15-451 Algorithms, Fall 2007

Homework # 1 **Due: September 11, 2007**

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 $x \leq 5$. Show your work.

(a) $T(n) = 4T(n/5) + n$.

(b) $T(n) = T(n-3) + n^3$.

(c) $T(n) = 3T(n-5)$.

(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.)

(25 pts) 2. **Recurrences and proofs by induction.** Consider the following recurrence:

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

Let's use a base-case of $T(2) = 2$ and let's assume n is a power of 2. We would like you to solve this recurrence using the "guess and prove by induction" method.

(a) Try to prove by induction that $T(n) \leq cn \lg n$. 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!!) Point out where this proof fails.

(b) Use the way the above proof failed to suggest a better guess $g(n)$. Explain why you chose 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 $T(n) = \Theta(g(n))$.

(25 pts) 3. **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.

(a) 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.

(b) 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)?

(25 pts) 4. **Matrix games.** In the game of rock-paper-scissors, two players simultaneously choose rock, paper, or scissors, and the winner is determined according to the rule "rock beats scissors, scissors beats paper, and paper beats rock". Let's say the loser pays the winner \$1, and if both players choose the same object, then it is a draw.

For a game of this sort, a *strategy* is a deterministic or randomized method for picking what to play. The *value* of a strategy is the amount of money you win (or the expected amount if the strategy is randomized) against an opponent who plays optimally knowing your strategy (he has spies). For instance, the deterministic strategy "play rock" has value -1 since an opponent knowing this is your strategy would just play paper. The strategy "flip a coin and with probability 1/2 play rock and with probability 1/2 play paper" has value $-1/2$ since an opponent who knows this is your strategy would again play paper, giving you an expected gain of $\frac{1}{2} \cdot (-1) + \frac{1}{2} \cdot 0$. The strategy "pick each of rock, paper, or scissors with probability 1/3" has value 0, because your expected gain is 0 no matter what your opponent plays.

Here is a new game. Player A (Alice) hides either a nickel or a quarter behind her back. Then, player B (Bob) guesses which it is. If Bob guesses correctly, he wins the coin. If Bob guesses incorrectly, he has to pay Alice 15 cents. In other words, the amount that Alice wins can be summarized by the following "payoff matrix":

		Bob guesses	
		N	Q
Alice hides	N	-5	$+15$
	Q	$+15$	-25

This seems like a fair game since when Bob loses, he pays Alice the average of 5 and 25, but we will see that one of the players in fact has an advantage!

(a) What is the value to Alice of the strategy "with probability 1/2 hide a nickel and with probability 1/2 hide a quarter"? Remember, to solve this, figure out what Bob would do knowing that this is Alice's strategy.

(b) What strategy for Alice has the highest value, and what is its value?

(c) What strategy for Bob has the highest value to Bob, and what is its value? Remember, we are assuming now that *Alice* has spies and will play *her* best response to this strategy.

(d) Explain in a few sentences why your answer to part (*b*) proves that your answer in (*c*) was optimal (Bob can't hope to guarantee any better), and vice-versa.

(e) Is it better to be Alice or Bob in this game?

Comment: Matrix games often provide a helpful perspective on algorithm design. Think of a matrix where each row represents some algorithm, each column represents some possible input, and entry ij is the cost of running algorithm i on input j. (So, matrix entries are like payoffs to the adversarial column-player.) A good randomized algorithm, like randomized-quicksort, can be thought of as a randomized strategy for the row-player that has a low expected cost no matter what input the adversary selects.