Introduction to Algorithms	**Problem Set 2 (3 problems)**
CS 4820 Fall 2018	**Due 11:59pm Thursday, September 6**

Your homework submissions need to be typeset (hand-drawn figures are OK). See the course web page for suggestions on typing formulas.

The solution to each question needs to be uploaded to CMS as a separate pdf file. To help provide anonymity in your grading, do not write your name on the homework (CMS will know it's your submission). If you collaborated with other students, please **write down the collaborators in a separate file and submit that as a separate upload**.

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size.

(1) Earliest Deadline First In the class on Friday, Aug 31st, we showed that the Earliest Deadline First scheduling algorithm minimizes the maximum lateness. The problem gives with n jobs, each with deadline d_i and a processing time t_i. If job i finishes at time C_i (completion time) than its lateness is $\max(0, C_i - d_i)$, that is, its not late if $C_i \leq d_i$ and otherwise late by $C_i - d_i$ amount. We showed that the Earliest Deadline First scheduling algorithm minimizes the maximum lateness, i.e., minimizes the value $\max_i \max(0, C_i - d_i)$ among all schedules.

Minimizing maximum lateness is often not the right objective function. Two alternative objectives that may describe the goal better are

(i) minimize the sum of all latenesses: using the notation above, this objective is to minimize $\sum_i max(0, C_i - d_i)$.

(ii) minimize the number of jobs that are late: using the same notation, minimize the number of jobs i with $C_i > d_i$.

(a) Consider the objective (i). Either prove that the Earliest Deadline First greedy algorithm is guaranteed to produce the optimal schedule also for this objective, or give an example to show that this is not the case.

(b) Consider objective (ii). It makes sense to modify the Earliest Deadline First greedy algorithm by delaying all jobs that won't complete in time until the very end (as the objective is not sensitive to the amount of lateness). So, the modified version of the algorithm is now as follows:

- Order the jobs by deadline.
- Consider them in this order: when it's job i's turn, check if i can finish by its deadline, do it if it can, and move to the next job, if it cannot.
- Once all deadlines passed, process all the late jobs.

Either prove that this algorithm is guaranteed to produce the optimal minimizing the number of late jobs, or give an example to show that this is not the case.

(2) Mentor Assignment You are managing a company with n regular employees, and you just hired some $m < n$ students for summer internships. You would like to pair each new student with one of the n employees as a mentor, where each worker can mentor at most one new student. With the summer being vacation time for many, you need to be aware that not all employees are available for monitoring all the time. For each employee i you asked them for an interval of days $[s_i, f_i]$ that they are available. You would like the assignment to be such that for all new students, if student j starts on day a_j, and is assigned mentor i than $s_i \leq a_j \leq f_i$. So the input to your problem consists intervals $[s_i, f_i]$ for the workers, and start date for the students a_j.

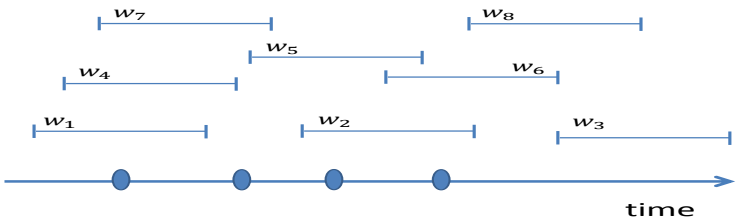

Figure 1: Time intervals of $n = 8$ workers, with dots on the time axis indicating the arrival times of $m = 4$ students.

In the example depicted on the Figure above, we can pair the students in arrival order with workers w_1, w_7, w_2, and w_6.

Give an $O(mn)$ polynomial time algorithm that assigns each new student to a mentor satisfying the requirements above, whenever such an assignment is possible. You may assume that $\log n \leq m$.

Karma question: For extra challenge, you can do this in $O(n \log n)$ time, but this part will not be graded.

(3) Priority Scheduling. You are helping your friend decide on how to schedule jobs in his operating system. Here is a simple version of this problem. You have n jobs, where job i requires time $t_i > 0$ to complete, and has an importance or priority value w_i (with higher weight being more important). You are asking for a simple policy of what order to do these jobs. Let C_i denote the finishing time of job i. For example, if job j is done first, its finishing time will be $C_j = t_j$; if it is done right after job i, we would get $C_j = C_i + t_j$.

You would like to order jobs, so that the important jobs will complete quickly. More concretely, you would like to order the jobs so as to minimize the weighted sum of completion times, i.e., to minimize $\sum_{i=1}^{n} w_i C_i$. Note that there are no deadlines in this problem, unlike problem 1 above.

For example, if we have two jobs with $t_1 = 2$ and $w_1 = 8$, and $t_2 = 5$ and $w_2 = 6$, then doing job 1 first, we get completion times $C_1 = 2$ and $C_2 = 2 + 5 = 7$, and the sum $\sum_{i=1}^{n} w_i C_i = 8 \cdot 2 + 6 \cdot 7 = 16 + 42 = 58$. Ordering the jobs with job 2 first, will get us $C'_1 = 7$ and $C'_2 = 5$ and $\sum_{i=1}^{n} w_i C'_i = 8 \cdot 7 + 6 \cdot 5 = 56 + 30 = 86$, which is worse.

They are considering three different greedy algorithms:

(a) order jobs in increasing order of the time they require, i.e., do short jobs first.

(b) order jobs in decreasing order of weight, i.e., do jobs for important customers first.

(c) order jobs in decreasing order of the ratio of the weight to the time, w_i/t_i.

Assume for simplicity that all jobs require different amount of time, all customers have different weights, and all ratios w_i/t_i are different.

For each of these proposed greedy algorithms, either prove that the algorithm is guaranteed to produce the optimal schedule (minimizing the total $\sum_{i=1}^{n} w_i C_i$), or give an example to show that this is not the case. (You do not have to consider how to implement the greedy algorithms, and do not have to analyze running time).