MA/CSSE 473 – Design and Analysis of Algorithms

Homework 5 **64 points total** Updated Summer, 2016

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.

3.2.2 [3.2.2]	(best/worst case for sequential search)
3.2.8 [3.2.9]	(Substrings that begin/end with specific characters) Brute force is Ɵ(N^2)
3.3.3 [3.3.2]	(closest straight-line post office)
3.4.2 [3.4.2[	(Hamiltonian Circuit)
3.4.10 [3.4.9]	(Magic Squares)
3.4.11 [XXX]	(Famous alphametic)

Problems to write up and turn in:

1. (14) 3.2.3 [3.2.3] (Gadget drop) If a gadget breaks, it cannot be repaired and used for another test.

You would do well to give yourself a few days between the time when you first start thinking about this problem and the time when you intend to submit it.

Let N be the total number of floors, and F the number of the lowest floor on which a gadget fails when dropped from there.

Assume that due to weight, volatility, or some other factor, there is a high cost (C) for each floor that a gadget must be carried up. Also a cost (T) for each test to determine whether the drop caused the gadget to fail after each drop.

First, give big-Theta worst-case costs in terms of N, C, and T for the obvious algorithm that tries every floor in succession (that algorithm only requires one gadget). Then design and analyze the most efficient algorithm you can.

Can you think of any flaws in this general approach to testing?

Points: (Algorithm/analysis: 2, efficient algorithm/analysis: 10, flaw: 2)

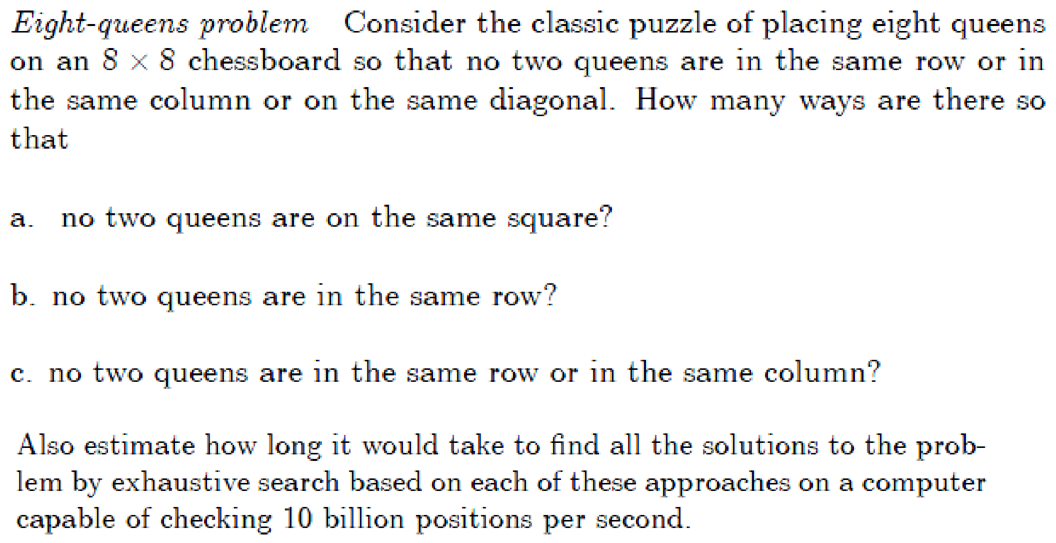
In summer, 2013, almost all students said #1 was the most time-consuming problem.

2. (2) 3.2.4 [3.2.4] (String matching count)
3. (5) 3.2.6 [3.2.6] (String matching worst case). Searching for a pattern of length m in a string of length n. Your answer for the analysis part should be a formula involving n and m, not just a big-O approximation.
4. (10) 3.3.4 [3.3.3] (Manhattan distance). Clarifications: (b) You can sketch or simply list them.
   (c) By "a solution", I interpret it to mean "an algorithm to solve the problem". Obviously Manhattan distance may give different nearest points than Euclidean distance. But here is the question you must answer: is the closest point algorithm itself the same?
   Points: (a:5 , b:3, c:2).
5. (5) 3.3.7 [3.3.5] (brute-force k-dimensional closest-point algorithm)
6. (4) 3.3.10 [3.3.8] (dealing with collinear points in brute-force complex hull algorithm)
7. (7) 3.4.1 [3.4.1] (traveling salesman analysis) Points: (a:3, b:4)

   I told students that you can use either the 2nd or 3rd edition of the textbook. One says “one billion additions per second”; the other says “ten billion additions per second”. Use “one billion additions per second” when you do this problem.
8. (3) 3.4.5 [3.4.5] (simple cost matrix for an assignment problem whose optimal solution does not include smallest element)

9. (8) 3.4.6 [3.4.6] (partition problem) There is only so much a brute force algorithm can do to make this efficient. Mainly, try to avoid checking duplicate subsets or subsets that cannot possibly be a solution.

10. (6) 3.4.9 [not in 2nd edition] (non-attacking queens) **Details and my commentary on next page.**



Eight-queens problem Consider the classic puzzle of placing eight queens on an 8×8 chessboard so that no two queens are in the same row or in the same column or on the same diagonal. How many ways are there so that

a. no two queens are on the same square?

b. no two queens are in the same row?

c. no two queens are in the same row or in the same column?

Also estimate how long it would take to find all the solutions to the problem by exhaustive search based on each of these approaches on a computer capable of checking 10 billion positions per second.

The real question in each case is “if we make only these restrictions, then check each possibility to see if it is a solution, how many possibilities will we be checking?” If we made no restrictions at all (not even the restriction that multiple queens cannot occupy the same square), then there would be 64^8 placements to check.