Introduction to Algorithms	Problem Set 2
CS 4820, Spring 2014	Due 11:59pm Thursday, February 6

Hand in your solutions electronically using CMS. Each solution should be submitted as a separate file. For multi-part problems, all parts of the solution to that problem should be included in a single file.

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.

This problem set contains optional "karma exercises." These exercises have educational value but you should not turn in solutions to them, and you cannot receive credit for them (only karma).

(1) *(10 points)* Consider the following candidate algorithm for computing a minimum weight spanning tree of a graph with n vertices and m edges.

The algorithm builds a subgraph proceeding in rounds and starting from the empty subgraph. In each round, for each connected component of the current subgraph, the algorithm selects an edge of minimum weight that leaves the component. At the end of the round it adds these selected edges to the subgraph. The algorithm stops when the subgraph is connected.

(1.a) Show that the subgraph computed by the algorithm is a minimum weight spanning tree of the input graph. You may assume that the input graph is connected and that all edge weights are distinct.

(1.b) Show how to implement the algorithm in time $O(m \log n)$.

Warning: *This minimum spanning tree algorithm is moderately well known, and it is possible to find analyses of it on the Web. If you use Web sources you must cite them, and citing a Web source that contains an analysis of this algorithm will result in loss of points. For this reason, you are advised to solve the problem using your textbook and lecture notes (and office hours or Piazza, if desired), but not consulting the Web.*

(2) *(10 points)* Solve Exercise 4.16 in Chapter 4 of the Kleinberg-Tardos textbook.

Hint: The simplest proof of correctness that we know of uses an exchange argument.

Karma exercise: Show how to implement your algorithm in time $O(n \log n)$.

(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 $i \in \{1, \ldots, n\}$).

Karma exercise: One of the remarkable features of the minimum spanning tree problem is its robustness: in a very strong sense, the minimum spanning tree is optimal for any reasonable choice of the objective function, not just for the sum of the edge weights. In this sense, the minimum spanning tree problem behaves very differently from the scheduling problem addressed in problems 3a and 3b above.

More precisely, suppose f is a function that assigns a value to every $(n-1)$-tuple of numbers, and suppose

- f **is symmetric:** its value is invariant under permuting the elements of the $(n-1)$-tuple;
- f **is monotone:** its value never decreases when we increase one element of the $(n-1)$-tuple while holding the other elements fixed.

Examples of symmetric monotone functions are the sum, the maximum, and the median.

For any such f, we can consider the "f-minimum spanning tree problem" in which the goal is to find a spanning tree that minimizes the value of f, when it is applied to the $(n-1)$-tuple of edge weights in the tree. Prove that any minimum spanning tree of an edge-weighted graph is also an f-minimum spanning tree for *every* symmetric, monotone objective function f.