Problem Set 1

CS 331H

Due Tuesday, February 9

General rules:

- For full credit, you must give proofs of every step.
- Either typeset your solutions or write in clearly legible handwriting. The TA *will* dock points for bad handwriting.
- Collaboration is encouraged, but you must write up the solutions on your own and acknowledge your collaborators at the top of your solutions.

1. Recursive time bounds: give a big-O bound for $T(n)$ given each of the following recursive formulas:

   (a) $T(n) = 2T(n/4) + n \log n$

   (b) $T(n) = 2T(n/4) + \sqrt{n}$

   (c) $T(n) = 3T(n/2) + n$

   (d) $T(n) = T(2n/3) + T(n/3) + n/6$.

2. Suppose that you have n jobs that you would like to schedule. Each job takes a different duration of time $d_i > 0$ to complete, and you must pay a cost $c_i > 0$ for every unit of time that it is not completed. You can only work on one job at a time, but you can choose an arbitrary order among the jobs. How do you find the schedule of minimum cost?

   For a given order of the jobs, let t_i be the time that you finish job i, which is the sum of the durations of the previous jobs and this one. Your total cost is $\sum_{i=1}^{n} c_i t_i$.

   (a) (No response necessary) Think about this problem on your own for 10 minutes before reading the spoilers below.

   (b) Suppose that $n = 2$. What order should you take?

(c) Consider any ordering among the jobs for general n, and look at any pair of adjacent jobs in that ordering. How would the total cost change if you swap the ordering?

(d) Observe that repeatedly applying the idea in the previous part would lead to an $O(n^2)$ time bubble sort of the jobs, based on some function $f(c_i, d_i)$ of each element.

(e) Give an $O(n \log n)$ time algorithm for the problem.

3. Consider an interval scheduling problem where we have multiple machines, and each interval can specify which machines it can run on. That is, each job i contains (s_i, f_i, M_i) where $M_i \subseteq [n]$ is a subset of machines and the set of intervals scheduled on each machine must be disjoint.

   (a) Show that determining whether at least k jobs can be scheduled is NP-hard via a reduction from vertex cover.

   Hint: Associate each edge $e_i = (u_i, v_i)$ with the job $(i, i, \{u_i, v_i\})$, and each vertex with the job $(1, n, \{i\})$.

   (b) Now suppose that every job is associated with exactly 2 machines. Show that determining whether at least k jobs can be scheduled is still NP-hard.

   (c) Show that the above problems are NP-complete.

4. In this problem, we consider interval scheduling on a *tree*. We are given a rooted binary tree T of size m. For a node $x \in T$, let $p^k(x)$ denote the kth ancestor of x. We are given n jobs that correspond to paths down the tree: each one can be written as (s_i, k_i) for $s_i \in T$ and $k_i \in [m]$, and scheduling the ith job fills up the k_i nodes walking up from s_i:

$$\{s_i, p(s_i), p^2(s_i), \ldots, p^{k_i - 1}(s_i)\}.$$

   Each node can only be used for one job. You would like to find the maximum number of jobs that can be concurrently scheduled.

   If T were a line, then this devolves to standard interval scheduling and can be solved in linear time (after sorting, but sorting is linear in m using a counting sort). How fast can you solve it for general trees? Linear time is still possible, but partial credit will be given for any polynomial time algorithm.