Problem Set 2: Algorithm Design Basics

Handed out Friday, August 30. Due at the start of class Friday, September 6.

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.

A Note about Writing Algorithms: When presenting algorithms, more detail is not necessarily better. Remember that your algorithm will be read by a human, not a compiler. You may assume that the reader is familiar with the problem and the material discussed in class. Be sure that your description or pseudo-code is sufficiently detailed that your intentions are clear and unambiguous. Avoid adding extraneous details and confusing jargon. (E.g., It is much clearer to say "Insert x at the end of the list" rather than list.insertAtEnd(x).) You may make use of any standard data structures (linked lists, binary trees, heaps, etc.) without explaining how to implement them. If you have any questions, check with me. In addition to presenting pseudo-code, explain your general strategy in English. (This way, if you make a coding error, I can ascertain your real intent and give partial credit.) It is also a good idea to provide a short example to illustrate your approach. Even if you are not explicitly asked, you should always provide a justification of correctness of your algorithm and analysis of its running time.

Problem 1.(20 points) Arrange the following functions in increasing order of asymptotic growth rate. If two or more functions have the same asymptotic growth rates, then indicate this. No explanation is required, but they are encouraged, since I can give partial credit if your explanation shows some understanding of the concepts.

$$
\begin{aligned}
f_0(n) &= n \lg n + n^{3/2} \\
f_1(n) &= n! \\
f_2(n) &= 2^n \\
f_3(n) &= n^5 \lg n + 3^n \\
f_4(n) &= \lg \lg n \\
f_5(n) &= \log_{10} n \\
f_6(n) &= 3^{\lg n} \\
f_7(n) &= \sqrt{\log_3 n} \\
f_8(n) &= n\sqrt{n} \\
f_9(n) &= n \lg n
\end{aligned}
$$

Problem 2.(15 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 the naive $\Theta(n^2)$ 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 3.(15 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) $T(n) = 5T(\frac{n}{2}) + 4n$

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

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

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

(e) $T(n) = 10T(\frac{n}{3}) + 6n^2$

Problem 4.(10 points) Programming Problem: (upload your code to Moodle)

The *counting sort* algorithm is an algorithm for sorting an array of integers, each of which happens to fall in the range $[0, U)$ for some number U.

Implement the counting sort algorithm in Python (pseudocode below). Your code should assume that you are given a text file with the input (prompt the user for the name of the input file).
The input file will have the following :

- The first line of the file will contain the value of U, followed by an empty line.
- The remaining lines will contain 1 integer per line. (You may assume each integer is between the values $[0, U)$).
- (See hw2input.txt on the course website for an example input file).

Your output should be a sorted list containing all the numbers in the file, except for U.

Here is pseudocode for counting sort:

```
countingSort(array A, int U) {
    let counts = new array of size U
    for i = 0 to U - 1:
        counts[i] = 0
    for i = 0 to length(A) - 1:
        counts[A[i]] = counts[A[i]] + 1
    let index = 0
    for i = 0 to U - 1:
        for j = 0 to counts[i] - 1:
            A[index] = i
            index = index + 1
}
```