15-451 Algorithms, Fall 2000

Homework # 1 **due: September 7, 2000**

Groundrules: You should try to do the assignment as much as possible by yourself. If you get stuck you may ask for help from friends or TAs or the instructors. In any event, you should do your writeup by yourself, in your own words. If you did get help from others, please list their names at the top of the first page of your assignment. Your assignment is due at the beginning of class on the due date.

Problems:

1. For each pair $\langle a, b\rangle$ of functions below, indicate whether $a = o(b)$, $a = \Theta(b)$, or $b = o(a)$:

$$\langle n \ln n, n \log_{10} n\rangle, \quad \langle n \log_2(n^c), n(\log_2 n)^c\rangle, \quad \langle \log_2 n, \sqrt{n}\rangle,$$

$$\langle (\log_2 n)^{\log_2 n}, n^{\log_2 \log_2 n}\rangle, \quad \langle 2^n, 2^{2n}\rangle, \quad \langle 2^n, n!\rangle.$$

   Give brief reasoning for each answer.

2. Solve the following recurrences, giving your answer in Θ notation. For each of them, assume the base case $T(x) = 1$ for $x \leq 2$. Show your work.

   (a) $T(n) = 2T(n/5) + n$.

   (b) $T(n) = 2T(n-1) + 1$.

   (c) $T(n) = 2T(n-2) + 1$.

   (d) $T(n) = T(n-3) + n^2$.

   (e) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$. (E.g., we might get this from a divide-and-conquer procedure that uses linear time to break the problem into $\sqrt{n}$ pieces of size $\sqrt{n}$ each. Hint: write out the recursion tree.)

   (f) $T(n) = 4T(\sqrt{n}) + (\lg n)^2$. Hint: define a new function $S(i) = T(2^i)$.

3. An improvement to multiplication method given in class involves splitting each n-bit number into *three* pieces of $n/3$ bits each (i.e., write X as $2^{2n/3}A + 2^{n/3}B + C$ and write Y as $2^{2n/3}D + 2^{n/3}E + F$). A straightforward product would now involve 9 multiplications of $n/3$-bit numbers, but by cleverly rearranging terms, it is possible to reduce this to 5 multiplications of $n/3$-bit numbers, plus a constant number of additions and shifts.

   Write down the recurrence that results, and solve it using Θ notation. (We are *not* asking you to come up with the algorithm for rearranging the terms.)

4. [Insertion Sort] Here is the C code for a sorting algorithm called insertion sort:

```
void insertionsort(int a[], int n) {
    int i,j,v;

    for (i=1; i<n; i++) {
        v = a[i];
        for (j=i-1; j>=0; j--) {
            if (v >= a[j]) break;
            a[j+1] = a[j];
        }
        a[j+1] = v;
    }
}
```

Roughly speaking, this algorithm works by sorting the first $i-1$ elements, then inserting the ith element where it belongs among the first $i-1$. In this problem we'll consider the number of comparisions between elements as our measure of the running time of the algorithm. (This is the number of times the code `v >= a[j]` is executed.)

(a) If the initial array consists of n distinct elements in decreasing order, how many comparisons are done? Justify your answer.

(b) If the initial array consists of n distinct elements in increasing (sorted) order, how many comparisons are done? Justify your answer.

It turns out that if the array is "mostly" sorted then insertion sort also does well.

Call a pair of indices $\langle i, j\rangle$ an *inversion* in an array $A = (a_0, \ldots, a_{n-1})$ if $i < j$ but $a_i > a_j$.

(c) Let C be the number of comparisons used to sort an array A. Let I be the number of inversions in A. Prove that:

$$I \le C \le n - 1 + I$$

5. Attach a photo of yourself to your handin, or alternatively, write down the URL of a photo of yourself. Students not solving problem 5 may be subjected to spontaneous use of the instructor's digital camera at some point in the course.