Practice Problems for the Final Exam

Disclaimer: These are practice problems for the upcoming final exam. This does NOT reflect the length, difficulty, or coverage of the actual exam.

Problem 1. Consider a mergesort-like algorithm that splits a list into three equal sized lists, recursively sorts each list, and merges the three lists into a single sorted list. You may assume that the original list size is a “nice” number.

(a) How fast can you merge the three lists? Count the number of comparisons exactly.
(b) Write a recurrence for the number of comparisons your algorithm uses.
(c) Solve the recurrence. You may use the “master theorem”.
(d) How does this algorithm compare to standard merge sort?

Problem 2. The number of combinations of $n$ things taken $m$ at a time ($\binom{n}{m}$) can be computed using the following recurrence:

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \quad \text{for } 0 < m < n$$

and

$$\binom{n}{0} = \binom{n}{n} = 1.$$

(a) Write a recursive algorithm to compute ($\binom{n}{m}$) using the above recurrence.
(b) Show that the worst-case running time of your algorithm can be exponential in $n$ (depending on the value of $m$).
(c) Produce a memoized version of your algorithm.
(d) Give a dynamic programming algorithm to compute ($\binom{n}{m}$).
(e) Analyze the running time of your dynamic programming algorithm as a function of $m$ and $n$.

Problem 3. Do Exercise 3 on page 596 of Kleinberg and Tardos.

Problem 4. For each of the following give the answer “True,” “False” or “Not known to science”. Let MST denote the decision problem form of the minimum spanning tree: given a weighted graph $G$, and integer $X$ does $G$ have a spanning tree of total weight at most $X$?

(a) There is a problem $A$ that is in NP, but is neither in P nor is NP-complete.
(b) MST $\leq_P$ 3-SAT.
(c) 3-SAT $\leq_P$ MST.
(d) Suppose some known NP-hard problem can be solved in polynomial time, but this problem is not in NP. Then $P = NP$.

(e) Suppose some known NP-hard problem cannot be solved in polynomial time, but this problem is not in NP. Then $P \neq NP$.

(f) Suppose that $A \leq_P B$, and a factor-2 approximation algorithm is known for $B$. Then there is a constant factor approximation (perhaps with a different factor) for $B$.

(g) The following problem is NP-complete: Given an undirected graph $G$, does it have a clique of size 99?

**Problem 5.** Consider the following problem, called HP2: Given an undirected graph $G$, such that the number of vertices in $G$ is a power of two, does $G$ have a Hamiltonian path?

Show that HP2 is NP-complete, by showing (a) that HP2 is in NP, and (b) HP (Hamiltonian path) is reducible to HP2. Prove the correctness of your reduction. (Hint: Reduction from HP.)

**Problem 6.** An Independent Set (IS) set in a graph is a set of vertices that do not have any edges between them. Prove that the following problem, called the High-Degree Independent Set problem (HDIS) is NP-complete. Given an undirected graph $G$ with $n$ vertices and an integer $k$, does $G$ have an independent set of size $k$, which consists entirely of vertices of degree at least $n/2$? (Hint: Reduction from Independent Set.)

**Problem 7.** Prove that the following problem, called the acyclic subgraph problem (AS) is NP-complete. Given a directed graph $G = (V, E)$ and an integer $k$, determine whether $G$ contains a subset $V'$ of $k$ vertices such that the induced subgraph on $V'$ is acyclic. The induced subgraph on $V'$ is the subgraph $G' = (V', E')$ whose vertex set is $V'$, and for which $(u, v) \in E'$ if $u, v \in V'$ and $(u, v) \in E$. (Hint: Reduction from Independent Set. Think of a reduction that maps undirected edges to directed cycles.)

**Problem 8.** Consider the optimization problem of finding the longest simple path in a directed graph (LSP), where the length of a path is the number of edges.

(a) Define a decision version of the LSP problem.

(b) Show that the decision version is in NP.

(c) Show that the decision version is complete for NP (that is, it is NP-hard).

(d) Show that if you could solve the optimization version in polynomial time that you could also solve the decision version in polynomial time.

(e) Show that if you could solve the decision version in polynomial time that you could also solve the optimization version in polynomial time. HINT: First find the length of the longest simple path.

(f) Show that you can solve the decision version for constant length paths in polynomial time.