CSE 101 Final Topics

**General Topics:**

1. **Running time analysis:** lower bounds and upper bounds, best case and worst case and average case analysis, asymptotic notation, writing and solving recurrences (but the Master Theorem will be printed on the final, so that you don't have to memorize its details), running times of algorithms covered in class.

2. **Algorithm design:** understanding new algorithms, using algorithm templates (divide-and-conquer, dynamic programming, greedy) to design algorithms, using algorithms covered in class as a part of algorithms to solve new problems, modifying algorithms from class to solve modified problems, finding counterexamples to incorrect algorithms (especially greedy, such as Exercise 17.1-3).

3. **Correctness proofs:** Proof of divide-and-conquer algorithm correctness using induction (such as the problem on the midterm, or the QUICKSORT correctness proof - but you *don't* need to know the techniques that were used to prove PARTITION correct), proving optimal substructure for dynamic programming algorithms (such as Theorem 16.1), proving greedy algorithm correctness (such as Theorem 17.1).

4. **NP:** Polynomial vs. exponential running times, complexity classes P and NP and the relationship between them, certificates and verifiers, showing problems are in NP (by producing a verifier), showing problems are NP-hard (using reductions from known NP-complete problems), showing problems are NP-complete.

**Reading:**

**Divide & Conquer, Sorting, and Selection:** Insertion Sort and Merge Sort (Chapter 1), Divide-and-conquer multiplication (see handout), Quicksort (8), Counting Sort (9.2), Selection (10).

**Dynamic Programming, Greedy Algorithms, and Graph Problems:** Dynamic Programming and Matrix Chain Multiplication (16-16.2), All-Pairs Shortest Paths (P. 514-516, P. 519, 26-26.2 except for material about the "predecessor matrix"), Greedy Algorithms and Scheduling (17-17.2), Minimum Spanning Trees (24).

**NP:** Chapter 36.

**Algorithms:** For each algorithm, you should know what problem it solves; know what it takes as input and what it produces as output. Know the running times for the algorithms, and have a basic idea of how they work (for example, "divide-and-conquer, splitting into subproblems this way...”). You don't need to memorize line-by-line details, but if you see the line-by-line pseudocode on the final, you should be able to understand the purpose of every line so that you can modify the algorithm, etc.

Note that you should also be familiar with functions required by these algorithms (such as PARTITION and EXTEND-SHORTEST-PATHS). It would also be helpful to study algorithms covered only on the homework (such as STOOGE-SORT), even though you won't be explicitly tested on details of these algorithms. For the data structures used by MST-KRUSKAL and MST-PRIM, you should know what the operations being used are doing, and how their running time contributes to the running time of the algorithm, but you don't need to know any details about how the data structures are implemented.

**Divide & Conquer, Sorting, and Selection:** INSERTION-SORT, MERGE-SORT, QUICKSORT, RANDOMIZED-QUICKSORT, COUNTING-SORT, MULTIPLY, RANDOMIZED-SELECT, SELECT.

**Dynamic Programming, Greedy Algorithms, and Graph Problems:** MATRIX-CHAIN-ORDER (and MATRIX-CHAIN-MULTIPLY), LCS-LENGTH (and PRINT-LCS), FASTER-ALL-PAIRS-SHORTEST-PATHS, FLOYD-WARSHALL, GREEDY-ACTIVITY-SELECTOR, MST-KRUSKAL, MST-PRIM (and also GENERIC-MST).

**NP-Complete Problems:** For each of these, you should know the definition of the problem, so that you can use it in reductions: 3-CNF-SAT, CLIQUE, HAM-CYCLE, TSP. You also might be asked general questions about CIRCUIT SATISFIABILITY, but won't be asked to use it in new reductions. You don't need to know all the details of the NP-completeness proofs for the above problems, but it would be good to study these proofs to help you know how to show new problems are NP-complete.