the one inequality $\sum_{i=1}^{n} b_i \leqslant r \sum_{i=1}^{n} t_i$ to determine if a valid schedule exists, which takes a linear amount of steps.

4.16:

Answer:

We can solve this problem using the following greedy algorithm. The strategy involves matching the n account events from the suspected bank account with the approximate time-stamps from the suspicious transactions. The approximate time-stamps can be expressed as a set of intervals $[t_i - e_i, t_i + e_i]$.

```
For i = 1, 2, ..., n
    If there are unmatched intervals containing x_i
        Match x_i with the interval ending earliest
    Else
        Return there is no perfect matching
Endfor
```

If the greedy algorithm succeeds, then we have found a perfect matching. We can prove using an exchange argument that if there is a perfect matching, our algorithm will find it. To set up our proof by contradiction, assume there is a perfect matching but that our greedy algorithm does not find it. Suppose a perfect matching S contains matches for the account events $x_1, x_2, ..., x_i$ and i is the largest number with this property. Let's suppose the next account event x_{i+1} is matched to the interval around t_l, but our greedy algorithm says x_{i+1} should match to the interval around t_j. By definition of the greedy algorithm, $t_j + e_j \leqslant t_l + e_l$. But assume t_j is matched to x_k in S, where $x_k \geqslant x_{i+1}$. Thus we know $t_l - e_l \leqslant x_{i+1} \leqslant x_k \leqslant t_j + e_j \leqslant t_l + e_l$. But this means we can do a swap and instead match x_k with the interval around t_l and x_{i+1} with the interval around t_j, resulting in a new perfect matching S' that agrees with our greedy algorithm. But S' agrees with our greedy algorithm on the first $i + 1$ account events. $\Rightarrow\Leftarrow$ The running time of this algorithm is $O(n^2)$, as required, because there are n iterations of the for loop and the step "Match x_i with the interval ending earliest" takes $O(n)$ time to iterate over the unmatched intervals.