CSE 347 Analysis of Algorithms February 27, 2017

Sample Midterm Exam

Instructors: Brendan Juba & Robert Utterback *Due: End of class*

This is a sample midterm exam. It is intended to be representative of the content, length, and difficulty of the questions you will be asked on your midterm exam.

Instructions. Answer questions as clearly and concisely as you can. Try to fit your solutions in the space provided, continuing on the provided extra sheets of paper if necessary. We will award partial credit based on the work that you provide; if you get stuck, *be sure to explain what you are trying to do.*

This is a *closed book* exam. The use of laptops, phones, smartwatches, etc. is *not allowed.* You are allowed to reference a crib sheet that you prepared yourself, contained entirely on a single 8.5×11" (US letter size) sheet of paper. Any other written material is not permitted.

You may use (without proof) any theorem that has been stated in lecture. Except on the Reading Solutions problem, you may also reference the result of the homework problems you were assigned without proof. You *may not* use the result of other problems from the book without providing a proof.

Name:

Student ID:

Problem Number	Points Possible	Points Received
1	15	
2	15	
3	15	
4	15	
5	20	
TOTAL	80	

1. *(15 points)* Suppose that we wish to multiply a sequence of matrices, $A_1 A_2 \cdots, A_{m-1}$, in which the ith matrix has dimensions $n_i \times n_{i+1}$ for varying integers n_i. The total number of operations performed may depend significantly on the order in which we multiply the matrices: for example, if we have matrices that are $n \times 1$, $1 \times n$, and $n \times 1$, it is advantageous to first multiply the second two matrices (essentially an inner product, taking only n steps), followed by multiplying the first $n \times 1$ matrix by the 1×1 result, in another n steps. If we had performed the multiplication in the other order, we would first produce the $n \times n$ product (taking n^2 steps) followed by the product of an $n \times n$ matrix and a $n \times 1$ matrix, essentially a matrix-vector product, using another $O(n^2)$ steps. Consider the following algorithm:

input : Sequence of matrix dimensions $n_1, n_2, \ldots, n_m$
begin
  Initialize a $(m-1) \times (m-1)$ table M such that $M[i,i] = 0$ for all i and a $(m-2) \times (m-1)$ table C
  for $\ell = 2, \ldots, m-1$ **do**
    for $i = 1 \ldots, m - \ell$ **do**
      Put $j \leftarrow i + \ell - 1$, $M[i,j] \leftarrow \infty$
      for $k = i+1, \ldots, j-1$ **do**
        if $M[i,k] + M[k+1,j] + n_i n_{k+1} n_{j+1} < M[i,j]$ **then**
          Put $M[i,j] \leftarrow M[i,k] + M[k+1,j] + n_i n_{k+1} n_{j+1}$,
          Put $C[i,j] \leftarrow k$.
        end
      end
    end
  end
end
return M, C

(a) *(8 points)* Prove that in the table M, $M[i,j]$ contains the optimal cost of multplying the matrices in the range $i, i+1, \ldots, j$, assuming we use the naïve $n_i n_{i+1} n_{i+2}$-time algorithm for multiplying matrices of dimension $n_i \times n_{i+1}$ and $n_{i+1} \times n_{i+2}$.

(b) *(7 points)* Give an algorithm that uses the returned M and C to produce a binary tree in which the leaves are labeled by the index of one of the matrices A_i, such that a recursive algorithm that multiplies (using the naïve algorithm) the matrix obtained from recursing on the left subtree with the matrix obtained from recursing on the right subtree will return the matrix product $A_1 A_2 \cdots A_{m-1}$ using this optimal number of operations. (You must prove that your algorithm is correct and runs in polynomial time.)

2. *(15 points)* We consider the task of scheduling tasks of unit length, each of which has a *deadline* and a *penalty* for missing the deadline. Naturally, we would like to perform the tasks in some schedule that minimizes the total penalty we suffer.

   Consider the following algorithm:

```
input : List T of n tasks with deadlines d_1, ..., d_n and penalties p_1, ..., p_n
begin
    Initialize an array S of size n in which every entry is NULL.
    Initialize k = n.
    Sort the tasks in order of decreasing penalty
    for i = 1, ..., n do
        Put j ← d_i
        while j > 0 and S[j] ≠ NULL do Decrement j
        if j > 0 and S[j] = NULL then Put ith task in S[j]
        else
            while S[k] ≠ NULL do Decrement k
            Put the ith task in S[k]
            Decrement k
        end
    end
    return S
end
```

   Prove that this algorithm returns a schedule with the minimum possible total penalty (sum of the penalties for tasks whose deadlines were missed in the schedule).

This page intentionally left blank.

3. *(15 points)* Consider the following variant of the maximum network flow problem: in addition to the usual capacity constraints on edges, we also have a capacity constraint on *vertices*. That is, for each vertex $v \in V$, there is a number $c(v) \geq 0$ such that in any valid flow f, $\sum_{e \text{ in to } v} f(e) \leq c(v)$. Give an algorithm that uses an algorithm for the usual maximum flow problem to solve this problem, by invoking it on a new graph of size polynomially related to the size of the original input. (Be sure that you bound the running time and argue the correctness of your algorithm!)

4. *(15 points)* **Reading solutions.** The following is a problem that appeared on your homework. *For this problem only,* you may not simply reference the fact that it appeared on your homework. You must present a solution to this problem.

   You are given n nonvertical lines in the plane, labeled $L_1, \ldots, L_n$, with the ith line specified by the equation $a_i x + b_i$. We assume that no three of the lines meet at a single point. We say that a line L_i is *uppermost* at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x-coordinate at which it is uppermost—intuitively, some portion of it can be seen if you look down from "$y = \infty$."

   Give an algorithm that takes n lines as input and in $O(n \log n)$ time returns all of the lines that are visible.

5. *(20 points)* Suppose you are given an array of integers. Give an algorithm that returns an array in which only the *first* occurrence of an element remains, and these first occurrences appear in the same order as the original list. Your algorithm must run in time $O(n \log n)$. Prove its correctness and running time bound.