Final Exam Study Guide

Test 2/final exam is cumulative in that all topics build on each other. However, the questions in this test will be more on the material in the second half of this course (not covered on test 1). Note that you may still need to understand the first half material as it relates to these topics, but we will not ask a written question specifically on the first half material. You might have multiple choice questions on the first half. This study guide indicates what sorts of activities you should be able to perform with each topic. You are welcome to use Pseudocode, Python, c++, or Java to solve some of the problems.

1. Turing Machine
    - Draw a Turing Machine which decides a given language.
        - Draw the state diagram (circles and arrows).
        - Write out the state transitions as a table.
2. Complexity
    - Given an algorithm, you should be able to produce a table to analyze its runtime (O) in the format that we did in class.
    - Explain the limitations of Big O (Engineer's side in Engineer vs Theoretician) and how they relate to specific scenarios.
3. Correctness
    - Given a piece of code, you should be able to state its loop invariants.
    - Given a piece of code, you should be able to state the precondition required to give partial or total correctness.
    - Given a piece of code and a precondition, you should be able to state the postcondition.
4. Divide and conquer
    - You should be able to identify when a divide-and-conquer approach is appropriate to solving a problem.
    - You should be able to write a divide-and-conquer algorithm to solve a particular problem when it is the appropriate technique.
    - You should be able to analyze the runtime (O) of a divide-and-conquer algorithm.
5. Greedy
    - You should be able to identify when a greedy approach is appropriate to solving a problem.
    - You should be able to write a greedy algorithm to solve a particular problem when it is the appropriate technique.
    - You should be able to analyze the runtime (O) of a greedy algorithm.

6. Dynamic programming

    - You should be able to identify when a dynamic programming approach is appropriate to solving a problem.
    - You should be able to write a dynamic programming algorithm to solve a particular problem when it is the appropriate technique.
    - You should be able to analyze the runtime (O) of a dynamic programming algorithm.

7. Graphs

    - You should be able to identify when a graph can solve a problem, and explain how the graph can be used to solve such a problem.
    - You should be able to modify DFS, BFS, or Djikstra's algorithm to solve a particular problem when appropriate.
    - You should be able to match a problem description to any of the following graph concepts such as topological sort.

8. Advanced data structures

    - Identify problems in which bloom filters or tries are appropriate to solve the problem.
    - Explain why a bloom filter or trie is appropriate to solve a given problem.
    - Show how a bloom filter or trie would operate in a particular situation (e.g., draw a picture of adding to a bloom filter, or searching a trie).

9. Union Find

    - You should be able to identify situations where the union find data structure is appropriate to solve a problem efficiently
    - You should be able to show how operations are carried out on a union find data structure (*e.g.*, draw a picture of what happens in a union, draw a picture of how the path is compressed during find, etc.).