CMSC 451: Lecture EX1
Review for the Midterm

General Remarks: Recall that our overall goal in this course is to provide a understanding of the tools used in designing and analyzing efficient algorithms. So far the main topics have included graph exploration (DFS), shortest paths, greedy algorithms (including greedy approximation algorithms for NP-hard problems), and dynamic programming.

Material from Text: You are only responsible for material that has been covered in class or on class assignments. However it is always a good idea to see the text to get a better insight into some of the topics we have covered.

Cheat Sheet: Recall that the exam is closed-book, closed-notes, but you are allowed a sheet of notes (front and back). Use this sheet to write down things like:

Important formulas: Such as summations such as $\sum i$, $\sum c^i$, and facts about asymptotics.

Definitions: Definitions of concepts and important algorithmic elements (e.g., strong component, edge types in DFS, principle of optimality, negative-weight cycle, flows and cuts). It's also good to write out problem definitions (e.g., what is the k-center problem?).

Basic lemmas: Such as, "The sum of vertex degrees in an undirected graph is equal to twice the number of edges." It is a good idea to be familiar with the proofs (since I may ask you to prove results of a similar nature).

Short descriptions of algorithms: For each algorithm it is good to make note about the input format and any restrictions (e.g., a weighted digraph with nonnegative vertex weights), the output, and the asymptotic running time. It is a good idea to write down a brief description to jog your memory of the algorithm. But (given the time pressure) it is important that you "internalizing" your understanding of each algorithm. After studying an algorithm, wait at least 30 minutes and work through an example (without referring to your notes).

After you do this, then *learn* everything on your cheat sheet. If you learn it well, you won't need to consult your cheat sheet. Memorization of facts is not really important to me. (In fact, if you fail to copy down a formula, ask me and I will give it to you.) What is important is that you understand how the algorithms and proofs work, and can apply this knowledge.

I am often asked, "Will the exam problems be as hard as the homework problems?" The short answer is no, but I try to design exam problems that are similar to something we have done in class or on a homework.

Topics:

Analysis Review: Summations, asymptotics, recurrences. I may ask you to solve a summation, put functions in asymptotic order, or give a (short) proof by induction.

Graph Traversals and Applications: We reviewed basic facts about graphs and digraphs, their representations (adjacency matrix and adjacency list). We presented depth-first search (DFS) and discussed a number of applications, including topological sorting and strong components.

Greedy Algorithms: Greedy algorithms are based on the notion of building up a structure by a series of choices based on a certain *greedy order*. The main issue with greedy algorithms is proving their correctness. We saw two common approaches to proving optimality. One is based on iteratively transforming an optimal solution into the greedy solution without ever altering the cost of the solution. The other is based on computing a lower bound on the optimum, and showing that greedy achieves this lower bound.

Interval Scheduling: Compute a maximum-sized subset of nonoverlapping intervals.

Interval Partitioning: Compute a minimum-sized partition of a set of intervals into nonoverlapping subsets.

Scheduling to Minimize Lateness: Schedule a collection a tasks with duration times and deadlines to minimize the maximum lateness of the scheduled tasks.

k-Center Problem: We presents a greedy algorithm (Gonzalez's algorithm) for computing a factor-2 approximation to the k-center problem, which involves computing a set of centers to minimize the maximum distance from any point to its closest center.

Set Cover: The problem is to compute a minimum size collection of sets to cover a domain of size m. We presented a greedy algorithm, which selects the set that covers the largest number of uncovered items. We showed that the algorithm has an approximation factor of $\ln m$, meaning that the number of sets generated is larger than the optimum by a factor of at most $\ln m$.

Dynamic Programming: We presented DP solutions for the following problems. We also discussed implementation methods, such as memoization and table construction. We discussed a number of examples of DP, including

- Weighted Interval Scheduling
- Longest Common Subsequence
- Chain Matrix Multiplication
- Floyd-Warshall Algorithm for all-pairs shortest paths

Network Flow: We introduced a number of concepts, including s-t networks, flows (capacity and conservation constraints), the residual network, augmenting path, and cuts. We covered the Ford-Fulkerson algorithm and the Max-Flow/Min-Cut Theorem. (You are not responsible for the scaling algorithm or the Edmonds-Karp algorithm.)

Further Tips:

Be able to trace algorithms: You should expect a question of the form: "Trace Algorithm X on the following input" showing all the intermediate results. You should also expect a series of short-answer problems (as appear in the practice problems). Although the answers are short, some of the problems may require a lot of thought.

Understand the "internals" of solutions/proofs: The other problems will involve giving proofs, designing and analyzing algorithms. I try to ask at least a couple of questions that involve a modification to a homework problem or theorem or algorithm seen in class. In order to answer such a question, it is important that you understand the solution, not just memorize it.

Understand the problem: When under time pressure, it is tempting to quickly skim the problem, and start working on it before you really understand it. Read each question carefully. Draw a small example. If you are not sure whether you understand it, call me over to look at your example.

Approach and insight: When it comes to grading, I tend to give more credit for good *insight* or good *approach* to problem solving than for *memorization*. For example, knowing that you should apply DP in a given context, and coming up with a partially correct formulation is worth much more than copying the LCS algorithm out of your cheat sheet. I don't like tricky problems on exams, so if a problem seems insanely hard, absurdly easy, or if you think that the answer is "it can't be done," you better check with me.

Avoiding "taking the wrong track": A common problem is spending way too long on a solution that is going nowhere. Keep your eye on the time, and don't spend too much time on any one problem. Also, if you are worried that you are not on the right track, you can call me over and ask whether your approach is sound. I can give either a "thumbs-up", meaning your general approach is correct (but I won't comment on the correctness of the details) or "thumbs-down", meaning that your approach is wrong.