# 15-451 Algorithms, Spring 2009

**Homework # 1** **Due: January 27, 2009**

---

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.

If you have any questions about the problems, please post them to the discussion bulletin board. Next to each problem is the name of the TA resposible for grading it. Any grading inquiries should be directed to the responsible TA.

---

**Problems:**

(25 pts) 1. **Recurrences. (Daniel Nuffer)** Solve the following recurrences, giving your answer in $\Theta$ notation. For each of them, assume the base case $T(x) = 1$ for $x \leq 5$. Show your work.

(a) $T(n) = 2T(n/3) + n$

(b) $T(n) = 3T(n-4)$

(c) $T(n) = T(n-3) + n^3$

(d) $T(n) = n^{2/3}T(n^{1/3}) + n$

(e) $T(n) = 3T(n/2) + n$

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

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

Let's use a base-case of $T(2) = 2$ and let's assume $n$ is a power of 2.

In parts $a$ and $b$, we will give a quick upper bound on $T(n)$. In parts $c$ and $d$, we will give a quick lower bound on $T(n)$. Finally, in parts $e$ and $f$, we will solve for $T(n)$ using the "guess and prove by induction" method.

(a) Show that $nlgn = O(n^K)$ for any $K > 1$.

(b) Use the master method to show that $T(n) = O(n^K)$ for any constant $K > 1$.

(c) Show that $nlgn = \Omega(n)$

(d) Use part c and the master method give a lower bound on $T(n)$

(e) 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.

(f) 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.

(25 pts) 3. **Probability and expectation. (Pranjal Awasthi)** The Insertion-Sort algorithm is as follows:

- **Input:** An array $A = [a_1, a_2, \ldots, a_n]$ of $n$ elements
- **Output:** The array $A$ in sorted order
- for$(i = 2 : n)$
  - Initialize $j = i$
  - while$(j > 1)$
    (a) if$(A[j-1] \leq A[j])$, break
    (b) else swap $A[j-1]$ and $A[j]$, set $j = j - 1$
  - end while
- end for
- Output $A$

In this problem we will analyze the average-case running time of insertion-sort. We will define running time as the number of comparisons performed (or the number of times the while loop is executed) by the algorithm.

(a) Let the input to the algorithm be $A = [4, 2, 5, 3]$. Show the state of the array $A$ after every comparison performed by the algorithm.

(b) 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.
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.

(c) Prove that after every comparison made by Insertion-Sort, the number of inversions in the array decreases by atmost 1.

(d) **Lower bound**: Based on part $c$, prove that the Insertion-Sort algorithm, when given as input an array $A$ of $n$ elements and having $I$ inversions, must perform atleast $I$ comparisons.

(e) Prove that after every comparison made by Insertion-sort, either the number of inversions in the array decreases by 1 or the loop variable $i$ (see the algorithm above) increases by 1.

(f) **Upper bound** Based on part $e$, prove that the insertion-sort algorithm, when given as input an array $A$ of $n$ elements and having $I$ inversions, will perform at most $n + I - 1$ comparisons.

(g) Based on your answers to part $b$, $d$ and $f$, what can you say about the expected (or average-case) running time of Insertion-sort (in $\Theta$ notation).

(25 pts) 4. **Matrix games (Maxim Makatchev).**

In the game of evens-odds, two players $E$ (Eve) and $O$ (Odelia) simultaneously show one or two fingers. Eve wins if the sum is even and Odelia wins if the sum is odd. For

example, they might play this game to decide who has to do some chore. Let's view winning as getting a score of $+1$ and losing as getting a score of $-1$.

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 score you get (or the expected score if the strategy is randomized) against an opponent who plays optimally knowing your strategy (she has spies). Notice that knowing a player's strategy does imply knowing whether this player is going to play one or two only when the strategy is deterministic (also called *pure strategy*). When the strategy is randomized (also called *mixed strategy*), knowing the strategy means knowing the probabilities of playing one and two, but not knowing exactly which random choice will actually be made.

For instance, considering the case of Odelia, the deterministic strategy "play one" has value $-1$ for Odelia since Eve knowing this is Odelia's strategy would just play one. ("Play one" would be Eve's optimal strategy, after she learned that Odelia's strategy is "play one".) Similarly, the deterministic strategy "play two" has value $-1$ for Odelia since Eve would just play two. ("Play two" would be Eve's optimal strategy, after she learned that Odelia's strategy is "play two".) The strategy "flip a coin and with probability $1/2$ play one and with probability $1/2$ play two" has value 0 for Odelia since whatever Eve chooses, the expected score will be 0. In other words, for this particular randomized Odelia's strategy, whatever Eve's strategy is, does not affect the expected score and hence there is no unique optimal strategy for Eve (i.e. any Eve's strategy can be called optimal).

Here is a variation on evens-odds: suppose that we change the game so that both players playing one is a draw (both get a score of 0). In other words, we can summarize the game from Odelia's point of view by the following "payoff matrix":

| | | Eve plays | |
|---|---|---|---|
| | | one | two |
| Odelia plays | one | 0 | 1 |
| | two | 1 | $-1$ |

(a) What is the value to Odelia of the strategy "with probability $1/2$ play one and with probability $1/2$ play two"? Remember, to solve this, figure out what Eve would do knowing that this is Odelia's strategy. As part of your answer provide a proof or explanation that Eve's strategy you found is optimal for the given Odelia's strategy.

(b) This time Odelia decides to play conservatively by always playing one, so that she never gets a negative score. What is the value to Odelia of the strategy "always play one"? What is Eve's optimal strategy in this case (provide a proof or explanation)?

(c) What is the strategy for Odelia that has the highest value, and what is its value? What is the corresponding optimal strategy for Eve? Is it unique?

(d) What strategy for Eve has the highest value to Eve, and what is its value to Eve? Remember, we are assuming now that *Odelia* has spies and will play *her* best response to Eve's strategy. Also, a win for Odelia is a loss for Eve and vice-versa.

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

**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.