MA/CSSE 473 – Design and Analysis of Algorithms

Homework 5 70 points total Updated Summer, 2017

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 $\Theta(N^2)$
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 cost in terms of N taking into account both the C and T parts, 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: (some algorithm/correct analysis: 2, efficient algorithm/correct analysis: 10, flaw: 2)

In previous terms, almost all students said #1 was the most time-consuming problem. In summer, 2017, I moved #9 here, and moved a couple of problems from this assignment to HW6A. Now #9 may be the most time0consuming problem.

Previous questions and answers from Piazza:

HW5 Problem 1 - What is "general approach" referring to? For the question "Can you think of any flaws in this general approach to testing?", what is the general approach referring to? The obvious algorithm or the efficient one?
Instructor Answer: Neither. It's a flaw in the whole approach of repeatedly dropping the gadget until it breaks.

HW 5 #1 Clarification What is meant by getting the big-theta worst-case running times in terms of N, C, and T? The number of floors the gadget is carried up and the number of tests required, the only two proxies for running time I can think of here, are solely dependent on N. Are you looking for the cost as the sum of C times the big-theta approximation of the number of floors the gadget is carried up and T times an approximation of the number of tests required?
Instructor Answer: If we were just doing the brute force test, N would be all that matters. But the other numbers might matter for a more efficient test approach. So go ahead and include the effects of C and T on the runtime for the brute force algorithm, in order to have good comparisons of the two approaches.

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.

   Clarification: For any n and m, you are to give an example of strings of length n and m that lead to the worst case. I.e. your answer should be general, not an answer in which you choose a specific m and n. So you can't begin with something like "Let m=7 and n=15". I think the need for this generality in your answer is implied by the phrase "a formula involving m and n"; you could only do this if your descriptions of the text and pattern also involve ma and n.

4. (10) 3.3.4 [3.3.3] (Manhattan distance). **Clarifications:** (b) You can sketch or simply list them. Assume that all points have integer coordinates.
   (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 for both?
   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 said that you can use either the 2nd or 3rd edition of the Levitin 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. (20) (Miller-Rabin test) For this problem you will need the excerpt from the Dasgupta book that is posted on Moodle, and/or Weiss section 9.6.
   Let N = 1729 (happens to be a Carmichael number, but you should not assume that as you discover the answers) for all parts of this problem.
   (a) How many values of **a** in the range 1..1728 pass the Fermat test [i.e. $a^{1728} \equiv 1 \pmod{1729}$]?
   (b) For how many of these "Fermat liar" values of **a** from part (a) does the Miller-Rabin test provide a witness that N is actually composite?
   (c) If we pick **a** at random from among 1, 2, ..., 1728, what is the probability that running the Miller-Rabin test on **a** will show that N is composite? I.e., Rabin showed that for any N, the probability is at least 75% for every N; what is that probability for the N=1729 case?

You'll need to write some code to help you solve this problem. Include the code in your submission.

There is not a lot of code to write, and it is straightforward. The issue is that you have to understand what is being asked and how to get the answers. For some of you, you won't recognize that you don't yet understand until you start to write the code.

Make sure that you understand what the *t* and *u* are; your code will need to calculate them.

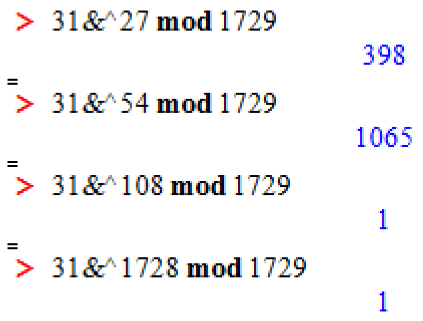
For part (b), I want to demonstrate an example of a number **a** that fits the description of the numbers that you are supposed to count. It is **a**=31.

I discovered how to do fast modular exponentiation in Maple, so I illustrate that here as well. You can also use the **modexp** Python code that I gave you, or use Python's **pow** function:

```
>>> pow(31,54, 1729)
1065
```

Java has `BigInteger.modPow`

Demonstration that 31 should be counted on part (b), using Maple: For n=1729, u=27 and t=3



```
> 31&^27 mod 1729
                              398
=
> 31&^54 mod 1729
                              1065
=
> 31&^108 mod 1729
                               1
=
> 31&^1728 mod 1729
                               1
```

Notice from the last line that 31 is a Fermat liar for 1729, since 1729 is composite but the 1728th power of 31 is congruent to 1 mod 1729.

In the Miller-Rabin test, we start with the 27th power, and keep squaring until we either get to the 1728th power or we find a non-trivial square root of 1 (mod 1729). In this case the latter happens; we see that 1065 is a non-trivial square root of 1, and so 1729 is composite.