CS 5633 Analysis of Algorithms – Spring 08

2/21/08

5. Homework

Due **2/28/08** before class

1. **Red-black trees (6 points)**
   Find a sequence of numbers which, when incrementally inserted into a red-black tree, causes the following sequence of rotations:

   left, right, right, left.

   You may start with an initially non-empty tree, and you may insert numbers that do not cause any rotations. But there should not be any additional rotations performed.

   Draw the sequence of trees that you obtain after each insertion. For each such tree indicate the node that violates the red-black tree condition, indicate the nodes that participate in the rotation, the type of the rotation, and the subtrees that correspond to each other before and after the rotation.
   Hint: Use a red-black tree demo from the web.

2. **Deterministic select (6 points)**
   Consider two variations of the deterministic selection algorithm:

   a) Divide into groups of 7.
   b) Divide into groups of 3.

   Show that the runtime proof of $O(n)$ works fine in case a). In case b) show where you run into problems when trying to prove a runtime of $O(n)$.

3. **Median computation (6 points)**
   Suppose arrays A and B are **both sorted** and contain n elements each. Give a randomized divide-and-conquer algorithm to find the median of $A \cup B$ in expected $O(\log n)$ time. (Describe it either in words or as pseudo-code; whatever you prefer). Argue **shortly** why the runtime is $O(\log n)$. *Hint: Take a look at randomized select.*

4. **Multi-min (6 points)**
   Consider the following task: Given an unsorted array of n numbers, find the k smallest numbers and output them in sorted order.
   Describe three inherently different algorithms that solve this problem. Analyze their runtimes in terms of n and k (so you should have n and k in the big-Oh notation). Try to find the fastest possible algorithm. Which of your algorithms is the fastest?

Related questions from previous PhD Exams

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

1. This problem is concerned with binary search trees and the successors of a node.

   (a) Define "binary search tree".

   (b) Given a node x in a binary search tree, write an *efficient* procedure in pseudocode to find the successor of x. Assume that nodes have fields *left*, *right*, *parent*, and *key*. What is the running time of your procedure in terms of n and h? (Justify your answer.)

   (c) Consider the following procedure to print all n elements of a binary search tree in order: First find the minimum element and then make $n-1$ calls to your successor procedure. What is the running time of this procedure? (Justify your answer.)

   (d) For a binary search tree, determine the running time of finding a value v and then performing m successive calls to your successor procedure. (Justify your answer.)

2. The k-way merging problem is concerned with merging k sorted lists, each containing n/k elements, into a single sorted list of n elements.

   (a) Consider the incremental algorithm which merges the first two lists into a sorted list of $2n/k$ elements, then merges the result with the third list, then with the fourth list, and so on. Analyze the worst-case running time of the incremental algorithm in terms of n and k.

   (b) The incremental algorithm is not efficient. Describe a more efficient algorithm for the k-way merging problem. Analyze its worst-case running time in terms of n and k. (For simplicity you may assume k is a power of 2.)

   (c) What is the space complexity of the two above algorithms?

   (d) In this part we will derive a lower bound for the k-way merging problem, assuming that the merging is done by performing comparisons.

      i. The elements of the k lists could end up in various positions in the final merged list. Argue that there are $n!/((n/k)!)^k$ possible different ways for all elements of the k lists to be placed in the final list.
      (Hint: First choose n/k out of the n positions for the first list, then choose n/k out of the remaining $n-k$ positions for the second list, and so on.)

      ii. Use part A to prove a lower bound of $\Omega(n \log k)$ on the worst-case number of comparisons needed for the k-way merging problem. Use the fact that $(n/e)^n \leq n! \leq n^n$.