MA/CSSE 473 – Design and Analysis of Algorithms

Homework 6 (88 points total) Updated for summer 2012

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

3.5.2 [5.2.2]	(adjacency matrix *vs* adjacency list for DFS)
3.5.7 [5.2.7]	(Use BFS/DFS to find a graph's connected components)
3.5.10 [5.2.10]	(DFS and mazes)
4.1.8 [5.1.5]	(insertion sort sentinel)
5.1.125.1.10	(Shell's sort) This should be review from 230
4.2.1 [5.3.1]	(Topological sort examples)
4.2.2 [5.3.2]	(Theoretical properties of topological sort)
4.3.1 [5.4.1]	(Reasonableness of generating all permutations, subsets of a 25-element set)
4.3.9 [5.4.9]	(Generation of binary reflected Gray Code based on bit-flipping)

Problems to write up and turn in:

1. (6) 3.5.3 [5.2.3] (independence of properties from specific DFS traversals) Explain your answers.
2. (10) 3.5.8a [5.2.8a] (Bipartite graph checking using DFS)
3. (5) 4.1.1 [5.1.1] (Ferrying Soldiers)
4. (5) 4.1.4 [5.1.3] (generate power set)
5. (5) (not in book) [5.1.9] (binary insertion sort efficiency).

   Binary insertion sort uses binary search to find an appropriate position to insert A[i] among the previously sorted A[0] ≤ … ≤ A[I - 1]. Determine the worst-case efficiency class of this algorithm. I.e. get big-Θ time for number of comparisons and number of moves.
6. (9) 4.2.6 [5.3.6] (finding dag sources) Be sure to do all three parts.
7. (9) 4.2.9 [5.3.9] (Strongly connected components of a digraph)
8. (10) 2.4.14 [5.3.10] (Celebrity identification)

   It may seem strange for this problem from chapter 2 to be in this assignment. In the 3rd edition, the author moved it from Chapter 5 to chapter 2, and I did not realize it until I was working on HW 6.
9. (9) 4.3.2 [5.4.2] (Examples of permutation generation algorithms)

   You do not have to write any code, but you can do it that way if you wish.
10. (10) 4.3.10 [5.4.10] (Generation of all k-combinations from an n-element set)
11. (10) 4.3.11 [5.4.11] (Generation of binary reflected Gray code based on Tower of Hanoi moves)