COT 6405 Analysis of Algorithms
Homework 3

Michael Novo

October 18, 2015

Discussed with: Nestor Hernandez

1. *Scheduling to minimize average completion time (16-2 CLRS)*

   a. In order to minimize the average completion time, we will sort tasks by their processing times, which we can do in $O(nlogn)$ time. In order to prove that our algorithm is correct, we look at how we compute average completion time. If we have two tasks a_1 and a_2, as in the question stem, we add their respective completion times c_1 and c_2 and divide by 2. We also realize that c_2 is dependent upon p_1 by the following equation: $c_2 = c_1 + p_2 = p_1 + p_2$, because the completion time of a_2 is dependent upon the processing time of a_1. So, in order to get the average, we are adding p_1 twice, which is why it makes sense to first choose the task with the smallest processing time first. Suppose we had additional tasks reaching a_n. Then, we can see that $c_n = p_1 + p_2 + \ldots + p_n$ and every other task before it depends on the processing time of the tasks before it. Therefore, in order to minimize average processing time, $\frac{1}{n}\sum_{i=1}^{n} c_i$, we must choose the first task to have the smallest processing time because the completion time of every other task (including itself) is dependent upon the first task's processing time. We also use this strategy for every other task following the first task.

   If we did not use this strategy, then we would be able to arrive at a better average completion time by replacing the first task by one that has a shorter processing time. That is, if we assume there is an optimal solution and our greedy strategy is not optimal, then we can make a switch in our greedy solution to arrive at the optimal solution. But, upon making a switch we do not get a better solution than our greedy solution because we are multiplying a slower processing time twice rather than using the faster processing time twice. This is a contradiction, so our greedy solution must yield the optimal solution.

   b. In order to minimize average completion times when we have tasks with release times and allow preemption, we use a similar strategy to the one used in part a. That is, at any given time we choose to process the task with the shortest processing time remaining. We also use an additional data structure, priority queue, to keep a running tally of each task's remaining processing time. With each new task that enters the set of tasks available, we insert into the priority queue and begin that task if its processing time is less than the task that is currently being processed. When a task is done, it is removed from the priority queue. We can do this in $O(n\log n)$.

   Assume that our greedy algorithm does not minimize the average completion time. Then, there exists another algorithm that would be able to have a shorter time by switching the position of two tasks a_i and a_k. Assume that at a given time, we have $p_i > p_k$. Our greedy algorithm would then place a_k before a_i since its remaining processing time is shorter. Our assumption says that we can switch the positions of a_i and a_k and get the optimal result. But, by switching the tasks, we get a longer average completion time. Initially, without switching, the average completion time is $\frac{1}{2}(currentTime + 2p_k + p_i)$. After switching, we get $\frac{1}{2}(currentTime + 2p_i + p_k)$. But, p_i is greater than p_k, so the switch cannot get us an optimal result. This is a contradiction, so our greedy algorithm must be optimal.

2. *Competitive Analysis of self-orgnazing lists with move-to-front (17-5 CLRS)*

   a. If H does not know access sequences in advance, then we must proceed through the access sequence $\sigma = \sigma_1, \sigma_2, ..., \sigma_m$, which we can do in linear time $O(m)$. But, for each access we make, we may proceed down the entire list of n elements for at least one of σ_i; thus, we get $\Omega(mn)$

   b. Access is $rank_L(x)$, but then to move x to the front of the list using transpositions we need $rank_L(x)$ - 1 ,which brings us to $2{*}rank_L(x)$ - 1.

c. The cost of c_i^* should be the access time needed to get x before the list is changed to L_i, plus the number of transpositions t_i^* done after accessing x in L_{i-1}. Therefore, we obtain $c_i^* = \text{rank}_{L_{i-1}^*}(x) + t_i^*$

d. L_i has q_i inversions after processing access sequence $< \sigma_1, \sigma_2, ..., \sigma_m >$ and we know that $\Phi(L_i) = 2q_i$ and $\Phi(L_i) \geq 0 \ \forall i$ and if we are starting with the same list L_o, then $\Phi(L_o) = 0$. We can then see that a transposition will increase the potential function when the result of a transposition is an a new inversion. That is, with an additional inversion increases the potential by 2 because of $\Phi(L_i) = 2q_i$. Additionally, we see the result of a transposition will decrease the potential function by two if it results in inverting an inversion, so that $q_i = q_{i-1} - 1$.

e. Normally, the $\text{rank}_{L-1}(x)$ would be all elements preceding x plus x itself (or 1). Then, for the partitions above, we should seek to find all elements preceding x in L_{i-1}, which should be composed of the elements preceding x in L_{i-1} and in L_{i-1}^*, or $|A|$, together with the elements that precede x in L_{i-1} and follow x in L_{i-1}, or $|B|$. A similar argument holds for $\text{rank}_{L_{i-1}}^*$ in terms of the partitions above.

f. After we move x to the front of the list, x now forms inversions with $|A|$, since it is now in front of those elements. It then reduces the inversions that it had with $|B|$ since the position of x with respect to $|B|$ is now reversed. Additionally, there are additional inversions caused by heuristic H denoted by t_i^*. Therefore, we reach the change in potential of:
$\Phi(L_i)$ - $\Phi(L_{i-1}) \leq 2*(|A| - |B| + t_i^*)$.

g. $\hat{c}_i = c_i + \Phi(L_i)$ - $\Phi(L_{i-1})$, which is given. From part f we can obtain:
$\hat{c}_i = c_i + 2*(|A| - |B| + t_i^*)$. We also know $c_i = 2*\text{rank}_{L_{i-1}}(x)$ -1. Then,
$\hat{c}_i = 2*\text{rank}_{L_{i-1}}(x)$ -1 + $2*(|A| - |B| + t_i^*)$. From part e we can make a substitution for $|B|$ to obtain:
$\hat{c}_i = 2*\text{rank}_{L_{i-1}}(x)$ -1 + $2*(|A| - (rank_{L_{i-1}} - |A| - 1) + t_i^*)$
$\hat{c}_i = 4|A| + 2t_i^* + 1$, and
$\hat{c}_i \leq 4(|A| + |C| + 1) + 2t_i^* + 1$, from part e
$\hat{c}_i \leq 4(c_i^*)$, because of part e and $c_i^* = \text{rank}_{L_{i-1}}(x) + t_i^*$

h. Cost of move to front is $\sum_{i=1}^{n} c_i$, but from above we found the amortized cost to be bounded above by $4c_i^*$, so then we have cost of move to front $\leq 4C_H$