CIS 502 - Algorithms

Fall 2015 Homework 1

Problem 1 *Problem 6, page 191, of textbook.*

Solution: Lets begin with some names/notation of the input. Suppose that contestant i needs s_i units of time to swim and a t_i time to bike and run. We send the contestants in nonincreasing order of t_i.

We can prove this using standard swap arguments. Here is a different way of viewing that same proof. Suppose that the minimum time when all contestants are finished is t^*. We do not know t^*. But if player i does not leave the water by $t^* - t_i$ then the player cannot finish the race in time. This imposes a *deadline*. But we now have a scheduling problem where we want to minimize the maximum lateness (in particular we want the maximum lateness to be zero!) and we can use that scheduling algorithm. But that algorithm was to sort in the order of nondecreasing deadlines – here we do not know the deadlines. *But sorting nonincreasing order of t_i achieves the same exact ordering for any t^*.* Therefore can use the proposed sorting order above.

Running time: $O(n \log n)$ to sort.

Problem 2 *Problem 12, page 193-194, of textbook.*

Solution: Part (a) is not true. It is true that if each stream has $b_i \leq rt_i$ then there is a valid schedule – just send the streams in any order. The other direction is not true. Consider $r = 200$ and consider the (b_i, t_i) pairs: $(0, 1), (400, 1)$. We can send no bits in the first unit of time and double the number of bits in the second unit – but the second stream violates the $b_i \leq rt_i$ condition.

Part (b). We schedule the jobs in the order of nondecreasing b_i/t_i. We now argue that if there exists a feasible schedule (we will call that solution OPT – even though its just a feasibility problem) then there exists a feasible schedule following the greedy rule.

Suppose not. Consider the schedule of OPT – it clearly did not follow the greedy order since it is feasible and greedy is not. Then there exists two adjacent jobs i, j in OPT which were scheduled in the order not preferred by greedy (we saw in class the proof that violation of order $\implies$ the existence of such an adjacent pair that violates the order). In this case suppose OPT started transmitting i first at time t_0 and had already sent b_0 bits by then. Since OPT is feasible then

$$b_0 \leq rt_0 \qquad b_0 + b_i \leq r(t_0 + t_i) \qquad b_0 + b_i + b_j \leq r(t_0 + t_i + t_j)$$

And simultaneously

$$b_i t_j > b_j t_i \tag{1}$$

If $b_0 + b_j > r(t_0 + t_j)$ then $(b_0 + b_i + b_j)) - (b_0 + b_j) < r(t_0 + t_i + t_j) - r(t_0 + t_j)$. which would imply $b_i < rt_i$. But then using Equation 1 we get

$$\frac{b_j}{t_j} < \frac{b_i}{t_i} < r$$

But that would imply $b_0 + b_j < r(t_0 + t_j)$ which is a contradiction. Therefore $b_0 + b_j \leq r(t_0 + t_j)$.

Moreover if $b_0 \leq rt_0$ and $b_0 + b_j \leq r(t_0 + t_j)$ then for any $0 \leq t \leq t_j$,

$$b_0 + \frac{b_j}{t_j} t \leq r(t_0 + t)$$

(multiply the first term by $(1 - \frac{t}{t_j})$ and the second by $\frac{t}{t_j}$ and add them).

Therefore it is feasible to schedule j first and not violate the constraint. Once again since $b_0 + b_j \leq r(t_0 + t_j)$ (we just proved) and $b_0 + b_i + b_j \leq r(t_0 + t_i + t_j)$ (since OPT was feasible) we can say that for any $0 \leq t \leq t_i$,

$$b_0 + b_j + \frac{b_i}{t_i} t \leq r(t_0 + t_j + t)$$

which means we can schedule i after j and not violate the constraint. Since no other stream is changed, then the entire schedule remains feasible after the swap. Therefore we have another feasible schedule which agrees with greedy for one more pair than OPT. By repetition, greedy must itself be a feasible schedule.

Running time: $O(n \log n)$ to sort.

Problem 3 *Problem 13, page 194–195, of textbook.*

Solution: We will try deriving the sort order from the equations (of course this would work easily only in cases where the sort order is fixed - but that is the case here).

Consider some sort order and we have a pair of adjacent jobs i, j in the solution of OPT which violate the order. If job i went first and started at time t then the contribution of the two jobs to the total weighted completion time is

$$w_i(t + t_i) + w_j(t + t_i + t_j)$$

If we performed a swap to j, i (keeping everything else the same) then the contributions of all other jobs would remain the same, and the contribution of these two would be

$$w_j(t + t_j) + w_i(t + t_i + t_j)$$

The second term would be smaller or equal (this is the condition needed to prove that the swap does not make things worse) when

$$w_j(t + t_j) + w_i(t + t_i + t_j) \leq w_i(t + t_i) + w_j(t + t_i + t_j)$$

which turns out to be just $w_i t_j \leq w_j t_i$. Which means schedule in the nonincreasing order of w_j / t_j.

Remark: in this proof we derived the sort rule even when we had no idea what it should have been till the last line! This idea is often helpful if we correctly guess that (i) greedy applies and (ii) the sort order is fixed. Note that if the sort order is not fixed then the statements like "keep everything else unchanged" does not make sense.

Running time: $O(n \log n)$ to sort.

Problem 4 *Problem 15, page 196, of textbook.*

Solution: Here is the algorithm (and the proof of correctness is provided inline).

```
1: Sort everyone by the (nondecreasing) finish time. This is sequence $F$.
2: Also Sort everyone by the (nondecreasing) start time. This is sequence $S$.
3: Find the first finish time in $F$, say it is $f_i$ corresponding to job $i$. A committee member must have
   started his/her shift by then.
4: Consider all the jobs in $S$ which start by $f_i$. Among these choose the person who finishes last say $\sigma(i)$.
   Choose this person in the committee. Note that if the optimum chose differently then this person can
   oversee the same set of persons overseen by the committee member chosen by the optimum. Remember
   the last seen element of $S$ so far, say $s$.
5: Continue along $F$ and mark every person who finishes before $\sigma(i)$ as overseen.
6: repeat
7:     Consider the next person in $F$ (who is not yet overseen) – say this is $f_j$.
8:     Starting after $s$ in Step 4, find all persons who starts before $f_j$ in $S$. This person may have already
       been overseen. Remember the person who finishes last (say $\sigma(j)$) and the new value of $s$.
9:     Continue along $F$ and mark every person who finishes before $\sigma(j)$ as overseen.
10: until $F = \emptyset$
```

Running time: $O(n \log n)$ to sort the sequences. But after that observe that we are simply "walking" along the two sorted lists – that part is $O(n)$. Total $O(n \log n)$.