CS 5633 Analysis of Algorithms – Spring 08

1/31/08

3. Homework

Due **2/7/08** before class

1. **Master theorem (10 points)**
   Use the master theorem to find tight asymptotic bounds for the following recurrences. Justify your results. If the master theorem cannot be applied use the recursion tree method to estimate the answer. Assume that $T(n)$ is constant for $n \leq 2$.
   (a) **(2 points)**
   $T(n) = T(\frac{n}{2}) + n$
   (b) **(2 points)**
   $T(n) = 8T(\frac{n}{2}) + n^3 \log^3 n$
   (c) **(2 points)**
   $T(n) = 2T(\frac{n}{2}) + \sqrt[3]{n}$
   (d) **(2 points)**
   $T(n) = T(n-2) + n/2$
   (e) **(2 points)**
   $T(n) = 3T(\frac{n}{2}) + n^3$

2. **Multiplying polynomials (10 points)**
   A polynomial of degree n is a function

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

   where a_i are constants and $a_n \neq 0$. For simplicity you may assume that n is a power of 2.

   (a) **a) (1 points)** What is the runtime of the straight-forward algorithm for multiplying two polynomials of degree n?
   (b) **b) (5 points)** We can rewrite the polynomial $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ as $x^{n/2}(a_n x^{n/2} + a_{n-1} x^{n/2-1} + \cdots + a_{n/2+1} + a_{n/2}) + (a_{n/2-1} x^{n/2-1} + \cdots + a_1 x + a_0)$. Use this as a starting point to design a divide-and-conquer algorithm for multiplying two degree-n polynomials. The runtime of your algorithm should be the same as the runtime of part a). Your divide-and-conquer algorithm should recurse on inputs of size $n/2$ (i.e., on polynomials of degree $n/2$).
   (c) **c) (1 point)**
   Show how to multiply two degree-1 polynomials $ax + b$ and $cx + d$ using only three multiplications. *Hint: One of the multiplications is* $(a + b) \cdot (c + d)$.
   (d) **d) (3 points)**
   Design a divide-and-conquer algorithm for multiplying two polynomials of degree n in time $\Theta(n^{\log_2 3})$. *Hint: Reuse part* **b)** *and speed it up with the knowledge of part* **c)**

3. **Rolling dice (5 points)**
   - **a) (3 points)**
     Compute the expected value of rolling a six-sided die.
   - **b) (2 points)**
     Use linearity of expectation to compute the expected value of the sum of two 6-sided dice.

   Clearly describe the sample space and the random variables you use. 3 points will be given for correct notation.

FLIP OVER TO BACK PAGE ⟹

4. SIMPLEROULETTE (3 POINTS)

   The game SIMPLEROULETTE is played as follows: The roulette wheel has a slot for each number from 0 to 36. You can bet on any number between 1 and 36, but not on the number 0. A bet costs you \$10. If the ball drops on the slot with your number, you get paid \$360, otherwise you don't get paid anything.

   Assuming that the wheel is fair (i.e., all numbers are equally likely), what is your expected win/loss in this game?

   Clearly describe the sample space and the random variables you use. 2 points will be given for correct notation.

Related questions from previous PhD Exams

Just for your information. You **do not** need to solve them for homework credit.

P1 This problem is concerned with divide-and-conquer algorithms and recurrence relations. Note that in the following we will write $T(n/2)$ to denote $T(\lfloor n/2 \rfloor)$ or $T(\lceil n/2 \rceil)$, which simplifies arithmetic manipulation and does not change asymptotic bounds.

(a) Explain the divide-and-conquer paradigm for algorithm design, including a generic recurrence relation for the runtime $T(n)$ for inputs of size n. You may introduce additional constants or functions for your description.

(b) Consider the recurrence relation $T(n) = 2T(n/2) + n$, and assume $T(2) = 2$. Prove by induction that $T(n) = n \log_2 n$ when $n \geq 2$ is a power of 2.

(c) Provide and briefly justify asymptotically tight bounds for the following recurrence relations:

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

P2 This problem is concerned with binary heaps. Below are some key subroutines. The heap property is that $A[\text{PARENT}(i)] \geq A[i]$ whenever $1 \leq i$ and $\text{PARENT}(i) \leq \textit{heap-size}[A]$.

```
PARENT(i)
  return ⌊i/2⌋

LEFT(i)
  return 2i

RIGHT(i)
  return 2i + 1
```

(a) Using the subroutines above, write pseudocode for an algorithm that ensures that the heap property is true for an array A with elements from indices 1 to $\textit{length}[A]$.

(b) Justify the correctness of your algorithm.

(c) Provide asymptotically tight bounds on the running time of your algorithm.

(d) Justify your bounds on the running time of your algorithm.