CSCI-GA.1170-001/002 Fundamental Algorithms September 17, 2014

Solutions to Problem 1 of Homework 3 (22 points)

*Name: **** INSERT YOUR NAME HERE **** Due: Wednesday, September 24*

The sequence $\{F_n \mid n \geq 0\}$ are defined as follows: $F_0 = 1$, $F_1 = 1$, $F_2 = 2$ and, for $i > 1$, define $F_i := F_{i-1} + F_{i-2} + 2F_{i-3}$.

(a) (6 Points) Notice the following matrix equation:

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F_i \\ F_{i-1} \\ F_{i-2} \end{pmatrix} = \begin{pmatrix} F_{i+1} \\ F_i \\ F_{i-1} \end{pmatrix}$$

Use this equation and the ideas of fast $O(\log n)$ time exponentiation to build an efficient algorithm for computing F_n. Analyze it's runtime in terms of the number of 3×3 matrix multiplications performed (each of which takes a constant number of integer additions/multiplications).

Solution: ***************** INSERT PROBLEM 1a SOLUTION HERE *************** □

(b) (6 Points) Prove by induction that, for some constant $a > 1$, $F_n = \Theta(a^n)$. Namely, prove by induction that for some constant c_1 you have $F_n \leq c_1 \cdot a^n$, and for some constant c_2 you have $F_n \geq c_2 \cdot a^n$. Thus, F_n takes $O(n)$ bits to represent. What is the right constant a and the best c_1 and c_2 you can find. (**Hint**: Pay attention to the base case $n = 0, 1, 2$. Also, you need to do *two* very similar inductive proofs.)

Solution: ***************** INSERT PROBLEM 1b SOLUTION HERE *************** □

(c) (6 points) In your algorithm of part (a) you only counted the number of 3×3 matrix multiplications. However, the integers used to compute (F_i, F_{i-1}, F_{i-2}) are $O(i)$ (in fact, $\Theta(i)$) bits long, by part (b). Thus, the 3×3 matrix multiplication used at that level of recursion will not take $O(1)$ time. In fact, using Karatsuba's multiplication, let us assume that the last matrix multiplication you use to compute (F_i, F_{i-1}, F_{i-2}) takes time $O(i^{\log_2 3})$. Given this more realistic estimate, analyze the actual running time $T(n)$ of your algorithm in part (a).

Solution: ***************** INSERT PROBLEM 1c SOLUTION HERE *************** □

(d) (4 points) Finally, let us look at the naive sequential algorithm which computes $F_3, F_4, \ldots, F_n$ one-by-one. Assuming each F_i takes $\Theta(i)$ bits to represent, and that integer addition/subtraction takes time $O(i)$ (multiplication by two can be implemented by addition), analyze the actual running time of the naive algorithm. How does it compare to your answer in part (c)?

Solution: ***************** INSERT PROBLEM 1d SOLUTION HERE *************** □

CSCI-GA.1170-001/002 Fundamental Algorithms September 17, 2014

Solutions to Problem 2 of Homework 3 (10 (+6) points)

*Name: **** INSERT YOUR NAME HERE **** Due: Wednesday, September 24*

The Tower of Hanoi is a well known mathematical puzzle. It consists of three rods, and a number n of disks of different sizes which can slide onto any rod. The puzzle starts with all disks stacked up on the 1st rod in order of increasing size with the smallest on top. The objective of the puzzle is to move all the disks to the 3rd rod, while obeying the following rules.

- Only one disk is moved at a time
- Each move consists of taking one disk from top of a rod, and moving it on top of the stack on another rod
- No disk may be placed on top of a smaller disk.

A recursive algorithm that solves this problem is as follows: We first move the top $n-1$ disks from rod 1 to rod 2. Then we move the largest disk from rod 1 to rod 3 and then move the $n-1$ smaller disks from rod 2 to rod 3. Using the symmetry between the rods, the number of steps that this algorithm takes is given by the recurrence

$$T(n) = 2T(n-1) + 1 \ ,$$

which can be solved to get $T(n) = 2^n - 1$.

(a) (5 points) Show that the above algorithm is optimal, i.e., there does not exist a strategy that solves the Tower of Hanoi puzzle in less than $2^n - 1$ steps.

Solution: **************** INSERT PROBLEM 2a SOLUTION HERE *************** □

(b) (5 points) Suppose the moves are restricted further such that you are only allowed to move disks to and from rod 2. Give an algorithm that solves the puzzle in $O(3^n)$ steps.

Solution: **************** INSERT PROBLEM 2b SOLUTION HERE *************** □

(c) (6 points(**Extra credit**)) Suppose the moves are restricted such that you are only allowed to move from rod 1 to rod 2, rod 2 to rod 3, and from rod 3 to rod 1. Give an algorithm that solves the puzzle in $O((1+\sqrt{3})^n)$ steps.

Solution: **************** INSERT PROBLEM 2c SOLUTION HERE *************** □

CSCI-GA.1170-001/002 Fundamental Algorithms September 17, 2014

Solutions to Problem 3 of Homework 3 (5 Points)

*Name: **** INSERT YOUR NAME HERE **** Due: Wednesday, September 24*

Let n be a multiple of m. Design an algorithm that can multiply an n bit integer with an m bit integers in time $O(nm^{\log_2 3-1})$.

Solution: **************** INSERT PROBLEM 3 SOLUTION HERE ************** □

CSCI-GA.1170-001/002 Fundamental Algorithms September 17, 2014

Solutions to Problem 4 of Homework 3 (12 points)

*Name: **** INSERT YOUR NAME HERE **** Due: Wednesday, September 24*

An array $A[0 \ldots (n-1)]$ is called *rotation-sorted* if there exists some some cyclic shift $0 \leq c < n$ such that $A[i] = B[(i + c \bmod n)]$ for all $0 \leq i < n$, where $B[0 \ldots (n-1)]$ is the sorted version of A.[1] For example, $A = (2, 3, 4, 7, 1)$ is rotation-sorted, since the sorted array $B = (1, 2, 3, 4, 7)$ is the cyclic shift of A with $c = 1$ (e.g. $1 = A[4] = B[(4+1) \bmod 5] = B[0] = 1$). For simplicity, below let us assume that n is a power of two (so that can ignore floors and ceilings), and that all elements of A are distinct.

(a) (4 points) Prove that if A is rotation-sorted, then one of $A[0 \ldots (n/2-1)]$ and $A[n/2 \ldots (n-1)]$ is fully sorted (and, hence, also rotation-sorted with $c = 0$), while the other is at least rotation-sorted. What determines which one of the two halves is sorted? Under what condition *both halves* of A are sorted?

Solution: **************** INSERT PROBLEM 4a SOLUTION HERE *************** □

(b) (8 points) Assume again that A is rotation-sorted, but you are not given the cyclic shift c. Design a divide-and-conquer algorithm to compute the minimum of A (i.e., $B[0]$). Carefully prove the correctness of your algorithm, write the recurrence equation for its running time, and solve it. Is it better than the trivial $O(n)$ algorithm? (**Hint**: Be careful with $c = 0$ an $c = n/2$; you might need to handle them separately.)

Solution: ***************** INSERT PROBLEM 4b SOLUTION HERE *************** □

[1] Intuitively, A is either completely sorted (if $c = 0$), or (if $c > 0$) A starts in sorted order, but then "falls off the cliff" when going from $A[n - c - 1] = B[n - 1] = \max$ to $A[n - c] = B[0] = \min$, and then again goes in increasing order while never reaching $A[0]$.

CSCI-GA.1170-001/002 Fundamental Algorithms September 17, 2014

Solutions to Problem 5 of Homework 3 (5 Points)

*Name: **** INSERT YOUR NAME HERE **** Due: Wednesday, September 24*

Find a *divide-and-conquer* algorithm that finds the maximum and the minimum of an array of size n using at most $3n/2$ comparisons.
(**Hint**: First, notice that we are not asking for some iterative algorithm (which is not hard). We are asking for you to *explicitly use recursion*. In fact, your divide/conquer step should take time $O(1)$. Also, you have to be super-precise about constants and the initial case $n = 2$ to get the correct answer.)

Solution: **************** INSERT PROBLEM 5 SOLUTION HERE ************** □