Summer 2003 CMSC 451 Instructor: Ken Hennacy

Final Study Sheet Draft 1

Problems to consider:

Linear Programming

a. Simplex Algorithm
b. Constraint graphs

P, NP, NPC

a. Mapping between decision and optimization problems.
b. NP-C concepts

Approximations

Greedy Algorithms

a. Activity Selection Problem
b. Huffman Code
c. Fractional Knapsack Problem
d. Dijkstra's Algorithm
e. Kruskal's Algorithm
f. Prim's Algorithm

Graph Concepts & Algorithms

a. Breadth First Search
b. Depth First Search
c. Topological Sort
d. Strongly Connected Components
e. DAG
f. Weighted Graphs
    a. Single Source Shortest Paths
    b. Bellman-Ford Algorithm
    c. Topological Sorted DAG Algorithm

g. Performance and Proofs
    a. Correctness of Greedy Algorithm (Huffman code example)
    b. Correctness of Graph Algorithms (Chapt 22)
    c. Performance of various Algorithms : T(n) and Size(n)
    d. Minimal Spanning Tree (Generic Algorithm Proof)

2. Recurrences
    a. Psuedocode implementation (top down and bottom up).
    b. Derivation of recurrences from first principles.
    c. Application of the Master Method (provided on test).

3. T(n) [Worst or Average Case as appropriate] for various types of sorting.
    a. Insertion Sort
    b. Merge Sort
    c. Heap Sort
    d. Quick Sort
    e. Counting Sort
    f. Algorithmic Analysis in general

4. Binary Trees
    a. Definition and use
    b. Optimization utilizing dynamic programming

5. Assembly Line Problem, Matrix Multiplication

6. Memoization

Points to consider:

1. Difference between performing cost calculations of an algorithm T(n), and cost calculations associated with an algorithm (Ex: matrix mult. & opt. binary search tree).
2. Impact of Tree Structure on T(n) for various algorithms that use trees.
3. Understanding why you have to use big O notation as opposed to θ.
4. Understanding how to split a problem up into smaller subtasks that are identical to the larger task. Adding the appropriate terms to this division process to complete a recurrence.
5. For remembering T(n) dependencies, understand the algorithm's operation, i.e. is it dependent on "divide and conquer", explicit loops, or data structures? Does it involve comparisons?
6. What assumptions are built into the algorithms (if any) ?
7. What information does each algorithm require (generate) ?
8. Why do greedy algorithms perform better than dynamic programming solutions? Under what circumstances do greedy algorithms fail?