15-451 Algorithms, Fall 2004

Homework # 1 **due: September 14, 2004**

Please hand in each problem on a separate sheet and put your **name** and **recitation** (time or letter) at the top of each page. You will be handing each problem into a separate box, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done individually. Group work is only for the oral-presentation assignments.

Problems:

(25 pts) 1. **Recurrences.** 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) $T(n) = 4T(n/5) + n$.

(b) $T(n) = 4T(n - 5)$.

(c) $T(n) = T(n - 4) + n^4$.

(d) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$. (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.)

(15 pts) 2. **Recurrences for multiplication.** 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.)

(30 pts) 3. **Recurrences and proofs by induction.** Consider the following recurrence:

$$T(n) \quad = \quad 2T(n/2) + n \lg n.$$

(The base case isn't so important, but you can think of $T(2) = 2$ if you like.) We would like you to solve this recurrence using the "guess and prove by induction" method.

(a) Try a proof by induction using the guess "$T(n) \leq cn \lg n$." In other words, assume inductively that $T(n') \leq cn' \lg n'$ for all $n' < n$ and try to show it holds for n. This guess is *incorrect* and so your proof should *fail*. (If your proof succeeds, then there is a problem!!) Explain where this proof fails.

(b) Use the way the above proof failed to suggest a better guess $g(n)$. Explain how you arrived at this guess and prove by induction that $T(n) \leq g(n)$ as desired.

(c) Now give a proof by induction to show that $T(n) \geq c' g(n)$ where $c' > 0$ is some constant and $g(n)$ is your guess from (b). Combining this with (b), this implies that $T(n) = \Theta(g(n))$.

(30 pts) 4. **Probability and expectation**.

An *inversion* in an array $A = [a_1, a_2, \ldots, a_n]$ is a pair (a_i, a_j) such that $i < j$ but $a_i > a_j$. For example, in the array $[4, 2, 5, 3]$ there are three inversions. A sorted array has no inversions, and more generally, the number of inversions is a measure of how "well-sorted" an array is.

i. What is the *expected* number of inversions in a random array of n elements? By "random array" we mean a random permutation of n distinct elements $a_1, \ldots, a_n$. Show your work. Hint: use linearity of expectation.

ii. It turns out that the number of comparisons made by the Insertion-Sort sorting algorithm is between I and $n + I - 1$, where I is the number of inversions in the array. Given this fact, what does your answer to part (a) say about the average-case running time of Insertion Sort (in Θ notation)?