CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #1, due Monday May 16th, 2016 at 2:15pm in Room x235, Box 32

One mark will be deducted if your solution uses multiple sheets of paper that are not stapled.

[6] 1. Describe an efficient algorithm for the following problem. The input is an instance for the Stable Matching problem (n men, n women, and their preference lists). The output should be “yes” if that instance has *only one* stable matching. Otherwise the output should be “no”.

(You can use any theorems mentioned in the lecture or written in the textbooks.)

[12] 2. For all n, design an instance of the Stable Matching problem such that the Proposal Algorithm will execute for $\Omega(n^2)$ iterations when provided that instance as input. You need to describe the instance, describe the sequence of proposals that the algorithm should make, and prove that $\Omega(n^2)$ iterations are required.

[12] 3. You are doing 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. While this algorithm will require the fewest tests, it may also result in many broken jars.

[3] (a) How many jars might you end up breaking, in the worst case?

If your primary goal were to conserve jars, on the other hand, you could try a different strategy. Start by dropping a jar from the first rung, then the second rung, etc. In this way, you break at most one jar. Unfortunately, you may also need n attempts.

So there seems to be a trade-off: the more jars you are willing to break, the fewer tries you will need.

[6] (b) Your boss is really cheap, but he is also too impatient to let you make n attempts. So he gives you 2 jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, where $f(n)$ is a function that is $o(n)$. (So, for example, $f(n) = n/2$ is *not* acceptable.)

[3] (c) Analyze the worst-case number of attempts of your algorithm from part (b).

More on next page

[12] 4. Consider the following basic problem: you are given an array A of size n, and you want to generate a two-dimensional $n \times n$ array B such that

$$B[i,j] = \begin{cases} \sum_{k=i}^{j} A[k] & \text{when } i \leq j \\ 0 & \text{otherwise} \end{cases} .$$

That is, $B[i,j]$ contains the sum of the elements from $A[i]$ to $A[j]$ (unless $j < i$). Here is a simple algorithm that achieves this:

```
Algorithm ComputeMatrix
  for i $\leftarrow$ 1 to n do
    for j $\leftarrow$ 1 to n do
      if i $\leq$ j then
        B[i,j] $\leftarrow$ the sum of the elements A[i], A[i+1], ..., A[j]
      else
        B[i,j] $\leftarrow$ 0
```

[2] (a) Using O notation, give as close an upper bound as you can for the running time of algorithm `ComputeMatrix`, as a function of n.

[8] (b) Although algorithm `ComputeMatrix` is the most natural one to solve the problem, it is not the most efficient. Give a different algorithm to solve this problem whose running time is a factor of n faster than that of algorithm `ComputeMatrix`.

[2] (c) Using Θ notation, write down the running time of your algorithm from part (b). You **must** justify your answer.

[1] 5. (Bonus) How long did it take you to complete this assignment (not including any time you spent revising your notes before starting)?