CORE EXAM: ALGORITHMS PART

Department of Computer Science
New York University
May 12, 2000

• This is a *one and one-half hour* examination. It is the *Algorithms* part of the Core Exam for the M.S. program in Computer Science.
• Write your name on the front of the envelop, and *no where else*.
• Answer each question in a separate booklet, which is labeled accordingly.
• Please answer all three questions. Assume standard results, except where asked to prove them. Keep your answers brief but precise. Rewriting your solutions is recommended (but hand in the original as scratch).

QUESTION 1

PART (i) Given two sorted arrays A, B of lengths n, m respectively describe the MERGE algorithm for creating a single sorted array C with these values. Illustrate the algorithm by merging the lists $A[1] = 11, A[2] = 53, A[3] = 58$ and $B[1] = 20, B[2] = 36, B[3] = 49, B[4] = 63, B[5] = 70$ (so $m = 3, n = 5$).

PART (ii) Let $M(m, n)$ be the number of key comparisons the MERGE algorithm make *in the worst case* to merge two sorted arrays, of sizes m and n respectively. Give the exact solution for $M(m, n)$ with a brief justification for your answer.

PART (iii) There are many algorithms for sorting arrays. Give such an algorithm that makes critical use of the MERGE algorithm. Let $S(n)$ denote the number of key comparisons that your algorithm make in the worst case for n elements. Give a recursive formula for $S(n)$ in terms of $M(m, n)$, with brief explanation (do not forget the base case). Now, solve this recurrence (up to big-Oh order) using an induction proof.

QUESTION 2

PART (i) A binary search tree typically assumes that there is only one search key per node. Suppose we want to be able to perform searching based on two keys, KEY0 and KEY1. One way to do this uses a “2-d tree”. A 2-d tree is similar to a binary search tree, except that branching at even levels is done with respect to KEY0, and branching at odd levels is done with KEY1. Thus, the root which is at level 0, uses KEY0. Describe an algorithm to insert an item $(k_0, k_1, data)$ into a 2-d tree.

PART (ii) Now suppose that we want to maintain a 2-d tree of conferences with start dates and end dates of each conference. Thus, there are two keys: the start date, and the end date. Dates below are given as month/date/year. Assume the existence of a procedure that compares dates. Draw the tree that results from inserting (in the order indicated) the following conferences.

Artificial Intelligence Conference	6/1/2001-6/4/2001
Scientific Computing Conference	7/12/2001-7/15/2001
15th DataBase Conference	7/7/2001-7/9/2001
16th DataBase Conference	7/6/2002-7/8/2002
Theory of Computing Conference	7/8/2001-7/11/2001
Computational Geometry Symposium	6/15/2001-6/20/2001

QUESTION 3

Recall that the product $C = AB$ of two matrices A and B, with real entries, and of order ℓ by m and m by n, respectively, is given by the formula

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}, \quad (\text{for } i = 1, \ldots, \ell;\ j = 1, \ldots, n).$$

Here we will assume that at least one of the matrices is so large that it does not fit into the main memory but that it has to be stored on a slower device, which we assume is a tape drive. Random access into a tape is too slow, so our computational cost model is the number of times that your algorithm has to go over the data in a tape: at the start of the computation, you position the tape head at beginning of the data and read successive data on the tape sequentially (pausing as often as you like to do computation, etc). At any moment, you can "reset" (move the tape head back to the beginning of the data) to re-read the tape. We want to minimize the number of resets in our algorithm. Assume that the main memory can store a few rows or columns from the matrices (i.e., $O(\max(\ell, m, n))$ space is available). Also, the matrices A and B are stored in two separate "input" tapes, and the output C is written out into a third tape.

PART (i) Suppose the matrix A is stored in a one-dimensional array rA, column by column, on the first tape. This means that the matrix element a_{ij} is the $(i + (j-1)\ell)-$th component of rA. Further assume that the matrix B has just one column, i.e., that $n = 1$. Show how we can compute the product AB without any "resets" (so we pass over both input tapes only once).

PART (ii) Now assume A and B have many rows and columns and also has many zero elements so that the following *sparse matrix* data structure is used to store them. Suppose A has N_A nonzero elements. These elements are ordered column by column and stored as the first component of the record (RA_i, IA_i) for $i = 1, \ldots N_A$. The second component IA_i provides the row index of the matrix element in question. In addition, there is an array JA of integers with array length m that points to the first record containing the elements of each column of A. Thus, to find the elements of the third column of A we have to inspect the records (RA_i, IA_i) for $i = JA[3], \ldots, JA[4] - 1$. The matrix B is stored using exactly the same data structure.

Show how to compute an *arbitrary* row of $C = AB$ without any resets of the tapes. You may assume that the arrays JA and JB are kept in main memory.