5DV022 Problem Exercise 2 Fall 2008

Solutions to this assignment are due on September 18, 2008 at 1700 (5pm). The signed cover sheet must be turned in with your solutions.

Turn in your solutions to the course instructor. Do not put them in the red mailboxes for laboratory reports.

1. Define a modified Fibonacci sequence g as follows:

$$\begin{aligned} g(0) &= 0 \\ g(1) &= 1 \\ g(n+2) &= (2 \cdot g(n+1)) + g(n) \end{aligned}$$

The "obvious" iterative algorithm will compute $g(n)$ in time $\Theta(n)$. However, by using the divide-and-conquer strategy, it is possible to do better. You are to elaborate this improvement, proceeding as follows.

(a) Solve the recurrence relation for $g(n)$ which is given above. Do not worry about an algorithm at this point; just obtain a closed form for the value of $g(n)$. This means a complete solution, including constants. The solution will involve terms in $\sqrt{2}$; even if you use a numerical tool such as *octave*, you should convert it to a form involving $\sqrt{2}$. This form should be obvious from the numerical answer.

(b) Use the closed form for $g(n)$ developed in part (a) as the basis for a divide-and-conquer solution, by breaking the exponentiation (raising to the nth power) parts into simpler subproblems. Roughly, the idea is to exploit the fact that for any number a and any positive integer n

$$a^n = \begin{cases} (a^{\lfloor n/2 \rfloor}) \cdot (a^{\lfloor n/2 \rfloor}) & \text{if } n \text{ is even} \\ (a^{\lfloor n/2 \rfloor}) \cdot (a^{\lfloor n/2 \rfloor}) \cdot a & \text{if } n \text{ is odd} \end{cases}$$

to obtain a way to compute a^n with far fewer than n multiplications. Your solution must realize the exponentiation using the basic arithmetic operations of addition, subtraction, multiplication, and division; exponentiation may not be taken as a basic operation. Express your algorithm in an imperative notation, similar to that used in the course notes.

(c) Give and solve the recurrence relation which describes the running time of your algorithm. (This will not be the same as the recurrence which describes g.) Using this recurrence, provide an analysis of the worst-case running time of your algorithm. (It should be $\Theta(\log(n))$.)

(d) Implement your algorithm in the programming language C, and compute the values for g(n) for $0 \leq n \leq 20$. To avoid any possible size errors, use `long int` declarations for the variables. Turn in a source-code listing, as well as a printout of the computed results.

(e) Also implement the obvious iterative solution in C, and compute the same values, for comparison. (This is only a comparison to check the values; there is no need to profile the relative performances of the two algorithms.) Turn in a source-code listing, as well as a printout of the computed results.