CS 5633 Analysis of Algorithms – Spring 09

1/22/09

2. Homework

Due **1/29/09** before class

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

1. $\log n!$ **(2 points)**
   Show that $\log n! \in \Theta(n \log n)$ without using Stirling's formula. *(Hint: Show that* $n! \geq (n/2)^{n/2}$*)*

2. d**-Heaps (7 points)**
   A *d-ary max-heap*, d-heap for short, is the generalization of a binary heap to a d-ary tree. The tree still has to be almost complete, and for every child of a parent the child's value is less or equal than the parent's value.

   (a) (2 points) Suppose a d-heap is stored in an array (that begins with index 1). For an entry located at index i in which location is its parent and in which locations are its children? (You do not have to formally prove your answer, but please give at least an example)

   (b) (1 point) What is the height of a d-heap that contains n elements? Give your answer in Θ-notation, and shortly justify why it is correct (no formal proof needed). The height should be a function of n and d.

   (c) (2 points) Shortly explain how the insertion procedure works for d-heaps (you do not have to give pseudocode). What is the runtime of inserting an element into a d-heap of n elements? The runtime should be a function of n and d.

   (d) (2 points) Shortly explain how the extract_max procedure works for d-heaps (you do not have to give pseudocode). What is the runtime of extracting the maximum from a d-heap of n elements? The runtime should be a function of n and d.

3. **Majority Element (5 points)**
   Let $A[1..n]$ be an array of n numbers. A number in A is a *majority element* if A contains this number at least $\lfloor n/2 \rfloor + 1$ times.

   (a) (3 points) Write a divide-and-conquer algorithm that determines whether a given array $A[1..n]$ contains a majority element, and if so, returns it. Your algorithm should run in $O(n \log n)$ time. You are **not** allowed to sort the array.

   (b) (2 points) Set up a recurrence relation for the runtime of your algorithm. Argue why it solves to $O(n \log n)$.

FLIP OVER TO BACK PAGE $\Longrightarrow$

4. **Guessing and Induction (8 points)**

   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 $\log_4 n$ instead of $\log_2 n$.)*

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

   (a) **(4 points)**
   $T(n) = 4T(\frac{n}{4}) + 2n$

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

Related questions from previous PhD Exams

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

P1 (a) 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.

   i. 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.

   ii. 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.

   iii. Provide and briefly justify asymptotically tight bounds for the following recurrence relations:

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