CS 5633 Analysis of Algorithms – Spring 12

1/26/12

2. Homework

Due **2/2/12** before class

Always justify your answers. All algorithms should be as efficient as possible and all runtimes should be as tight as possible.

1. **Guessing and Induction**

   For each of the following recurrences use the recursion tree method to find a good guess of what it could solve to (make your guess as tight as possible). Then prove that $T(n)$ is in big-Oh of your guess by induction (inductive step and base case). *(Hint: Appendix A in the book has a list of solved summations that might be helpful. For simplicity you may want to use* $\log_3 n$ *instead of* $\log_2 n$ *in some cases.)*

   Every recursion below is stated for $n \geq 2$, and the base case is $T(1) = 1$.

   (a) $T(n) = 16T(\frac{n}{4}) + n^2$

   (b) $T(n) = 4T(\frac{n}{3}) + n^3$

2. **Master theorem**

   Use the master theorem to find tight asymptotic bounds for the following recurrences. Justify your answers (by showing which case of the master theorem applies, and why.) Assume that $T(1) = 1$. You should be able to solve this without using a calculator.

   (a) $T(n) = 81T(\frac{n}{3}) + n^4 \log^4 n$

   (b) $T(n) = T(\frac{3n}{4}) + \sqrt{n}$

   (c) $T(n) = 9T(\frac{n}{4}) + n^3 \log n$

   (d) $T(n) = 9T(\frac{n}{3}) + \log\log n$

3. **Tiling a room**

   Suppose you want to tile the floor of one of the rooms in your house. Your room has a quadratic shape of $n \times n$ square feet, where n is a power of two. Think of your room as being made up of n^2 squares. Your room is part of an old house in which one of the squares is taken up by an old heat vent, so, effectively there are only $n^2 - 1$ squares to cover with tiles in your room. You are going to use tiles that have a strange shape: They are each three square foot large and consist of three equal-sized squares glued together in a rectangular corner-shape.

   Give a divide-and-conquer algorithm to tile this room with these corner-shape tiles, for any $n \geq 2$ that is a power of two. Give an abstract problem definition, describe your algorithm in words and in pseudocode, argue why your algorithm is correct (no formal proof required), and analyze the runtime by setting up a runtime recurrence and solving it using the Master Theorem.

Flip over to back page $\Longrightarrow$

4. **Strassen**

(a) Consider multiplying a $mn \times n$ matrix with a $n \times mn$ matrix. Give an efficient algorithm that solves this task using Strassen's algorithm as a subroutine. Make your algorithm as efficient as possible. What is the runtime of your algorithm in terms of m and n?

(b) Consider multiplying a $n \times mn$ matrix by a $mn \times n$ matrix. Give an efficient algorithm that solves this task using Strassen's algorithm as a subroutine. Make your algorithm as efficient as possible. What is the runtime of your algorithm in terms of m and n?

(c) Suppose you want to develop an algorithm to multiply two $n \times n$ matrices in time faster than Strassen's algorithm. Suppose your algorithm proceeds in dividing the problem up into parts of size $\frac{n}{4} \times \frac{n}{4}$, and that your divide and combine steps together take $\Theta(n^2)$ time. You would like to find out how many subproblems you need in order to be faster than Strassen's algorithm. If you have a subproblems the recurrence is $T(n) = aT(\frac{n}{4}) + \Theta(n^2)$. Find the largest (integer) value of a for which your algorithm would be asymptotically faster than Strassen's.