CSC375F **Problem Set 2** **Fall, 2009**

Due: Wed, November 4, beginning of lecture

NOTE: Each problem set only counts 5% of your mark, but it is important to do your own work (but see below). Similar questions will appear on the first term test. You may consult with others concerning the general approach for solving problems on assignments, but you must write up all solutions entirely on your own. Anything else is *plagiarism*, and is subject to the University's Code of Behavior. You will receive 1/5 points for any (non bonus) question/subquestion for which you say "I do not know how to answer this question". You will receive .5/5 points if you just leave the question blank.
Advice: Do NOT spend an excessive amount of time on any question and especially not on a bonus question. If you wish to spend "free time" thinking about (say) bonus questions that is fine but you should not sacrifice time needed for other courses.

1. Consider the following variant of the knapsack problem. Given a set of items $\{(w_1, v_1), \ldots, (w_n, v_n)\}$ and total knapsack weight bound W, a feasible solution $S \subseteq \{1, 2, \ldots, n\}$ is a set (of item indices) such that

   - $\sum_{i \in S} w_i \leq W$ and
   - if $i \in S$, then either $i - 1 \notin S$ and/or $i - 2 \notin S$.

   As a special case, if $2 \in S$ then $1 \notin S$.
   Provide a dynamic programming solution (i.e. semantic and computational arrays) for computing the value of an optimal solution and give the time bound for your algorithm as a function of n and W.

2. In tutorial, it was sketched how Knuth's analysis of the optimal binary search tree dynamic programming algorithm results in time complexity $\Theta(n^2)$. Using any sources you can find, explain in your own words how to obtain this $\Theta(n^2)$ time bound.

3. The goal of this question is to show how to utilize the $O(n^{2s})$ time DP for makespan with s job sizes into a PTAS algorithm for the makespan problem on identical machines. The general idea is to round job sizes down so that there are a small number of job sizes (namely, $\frac{1}{\epsilon^2}$) and so that the underestimation of job sizes will not add too much to the makespan computed for the rounded jobs. Throughout this exercise we are assuming that all inputs are integers. We let m denote the number of machines, and let $\{p_1, \ldots, p_n\}$ denotes the input jobs (that is, the jobs are represented by their sizes). To simplify the problem we will assume that the optimal makespan value T is known and that $\frac{1}{\epsilon}, \frac{1}{\epsilon^2}, \epsilon \cdot T$, and $\epsilon^2 \cdot T$ are all integral.

   (a) Given input $I = (p_1, \ldots, p_n)$ with say $p_1 \geq p_2 \ldots \geq p_n$, remove all jobs p_j such that $p_j < \epsilon \cdot T$ and round each remaining input job p_i down to p'_i, the next integral multiple of $\epsilon^2 \cdot T$. We let $I' = (p'_1, \ldots, p'_{n'})$ denote the rounded jobs with the "small" jobs $p_{n'+1}, \ldots, p_n$ removed. Show why we can assume there are at most $\frac{1}{\epsilon^2}$ job sizes, and why there are at most $\frac{1}{\epsilon}$ jobs on any machine in a feasible schedule with makespan T on input I'.

(b) Show that the actual makespan for the unrounded jobs $p_1, \ldots, p_{n'}$ is at most $(1+\epsilon)T$ if T is a feasible makespan for I'.

(c) After scheduling all the"large" jobs in I', greedily schedule the small jobs $p_j < \epsilon T$. Show that if T is a valid makespan for the original input set I, then there will always be a machine with makespan at most T on which to schedule each small job and that the resulting makespan for input set I is at most $(1+\epsilon)T$.

(d) Conclude (under the given assumptions and using the optimal DP for a bounded number of job sizes as a black box) that there is a $(1+\epsilon)$ approximation algorithm for the makespan problem that runs in time $O(n^{\frac{2}{\epsilon^2}})$.

4. (a) Consider the following problem: We are given n points $X = \{x_1, \ldots, x_n\}$ in Euclidean d-dimensional space and constants $\delta > 0$ and $c \geq 1$ such that the n points satisfy the condition that in any axis oriented d-dimensional cube with length δ along each side, there are at most c points in the cube. The desired result is the set of all pairs of points (x_i, x_j) in X such that the Euclidean distance between x_i and x_j is at most δ. Use a divide and conquer algorithm to compute the desired result in time $O(n \log^{d-1} n)$ where the "big O" notation can be hiding factors involving c and d.

(b) Given n points $X = \{x_1, \ldots, x_n\}$ in Euclidean d-dimensional space, show how to compute the closest pair of points in X in time $O(n \log^{d-1} n)$.