CPSC 320: Intermediate Algorithm Design and Analysis

Assignment #1, due Wednesday May 20^{th}, 2015 at 2:15pm in Room x235, Box 42

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

[6] 1. The Stable Matching problem, as discussed in class, assumes that every woman and every man has a fully ordered list of preference. In this and the next problem, we consider the situation where we have n women and n men (as before), but where a woman or man may have ties in her/his ranking. For instance, woman w_1 might like man m_3 best, followed by m_1 and m_4 in no particular order (that is, she does not prefer m_1 to m_4, or m_4 to m_1), followed by m_2. In this case, we will say that w_1 is *indifferent* between m_1 and m_4. It is of course possible for a woman or a man to be indifferent between more than two people.

A *strong instability* in a perfect matching consists of a woman w and a man m such that w and m both prefer each other to their current partner. Prove that there always exists a perfect matching with no strong instability by giving an algorithm that finds such a matching. Prove the correctness of your algorithm in a couple of sentences.

[6] 2. Continuing with the same setup as in the previous question, let us define a *weak instability* as a woman w with partner m and a man m' with a partner w', where either

- m prefers w' to w and w' either prefers m to m' or is indifferent between these two choices, or
- w' prefers m to m' and m either prefers w' to w or is indifferent between these two choices.

Prove that there does not necessarily exist a perfect matching without weak instabilities. Give a (small) example where every perfect matching has a weak instability.

[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.

(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.

(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.)

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

[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 ← 1 to n do
    for j ← 1 to n do
      if i ≤ j then
        B[i,j] ← the sum of the elements A[i], A[i+1], ..., A[j]
      else
        B[i,j] ← 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`.

[3] (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)?