Problem Set 1: Algorithm Design Basics

Handed out Friday, September 1. Due at the start of class Wednesday, September 13.
Upload your code for Problem 6 to Moodle by 2pm (9/13/2017).

Homework Information: Some of the problems are probably too long to attempt the night before the due date, so plan accordingly. No late homework will be accepted. Feel free to work with others, but the work you hand in must be your own.

Notation: Throughout the semester, we will use $lg\ x$ to denote logarithm of x base 2 ($log_2\ x$) and $ln\ x$ to denote the natural logarithm of x. We will use $log\ x$ when the base does not matter.

Problem 1.(15 points) Consider the following simpler (extremely gender-biased) algorithm for the stable marriage problem. As in the standard problem, there are n men, and n women, and each man and each woman has an n-element preference list that rank orders all the members of the opposite sex. This algorithm *ignores* the preferences of the women and simply pairs each man with the first available woman on his list.

```
for (i = 1 to n) {
  let (w[1],...,w[n]) be the women of m[i]'s preference list (from high to low)
  j = 1
  while (j <= n and m[i] is not yet engaged) {
      if (w[j] is not yet engaged){
          match m[i] with w[j] (and both are now engaged)
      }
      else j = j+1
  }
}
```

(Note that in this algorithm, once a woman accept's a man's proposal, she will never break it off.) We will explore the correctness of this algorithm by answering the following questions.

(a) Is this algorithm guaranteed to produce a perfect matching (that is, is every man paired with exactly one woman and vice versa)? If so, give a proof, and if not, give a counterexample and explain your counterexample.

(b) If your answer to (a) was "no", skip the rest of this problem. Otherwise, is the matching produced by this algorithm guaranteed to be stable? If so, give a proof, and if not, present a counterexample and explain your counterexample.

(c) If your answer to (b) was "yes" skip this part. Otherwise, suppose that all the women in this system have exactly the same sets of preferences, and in particular, they rank the men in (decreasing preference) order $< m_1, m_2, ..., m_n >$. (Each man's list contains all the women, but otherwise each man's preferences are arbitrary.) Under this restriction, is the matching produced by this algorithm guaranteed to be stable? As before, either give a proof or present a counterexample.

(Note: Throughout the semester, whenever you are asked to present a counterexample, you should strive to make your counterexample as short and clear as possible. In addition to giving the input for the counterexample, briefly explain what the algorithm does when run on this input and why it is wrong.)

Problem 2 (20 points). Consider the following summation, which holds for all $n \geq 0$,

$$\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i \right)^2$$

That is $(1^3 + 2^3 + ... + n^3) = (1 + 2 + ... + n)^2$.

(a) Prove this identity holds for all $n \geq 0$, by induction on n. (Recall that by convention, for $n = 0$ we have an empty sum, whose value is defined to be the additive identity, that is, zero.)

(b) The following figure provides an informal "pictorial proof" of this identity. Explain why. Hint: $n = 5$ in this figure, but this figure could be expanded for any n value.

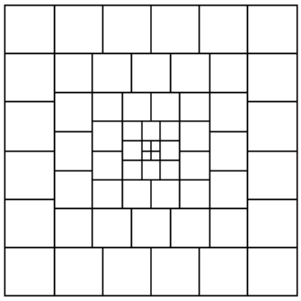

Figure 1: Problem 2.

Problem 3 (20 points). For each of the parts below, list the functions in increasing asymptotic order. In some cases functions may be asymptotically equivalent (that is f(n) is $\Theta(g(n))$). In such cases indicate this by writing $f(n) \approx g(n)$. When one is asymptotically strictly less than the other (that is, $f(n)$ is $O(g(n))$ but $f(n)$ is not $\Theta(g(n))$), express this as $f(n) < g(n)$. For example, given the set:

$$n^2 \qquad n \; log \; n \qquad 3n \; + \; n \; log \; n$$

the first function is $\Theta(n^2)$ and the other two are $\Theta(n \; log \; n)$, and therefore the answer would be

$$n \; log \; n \approx 3n \; + \; n \; log \; n < n^2.$$

Explanations are *not* required, but may be given to help in assigning partial credit.

(a) $(3/2)^n$ $3^{(n/2)}$ $2^{(n/3)}$

(b) $lg \; n$ $ln \; n$ $lg \; (n^2)$

(c) $n^{lg \; 4}$ $2^{lg \; n}$ $2^{(2 \; lg \; n)}$

(d) $max(50n^2, n^3)$ $50n^2 \; + \; n^3$ $min(50n^2, n^3)$

(e) $\lceil n^2/20 \rceil$ $\lfloor n^2/20 \rfloor$ $n^2/20$

Problem 4.(14 points) The purpose of this problem is to design a more efficient algorithm for the previous larger element problem, as introduced in class. Recall that we are given a sequence of numeric values, $< a_1, a_2, ..., a_n >$. For each element a_i, for $1 \leq i \leq n$, we want to know the index of the rightmost element of the sequence $< a_1, a_2, ..., a_i - 1 >$ whose value is strictly larger than a_i. If no element of this subsequence is larger than a_i then, by convention, the index will be 0. Here is naive the $\Theta(n^2)$ algorithm from class.

```
previousLarger(a[1..n]) {
    for (i = 1 to n) {
        j = i - 1;
        while (j > 0 and a[j] <= a[i]) j--;
        p[i] = j;
    }
    return p;
}
```

There is one obvious source of inefficiency in this algorithm, namely the statement `j--`, which steps through the array one element at a time. A more efficient approach would be to exploit p-values that have already been constructed. (If you don't see this right away, try drawing a picture.) Using this insight, design a more efficient algorithm. For full credit, your algorithm should run in $\Theta(n)$ time. Prove that your algorithm is correct and derive its running time.

Problem 5.(16 points) Consider the following recurrences defined by $n \geq 1$. In each case, apply the Master Theorem to derive an asymptotic formula for the recurrence. Try to present your answer in the simplest form you can. Show how you derived your answer in each case.

(a) $T(n) = 3T(\frac{n}{2}) + 4n$

(b) $T(n) = 2T(\frac{n}{2}) + n^5$

(c) $T(n) = 8T(\frac{n}{2}) + 3n^2 + n^3$

(d) $T(n) = 2T(\frac{n}{4}) + n\sqrt{n}$

Problem 6. Programming Problem (15 points) Implement insertion sort and mergesort sorting algorithms in Python. Your program should allow the user to specify which sorting algorithm they would like to use as well as the name of the input file they would like to use. (You may allow for this to be entered from command line or as prompts in your code. Please either supply comments in your code or helpful prompts so that I can easily run your code as you intended. Please also specify if you are using Python 2 or Python 3 in the comments.)

As part of your written homework, please indicate for each sorting algorithm how long your program takes to sort this input file:
http://cs.rhodes.edu/welshc/COMP355_F17/PS1_input.txt. (This file contains 300,000 nine-digit integer values.)
See code example "PythonReviewInputTimingParseFile.py" in my Public directory for help with calculating the run time of your program, parsing your input file, and getting arguments from the command line.

You should assume that all input files are of the same format: 1 integer per line. Your output may be the same format as the input file, or you may output the actual sorted array.

Example Input	Example Output
23	2
10	3
9	4
43	9
126	10
97	23
81	43
2	43
4	68
68	81
43	97
3	126