1 Topics Covered

This midterm covers up through dynamic programming; the RAM model and Turing Machines will not appear on this midterm. ***Disclaimer**: This is not a comprehensive list of possible topics for the exam; in general, any material from lecture, homeworks, or sections is fair game.*

1.1 Math

- **Recurrence Relations**. We can solve simple ones by hand (or perhaps by writing out a few terms). More complicated recurrences can be solved using the **Master Theorem**.
- You should be comfortable with the definitions for O, o, Ω, ω, and Θ.

1.2 Data Structures

- Arrays
- Heaps and Priority Queues
- Van Emde Boas Trees
- Disjoint Set, including the "union by rank" and "path compression" optimizations

1.3 Algorithms

You should be able to run any of these algorithms by hand on the exam; you should also be familiar with their runtimes and/or space requirements.

- **Bubble Sort, Counting Sort**.
- **Mergesort**. Divide and Conquer, merging sorted sublists together.
- **Prim's Algorithm**. Build a tree from a single seed vertex. Uses the **Cut Property** to choose the smallest edge leaving the group.
- **Kruskal's Algorithm**. Build a tree by sorting the edges, joining disjoint groups of vertices. Uses the **Disjoint Set** data structure.
- **Repeated Squaring**. Exponentiate quickly by squaring your previous result rather than multiplying one at a time.

1.4 Algorithm Strategies

- **Greedy**. Pick the best option at each step. e.g., Prim's Algorithm

- **Divide and Conquer**. Split the problem into smaller subproblems (often in half), solve the subproblems, and combine the results. e.g., Mergesort, Integer Multiplication, Strassen's
- **Dynamic Programming**. Useful for when there are many overlapping subproblems. e.g., String Reconstruction, Edit Distance

1.5 Lower Bounds

- **Sorting**. The comparison model.

2 Practice Problems

2.1 Minimum Spanning Trees

Exercise. *Give an algorithm for determining whether a graph has a unique minimum spanning tree.*

2.2 Binary Search

In binary search, you are given a sorted list of length n and want to find the position of a given element x in that list. The search procedes by dividing the list into halves, comparing the given element against the middle element, and then continuing recursively on the left or right half, depending on the result of the comparison.

Exercise. *What is the runtime of binary search?*

Exercise. *Now, suppose that instead of dividing the list into 2 halves, you divide into k pieces of equal length, where k is a constant, then use comparisons to figure out which piece the element lies in before running the algorithm recursively on that piece. What is the runtime of this?*

Exercise. *Prove that binary search is asymptotically optimal in a deterministic comparison-based model.*

2.3 Egyptian Fractions

For any rational fraction p, it is possible to write $p = \sum \frac{1}{p_i}$ for some finite number of positive integers p_i. For instance, for $p = \frac{6}{7}$, we can write $\frac{6}{7} = \frac{1}{2} + \frac{1}{3} + \frac{1}{42}$. This decomposition can be done in many ways (infinite number of ways actually).

Exercise. *Provide an algorithm to find one such summation, given the fraction p.*

2.4 Cutting Wood

From MIT 6.006, Spring 2011

Given a log of wood of length k, Woody the woodcutter will cut it once, in any place you choose, for the price of k dollars. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and $n+1$ denote the left and right endpoints of the original log of length L. Let the distance of mark i from the left end of the log be d_i, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$.

Exercise. *Determine the sequence of cuts to the log that will (1) cut the log at all the marked places, and (2) minimize your total payment to Woody.*

2.5 Guitar Hero

From MIT 6.046, Fall 2009

You are training for the World Championship of Guitar Hero World Tour, whose first prize is a real guitar. You decide to use algorithms to find the optimal way to place your fingers on the keys of the guitar controller to maximize the ease by which you can play the 86 songs. Formally, a note is an element of $[A; B; C; D; E]$ (representing the green, red, yellow, blue, and orange keys on the guitar). A chord is a nonempty set of notes, that is, a nonempty subset of $[A; B; C; D; E]$. A song is a sequence of chords: $[c_1; c_2; \ldots; c_n]$. A pose is a function from $[1; 2; 3; 4]$ to $[A; B; C; D; E; \emptyset]$, that is, a mapping of each finger on your left hand (excluding thumb) either to a note or to the special value $\emptyset$; meaning that the finger is not on a key. A fingering for a song $[c_1; c_2; \ldots; c_n]$ is a sequence of n poses $[p_1; p_2; \ldots; p_n]$ such that pose p_i places exactly one finger on each note in c_i, for all $1 \leq i \leq n$.

You have carefully defined a real number $D[p; q]$ measuring the difficulty of transitioning your fingers from pose p to q, for all poses p and q. The difficulty of a fingering $[p_1; p_2; \ldots; p_n]$ is the sum $\sum_{i=2}^{n} D[p_{i-1}, p_i]$.

Exercise. *Give an $O(n)$-time algorithm that, given a song $[c_1; c_2; \ldots; c_n]$, finds a fingering of the song with minimum possible difficulty.*