COMP 271 Design and Analysis of Algorithms
2003 Spring Semester
Written Assignment # 1
Distributed: Feb 20, 2003
REVISED & CORRECTED Feb 27, 2003
Due: March 6, 2003 *at start of class*

Note: Please follow the guidelines on doing your own work and avoiding plagiarism given on the class home page.

Problem 1: (20 points) Prove the following properties of the $O()$-notation:

1. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n)))$.
2. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$
   then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$.
3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$
   then $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n))$.

Problem 2: (20 points) Consider the following recurrence relation:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/4) + n \log n, \quad \text{for } n > 1 \end{aligned}$$

where n is a power of 4. *Prove* that $T(n) = O(n \log^2 n)$.

Problem 3: (30 points) Suppose that X and Y are two sorted sequences, containing m and n elements respectively. Design an efficient algorithm to find the kth smallest element in the set of $m+n$ combined elements. Note that the best algorithm runs in time $O(\log(\max(m, n)))$. (Hint: Use binary search).

Problem 4: (30 points) An $n \times n$ matrix is *monotone* if each of its rows and columns are sorted in nondecreasing order. For example,

$$\begin{pmatrix} 1 & 12 & 17 & 19 \\ 15 & 18 & 20 & 25 \\ 16 & 23 & 26 & 30 \\ 24 & 27 & 31 & 35 \end{pmatrix}$$

is monotone. Given monotone matrix A and value a, the *membership problem* is to test whether $a \in A$ or $a \notin A$. More specifically $MEM(a, x_1, y_1, x_2, y_2)$ will return `TRUE` if there exists i, j such that $a == A[i, j]$ where $x_1 \le i \le x_2$ and $y_1 \le j \le y_2$ and `FALSE` otherwise. (Note that we are labelling the rows so that the top row is 1 and the bottom one N, while the left column is 1 and the right column is N.) For example, in the given matrix, $MEM(31, 1, 1, 4, 4) ==$ `TRUE` while $MEM(31, 1, 1, 3, 3) ==$ `FALSE`. Consider the following algorithm:

```
MEM(a, x1, y1, x2, y2) :
If ((x1 > x2) OR (y1 > y2))                             % Check Validity of indices
      Return(FALSE)

Else if ((x1 == x2) AND (y1 == y2))                     % Check if only one element
      {  If (a == A[x1, y1]) Return(TRUE)
         Else Return(FALSE)
      }

Else if ((x1 + 1 == x2) AND (y1 + 1 == y2))             % Check if 2X2 submatrix
      {  If ((a == A[x1, y1]) OR (a == A[x1, y2]) OR (a == A[x2, y1]) OR (a == A[x2, y2]))
              Return(TRUE)
         Else
              Return(FALSE)
      }

Else  {  u = floor((x1+x2)/2); v = floor((y1+y2)/2);    % Recurse on smaller matrices
      If (Mem(a, u, y1, x2, v) == TRUE)                 % upper-right submatrix
         Return(TRUE)
      If (Mem(a, x1, v, u, y2) == TRUE)}                % lower-left submatrix
         Return(TRUE)
      If (a <= A[u, v])                                 % Corrected 27/03/03
         If  (Mem(a, x1, y1, u, v) == TRUE)             % upper-left submatrix
              Return(TRUE)
      If (a > A[u, v])                                  % Corrected 27/03/03
         If  (Mem(a, u, v, x2, y2) == TRUE)             % lower-right submatrix
              Return(TRUE)
      Return(FALSE)
      }
```

1. Show that this algorithm (i) terminates and (ii) correctly solves the membership problem for monotone matrices.
2. Analyze the running time of the algorithm when run on an $N \times N$ monotone matrix, i.e., when $MEM(a, 1, 1, N, N)$ is called. You may assume that N is always of the form $N = 2^k + 1$, k an integer, if that makes the analysis easier. The running time should measure the worst case number of comparisons made and be given using $O()$ notation. Your running time should be as precise as possible, e..g, if the running time is actually $\Theta(n^2)$ then $O(n^2 \log n)$ would not be tight enough.

Extra Credit Design an $O(N)$ algorithm for solving the membership problem in monotone matrices. To fully solve this problem you must show why your algorithm is correct AND prove the $O(N)$ running time.