Cody Hubbard - 004843389

CS180 - HW3

CH4 #3
Solution:

Proof by induction on k, the number of trucks needed, this will be our measure $f(\cdot)$ for how the greedy algorithm "stays ahead". Let the set $A = \{a_1, ..., a_i\}$ be the partial solutions generated by the greedy algorithm and let the set$O = \{o_1, ..., o_i\}$be the set of solutions generated by an alternate algorithm.

Let w_i be the set of i weighted packages needed to be shipped on k trucks.

case $k = 1$

This case is trivial, if only one truck is needed $f(a_i) = f(o_i) = 1$

case $k > 1$

By the inductive hypothesis $f(a_i) \leq f(o_i)$ for k trucks.

Consider the $k+1$th truck. Before this truck the greedy algorithm shipped m boxes on k trucks and the arbitrary solution shipped n boxes on k trucks where $f(a_m) \leq f(o_n)$ by the inductive hypothesis. Now consider the $(i-m)$ and $(i-n)$ boxes, by the inductive hypothesis $(i-n) \leq (i-m)$. The alternate algorithm will then ship $(i-n)$ boxes on the $k+1$th truck and the greedy algorithm will ship $(i-m)$ boxes *or more* due to the fact that the greedy algorithm fits boxes until the weight limit is reached. Effectively staying equal to or ahead of the alternative method.

CH4 #7
Solution:

Say there are n jobs $j_1, j_2, ..., j_n$ where each job j_i needs p_i seconds of computation on the supercomuter and f_i seconds on a high end PC. The optimal method to ensure the "completeion time" is the earliest time possible is to simply process the n jobs in order of largest f_i to smallest f_i. This is beacuse p_i is irrelveant, that is, the super computer can only process one job at a time and it must process all n jobs. Therefore the amount of time the supercomputer must spend computing is a constant, $\sum_0^n p_i$. Thus we base our order on the length of the f_i, and choosing them in said order ensures we obtain the lowest total completeion time.

Proof :
Suppose there is an arbitrary alternative solution with a different ordering that the one given by the greedy method. Then within the alternative ordering there exists two jobs $j_l$and j_k such that they are in a different order than the greedy method, namely suppose l comes before k in the greedy method but it is reversed in the alternative ordering. By the assumed greedy ordering it is implied that $f_l \leq f_k$ and thus by switchign them back to the greedy method's ordering will cause j_l to finish on the super computer earlier and thus finish on its high end PC earlier as well. If f_l is the task that finished last this improved the efficency, else it left it unchanged. We can repeat this method and result for as many "swapped pairs" as possibly present in the alternative solution, and thus swapping to the greedy methods ordering will always inrease efficency or leave it unchanged. This shows that the greedy method stays equal to or ahead of any alternative solution and thus proves the greedy method to give an optimal solution.

Algorithm:
Let $A[]$ be an array of all the j_i jobs.
sort $A[]$ from largest to smallest using quicksort
This algorithm simply relys on quicksort to find an optimal greedy solution. Quicksort is $O(n^2)$ and thus this algorithm is as well.

CH4 #12
Solution:

(a)

False, consider the case where the link parameter is $r = 500$. Say there are three streams of (bit b_i, time t_i) pairs: $(250, 1)$, $(1500, 2)$, $(500, 2)$. The second pair in this schedule does not satisfy $b_i \leq r \cdot t_i$ as $1500 \not\leq 500 \cdot 2$ however if you swap the third and second pair you will obtain a valid schedule thus proving the claim false.

(b)

Given a "link parameter" r, n streams each with specific number of bits b_i and a time duration t_i

Put the n streams in an array $A[]$

Sort $A[]$ by the bits per second, that is $\frac{b_i}{t_i}$ in ascending order (smallest first). This is the greedy method.

```
let $i = 0$
let $sum = 0$
for $i < n$
   add $A[i]$ to $sum$
   compare $sum$ and $t \cdot i$
   if $sum$ is greater return that there is no solution
   increment $i$
when $i = n$, return $A[]$ as a valid schedule
```

For any number of streams n this algorithm runs , at worst, in time $(1+1+n(1+1+1)) = 2+3n = O(n)$ as required.

CH4 #16
Solution:

Let $T[]$ be an array of n suspicious transactions, let $E[]$ be an array of n error margins, and let $X[]$ be an array of n recent events belonging to the account in question.

```
sort $T$ and $X$ from largest to smallest.
for $i \leq n$
  $diff = x_i - t_i$
  for each $e_j \in E$
    set $intvFound = False$
    if $|diff| \leq e_j$
        $e_i = 0$
        $intvFound = True$
    if $intvFound = False$
        return: no association exists
return: association exists
```

This algorithm will run in time $nlog(n)+nlog(n)+n(3+n(1+1(1+1)+1) = 2nlog(n) + 3n + 4n^2 = O(n^2)$ as required.