CS 4820, Spring 2017 **Homework 2, Problem 3**
Name: Yuxiang Peng
NetID: yp344
Collaborators: jl3455, zl542

(3) *(10 points)*
In class, we have seen several examples of optimization problems. These problems come with a notion of what constitutes a solution, and they come with an objective function that measures the quality of a solution. The goal is to find a solution that minimizes (or maximizes) the objective function. For example, in the minimum spanning tree problem, solutions are spanning trees of the given graph and the objective function is the total edge weight of a spanning tree. In the scheduling problem to minimize lateness, solutions are valid schedules for the given requests and the objective function is the maximum lateness of a request.

This way of framing the task of algorithm design — that problems have predefined, mathematically precise objectives, and that the the algorithm designer's job is to solve those predefined problems — is convenient from the standpoint of teaching lectures on algorithms or assigning exercises, but it isn't always an accurate reflection of how algorithm design takes place in real life. In many cases, there is more than one objective function that may be deemed appropriate for the problem at hand. For example for spanning trees, an alternative objective function could be the maximum weight of an edge of the tree (instead of the sum of the weights), whereas for valid schedules, we could consider the sum of the latenesses of requests (instead of the maximum lateness).

What is the right objective function? The choice of the objective function is often guided (and sometimes misguided) by applications and heuristic reasoning. However, it is important to take algorithmic considerations into account. For some objective functions the problem may be hard to solve, whereas for other objective functions, there may be efficient algorithms to solve the problem. In such cases, it often happens that the precise formulation of the objective is dictated by the choice of algorithm and not the other way around.

The following problems study the interaction between algorithms and objective functions.

(3.a) Consider the scheduling problem in Section 4.2 of the textbook. Suppose the goal is to minimize the sum of the latenesses of requests. Show that for this objective function, the earliest-deadline-first algorithm does not always find an optimal schedule.

(3.b) Again consider the scheduling problem in Section 4.2 of the textbook. Suppose every request has a positive weight w_i and the goal is to minimize the weighted sum of latenesses $\sum_i w_i \ell_i$.

Give an efficient algorithm for the special case that all deadlines are equal to the time the resource becomes available (i.e., $d_i = s$ for all $i \in \{1, \dots, n\}$).

(Solution: 3.a) Consider the following example: p stands for processing time, d stands for deadline.
p1=10, d1=3
p2=5, d2=4
According to earliest deadline first, we schedule job 1 then job 2. The total lateness is 18. If we schedule job 2 first, the total lateness is 13. Therefore earliest deadline first fails this case.

(Solution: 3.b) Rank the ratio of $\frac{w_i}{p_i}$ in decreasing order, schedule the job in this order without idle time in between jobs.

Proof: suppose there is an optimal schedule that does not follow this decreasing ratio order. There must be a pair of adjacent jobs a and b such that a is in front of job b but $\frac{w_a}{p_a} \leq \frac{w_b}{p_b}$. Let p be the total processing time before job a. If we swap the order of job a and b, the weighted lateness of other jobs do not change. The sum of weighted lateness of job a and b before swapping is $S_{ab} = w_a * (p + p_a) + w_b * (p + p_a + p_b)$. The sum of weighted lateness of job a and b after swapping is $S_{ba} = w_b * (p + p_b) + w_a * (p + p_a + p_b)$. $S_{ab} - S_{ba} = w_b * p_a - w_a * p_b \geq 0$. If it is strictly greater than 0, it contradicts the claim that the original schedule is optimal.

Runtime:
Sorting of the ratios could be efficiently done(eg: mergesort with O(n log n)). After sorting, the order is revealed. This is clearly a polynomial time algorithm.