CISC320 algorithms, 10S, final exam review notes

First half (Chapter 0 through 4.4 is summarized on the midterm review sheet. this sheet continues from there.
Chapter 4, sections 4.5-4.8:
Key points:
Consider Dijkstra's single source shortest path algorithm on a weighted directed graph of n vertices and m edges with no negative edge weights.

- It uses a priority queue which must be
  1. built on n vertices (using n insert()'s for instance), then
  2. experience n extract-min()'s as the algorithm proceeds.
  3. experience a total of up to m decrease-key operations as the distances to neighbors are updated for each vertex processed.
- A binary heap may be used to give O($m\log(n)$) time for Dijkstra's alg.
- A d-ary heap, for d about m/n performs better, especially as m approaches n^2. (pg 114)

When negative edges (but no negative cycles) are involved, the Bellman Ford algorthm works in O(mn) time.
Chapter 5, greedy algorithms:

- Minimal spanning trees: Kruskal's and Prim's algorithms are both greedy style, but with different strategies of "best greedy choice".
- Prim's is very similar to Dijkstra's algorithm for a different problem, uses a priority queue. It gradually builds a MST as a growing tree.
- Kruskal's algorith considers edges in increasing order by weight, Needs a union-find dynamic disjoint set implementation. It manages a forest of trees that are gradually joined to form the MST.
- Union-find with the union by weight heuristic performs n union() and find() operations in total time O($n\log(n)$).
- Union-find with also the path-compression heuristic performs n union() and find() operations in total time O($n\log^*(n)$).
- $\log^*(n)$ is a very slow growing function. For instance, $\log^*(n) \leq 5$ for any $n <$ google^2, where google $= 10^{100}$.
- Huffman codes for data compression: variable length bit codes for letters are designed greedily based on frequency of occurrence of the letters of the alphabet used.
- Importantly the bit codes have the prefix property. Know what this means and why it matters.
- Provably the Huffman codes are optimal over all variable bit codes with the prefix property.

Chapter 6, Dynamic Programming:
Strategy: Find a function of a set of parameters that

1. Has a clear conceptual definition
2. Has a recurrence relation expressing it's value at parameters in terms of other values at parameters in which at least one is decreased.
3. Solves the given problem for some set of parameter values.

Then figure out the table of values needed and the order of construction.
The run time cost is the size of the table times the cost to compute one entry. The latter is typically dominated by the number of other entries looked up to compute the given entry.

Memoization can do the table work for you. In the recursive function do: (1) first see if the parameters hash to a table entry already computed. If so, use it. Otherwise compute the value according to the recursive definition and put it in the table just before returning it.

Many problems yield good dynamic programming solutions.

- Longest increasing subsequence
- Edit distance (note relevance to bioinformatics)
- Knapsack in O(nW)
- chain multiplication in O(n^3)
- Traveling Salesman in O($n^2 2^n$)
- The many chapter-end exercises

Chapter 8 NP-complete problems:
Classes of problems

- P - search problems having an algorithm running in O(n^c) time for some constant c on problem instances with input of size n. Sometimes problems in P are called "tractable". A problem not known to be in P is "hard".
- NP - search problems having an "checker" algorithm C(x, y) that verifies if y is a solution for instance x in polynomial time.
- Reduction: Show problem A has an algorithm that works by calling an algorithm for problem B together with polynomially much additional work. If A is a hard problem, reduction to B is a way to show that B must also be hard.
- NP-complete - problems in NP to which all problems in NP can be reduced.

Hard problems:

- SAT, 3SAT, circuit-SAT
- vertex cover, clique, independent set
- Travelling salesman, Rudrata cycle, Rudrata path
- Knapsack, subset sum

- ...

Examples of reductions about which exam questions could be asked

- Traveling salesman to Rudrata cycle
- Rudrata cycle to Rudrata path
- SAT to 3SAT
- 3SAT to Independent vertex set Bisection and doubling: Optimization problems to search problems
- Search problem to decision problem

Reduction categories (generalizations, uses of gadgets).

Cook's theorem: reduction of any NP problem to circuit-SAT.

Overall: Things to know about each algorithm:

- Why is it correct (vis a vis it's input/output specification)? What are the theorems and properties used to explain it's workings?
- What measure, n, of it's input is used as basis for analysis (for instance, n = bound on number of bits in numbers, or n = size of an array)?
- What formula (function of that n) estimates it's runtime? Usually the formula is a recurrence relation.
- What is the solution of that formula/recurrence?

Kinds of questions: multiple choice, short answer, write algorithm, track algorithm.