Midterm 3 Study Guide

This midterm covers chapters 8, 9 and 11. Here is a list of the main ideas:

1. Describe the main idea behind *dynamic programming*, and what kinds of problems it is useful for.
2. Describe what the *coin-row* problem is. How is it solved via a recurrence relation? How can we translate that into a dynamic programming algorithm?
3. Describe the *change-making* problem, and the solution to it. Note that our solution to the change-making problem was different from the book's, we used a 2-dimensional table for it.
4. How can dynamic programming help us solve the *Knapsack* problem? What are *memory functions* and what are the advantages and disadvantages of a dynamic programming approach using memory functions vs not using memory functions?
5. In a greedy algorithm, each choice made exhibits three key characteristics. What are they? Describe them.
6. What are the three different possible approaches we can take in order to prove that a greedy algorithm produces the optimal solution to the problem?
7. What is a *minimum spanning tree*?
    - How does *Prim's algorithm* proceed to construct a minimum spanning tree?
    - What data structure do we need to maintain to assist us with Prim's algorithm?
    - What is the running time of Prim's algorithm, assuming an adjacency list representation for the graph and a suitable implementation for other needed structures?
8. How does *Kruskal's algorithm* for determining a minimum spanning tree work?
    - Kruskal's algorithm needs to make use of a *union-find* structure. What is that structure? How does it work?
    - Write pseudocode for the various operations in a union-find structure.
9. Write pseudocode for Prim's and Kruskal's algorithms. Be able to use them.
10. Describe *Dijkstra's algorithm* and the problem it solves. Be able to carry out the algorithm.
11. What is the main end result of a *lower-bound argument*? What does it tell us about a problem?
12. How do we find *trivial* lower bounds?

13. Explain the idea of the *adversary* argument, and use it to demonstrate that any comparison-based algorithm that is searching through a sorted array must take at least $O(\log n)$ time.
14. Explain how *decision trees* lead us to an *information-theoretic* lower bound, and use it to estimate that searching through a sorted array would take $O(\log n)$ time.
15. What decision problems to we classify as being "in P"? What about being "in NP"? Give examples of problems that are in P and problems that are in NP.
16. What do we call a *reduction* from a problem to another? Give examples.
17. What problems do we call *NP-complete*? What is so special about those problems?
18. Name some problems that are NP-complete, and describe them.