15-451 Homework 1

Jan 20, 2008

Due: Tuesday Jan 29.

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. Direct questions to Bryant Lee (bryantl@cs.cmu.edu)

1 Question 1: Recurrences

1. 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) $T(n) = 3T(n/7) + n$

   (b) $T(n) = T(n - 10) + n^7$

   (c) $T(n) = 2T(n - 3) + 1/n$

   (d) $T(n) = n^{1/2}T(n^{1/4}) + n$

2. Consider the following recurrence:

   $$T(n) = 2T(n/2) + n(lgn)^2$$

   Use a base case of $T(2) = 2$ and 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 $T(n) \leq cn(lgn)^2$. In other words, assume inductively that $T(n') \leq cn'(lgn')^2$ for $n' < n$ and try to show it holds for n. This guess is incorrect and so your proof should fail. Point out where the proof fails.

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

2 Question 2: Master Theorem

1. For each of the following algorithms: Write $T(n)$, a recurrence describing the cost of the algorithm, and explain each part of the recurrence. Then draw a sample recursion tree for $n = 16$, and explain which part of the tree dominates the cost of the algorithm. Finally, explain which case of the master theorem the algorithm falls into, and what its total running time is.

   (a) Karatsuba multiplication of two n-bit integers.

   (b) Mergesort. (Take an array of n elements, mergesort the first half and the second half, then merge the two halves into one sorted array. The base case is one element, which must always be sorted.)

   (c) Consider the following algorithm: Given an $n \times n$ matrix, the algorithm squares the matrix using the naive $O(n^3)$ algorithm and then recursively calls itself on the upper left quadrant (an $n/2 \times n/2$ matrix). The algorithm stops after squaring the last 1×1 matrix. The output is an array of $logn$ matrices of size $n \times n, n/2 \times n/2, ...1 \times 1$.

   For instance, on input:

   $$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$$

   The output is:

   $$\begin{vmatrix} 7 & 10 \\ 15 & 22 \end{vmatrix}, \begin{vmatrix} 49 \end{vmatrix}$$

2. In class we saw (or will see) a deterministic, linear time algorithm to determine the median element of an array. Part of the algorithm involved splitting the array into sub-arrays of size five. Imagine that instead of splitting it into sub-arrays of size five, we split the array into sub-arrays of size seven. Give a recurrence for the running time of this algorithm (and an explanation of each term in the recurrence). Solve this to find the running time of the algorithm in Big-O notation.

3 Question 3: Probability

1. Given a fair 6-sided die:
   (a) What is the probability that X (the outcome) is 1?
   (b) What is the probability that X is even?
   (c) What is the probability that $X \geq 5$?
2. Given two independent, fair 6-sided dice:
   (a) What is the probability that $X_1 = 1$ and $X_2 = 1$?
   (b) What is the probability that $X_1 = 1$ or $X_2 = 1$?
   (c) What is the expected value of the sum $X_1 + X_2$?
   (d) Is the answer the same when they are not independent? If you do not know what it means for events to be independent you may find it helpful to refer to the wikipedia entry on "statistical independence". An example of non-independent (i.e. dependent) events is if the 2nd die is always a mirror image of the first. Each die itself would then be fair, but the 2nd would always give the same result as the 1st.
   (e) What is the expected number of times needed to throw one die to achieve a 6 at least once?
3. An inversion in an array $A = [A_1, A_2, ..., A_n]$ 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 integers? By "random array" we mean a random permutation of n distinct integers $a_1...a_n$. Show your work. Hint: use linearity of expectation.
   (b) Recall that Insertion Sort works by first sorting the first i elements, then inserting the $(i+1)$th element in its right place among the first i ones.

```
PROCEDURE insertionSort(array A)
    for i = 1 to length[A]-1 do
        value = A[i]
        j = i-1
        while (j >= 0) and (A[j] > value) do
            A[j + 1] = A[j]
            j = j-1
        A[j+1] = value
```

   For the Insertion Sort algorithm: If the input is an array of n distinct elements in decreasing order, how many comparisons are done?
   (c) For the Insertion Sort algorithm: If the input is an array of n distinct elements in increasing order, how many comparisons are done?
   (d) 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 big-Theta notation)?