COMP 251 2015, Assignment 1
Due Thursday October 1st 2015

[20%]

1. Read Chapter 1. Solve Exercise 4.

4. Gale and Shapley published their paper on the stable marriage problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

Basically, the situation was the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the m hospitals.

The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

We say that an assignment of students to hospitals is *stable* if neither of the following situations arises.

- First type of instability: There are students s and s', and a hospital h, so that
  - s is assigned to h, and
  - s' is assigned to no hospital, and
  - h prefers s' to s.
- Second type of instability: There are students s and s', and hospitals h and h', so that
  - s is assigned to h, and
  - s' is assigned to h', and
  - h prefers s' to s, and
  - s' prefers h to h'.

So we basically have the stable marriage problem from the section, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals, and give an efficient algorithm to find one.

[10%]

2. **Read Chapter 2.** Prove that if

$$\lim_{n\to\infty} f(n)/g(n) = 0$$

then f(n) **is** O(g(n)) but f(n) **is not** θ(g(n)).

3. **Solve the following Exercises**

[15%]

1. Suppose you have algorithms with the five running times listed below. (Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

a) $99n$
b) n^2
c) n^4
d) $n2^n$
e) 3^n

[15%]

3. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$f_1(n) = 99n^2$
$f_2(n) = 2^{\lg n}$
$f_3(n) = n^2 \log \log n$
$f_4(n) = n2^n$
$f_5(n) = 3^n$

[15%]

5. Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$.

(b) $2^{f(n)}$ is $O(2^{g(n)})$.

(c) $f(n)^2$ is $O(g(n)^2)$.

[25%]

8. You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar—at the moment it breaks, you have the correct answer—but you may have to drop it n times (rather than $\log n$ as in the binary search solution).

So here is the trade-off: it seems you can perform fewer drops if you're willing to break more jars. To understand better how this trade-off works at a quantitative level, let's consider how to run this experiment given a fixed "budget" of $k \geq 1$ jars. In other words, you have to determine the correct answer—the highest safe rung—and can use at most k jars in doing so.

(a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n \to \infty} f(n)/n = 0$.)

(b) Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each grows asymptotically slower than the previous one: $\lim_{n \to \infty} f_k(n)/f_{k-1}(n) = 0$ for each k.