Written Assignment 1 - Solutions

Vancouver Summer Program – Algorithms – UBC

---

1. (Enter Fibonacci) The Fibonacci sequence is defined as follows: $F_0 = F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all integers $n \geq 2$.

   (a) You are to derive an efficient algorithm to compute the nth Fibonacci number. Observe that

   $$F_n = F_{n-1} + F_{n-2}$$
   $$F_{n-1} = F_{n-1} + 0 \cdot F_{n-2}.$$

   If we write this linear system in terms of matrices, we have

   $$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}$$

   Using this linear relation, derive an algorithm to compute F_n. Your algorithm should run in time $O(\log n)$.
   Hint: Use repeated squaring to compute matrix powers.

   Solution. Unfolding the given linear recurrence, we have

   $$\begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix},$$

   which gives us

   $$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix}.$$

   Continuing in this manner recursively, we get, for every $n \geq 1$,

   $$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

   so F_n is the top entry of the vector $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, with the understanding that for any matrix A, $A^0 = I$, the identity matrix. Then for a given $n \geq 1$, one needs to compute the $(n-1)$st power of $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. This can be done using repeated squaring with $\Theta(\log_2 n)$ matrix multiplications, and since our matrices are 2×2 and are thus of *constant* size (relative to the input n), multiplying two matrices requires a constant number (in n) of integer additions and multiplications. Thus computing F_n requires $\Theta(\log n)$ additions/multiplications, which is indeed polynomial in the size of the input, $\log_2 n$.

   (b) Now suppose that writing every bit of the output to memory counts as an operation that we wish to account for in our running-time analysis (in the previous part, we disregard the time required to write the output to memory). Can you compute F_n in time that is bounded by a polynomial in the size of the input? Justify your answer.

Solution. We can show by induction that $F_n \geq 2^{n/2}$ for all $n \geq 6$. Thus the number of bits required to encode the output in binary is at least $\log_2 2^{n/2} = n/2$. Since writing each bit to memory counts as an operation here, we will need at least $n/2$ operations to write the entirety of F_n. Thus $T(n) \geq n/2$. The input is an integer $n \geq 0$ so the size of the input is $\mathtt{size}(n) = \log_2 n$ bits. But $n = 2^{\log_2(n)} = 2^{\mathtt{size}(n)}$, which is an exponential function in $\mathtt{size}(n)$. Thus $T(n) \geq \frac{1}{2}2^{\mathtt{size}(n)}$, so $T(n)$ cannot be a polynomial in $\mathtt{size}(n)$.

(c) **(Bonus)** Find a if a and b are integers such that $x^2 - x - 1$ is a factor of $ax^{17} + bx^{16} + 1$. **Hint**: The answer is F_n for some $n \geq 1$. It is enough to show this and find n explicitly; you do not need to compute F_n.

Solution. Here is one possible solution. Let's work backwards! Let $F(x) = ax^{17} + bx^{16} + 1$ and let $P(x)$ be the polynomial such that $P(x)(x^2 - x - 1) = F(x)$.
Clearly, the constant term of $P(x)$ must be -1. Now, we have $(x^2 - x - 1)(c_1 x^{15} + c_2 x^{14} + \cdots + c_{15}x - 1)$, where c_i is some coefficient. However, since $F(x)$ has no x term, it must be true that $c_{15} = 1$.
Let's find c_{14} now. Notice that all we care about in finding c_{14} is that $(x^2 - x - 1)(\cdots + c_{14}x^2 + x - 1) = \text{something} + 0x^2 + \text{something}$. Therefore, $c_{14} = -2$. Undergoing a similar process, $c_{13} = 3$, $c_{12} = -5$, $c_{11} = 8$, and we see a nice pattern. The coefficients of $P(x)$ are just the Fibonacci sequence with alternating signs! Therefore, $a = c_1 = F_{16}$, where F_{16} denotes the 16th Fibonnaci number and $a = 987$.

2. (Time Complexity)

(a) Algorithms A and B spend exactly $T_A(n) = 0.1n^2 \log_{10}(n)$ and $T_B(n) = 2.5n^2$ microseconds, respectively, for a problem of size n. Choose the algorithm, which is better in the Big-Oh sense, and find out a problem size n_0 such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of the size $n \leq 10^9$, which algorithm will you recommend to use?

Solution. In the Big-Oh sense, algorithm **B** is better. It outperforms algorithm **A** when $T_{\mathbf{B}}(n) \leq T_{\mathbf{A}}(n)$, that is, when $2.5n^2 \leq 0.1n^2 \log_{10} n$. This inequality reduces to $\log_{10} n \geq 25$, or $n \geq n_0 = 10^{25}$. If $n \leq 10^9$, the algorithm of choice is **A**.

(b) Let $f(n) = (\log n)^{\log n}$ and $g(n) = 2^{(\log_2 n)^2}$. Determine whether $f \in O(g)$, $f \in \Omega(g)$, or both (in which case $f \in \Theta(g)$).

Solution. $g(n) = 2^{(\log_2 n)^2} = 2^{(\log_2 n)(\log_2 n)} = \left(2^{\log_2 n}\right)^{\log_2 n} = n^{\log_2 n}$. Thus, $f \in O(g)$ (in fact, $f \in o(g)$), but $f \notin \Omega(g)$.

(c) Show that for any $f, g : \mathbb{Z}_+ \to \mathbb{R}_+$, $O(f+g) = O\big(\max\{f,g\}\big)$. Recall that $O(\cdot)$ is a set (see notes #1), and therefore one has to show both $O(f+g) \subset O\big(\max\{f,g\}\big)$ and $O\big(\max\{f,g\}\big) \subset O(f+g)$.

Solution. If $h \in O(f+g)$, then there is $c > 0$ and $N \in \mathbb{N}$ such that $h(n) \leq c\big(g(n) + f(n)\big)$ for all $n \geq N$. But

$$\max\{f,g\} = \frac{f + g + |f-g|}{2},$$

from which it follows that $f + g \leq 2\max\{f,g\} - |f-g| \leq 2\max\{f,g\}$. Thus we may take $c' = 2c$ and $N' = N$ so that $h(n) \leq c' \max\{g(n), f(n)\}$ for all $n \geq N'$. For the other direction, if $h \in O(\max\{f,g\})$, then there is $c > 0$ and $N \in \mathbb{N}$ for which $h(n) \leq c\max\{f(n), g(n)\}$. Then $h(n) \leq cf(n)$ and $h(n) \leq cg(n)$ for all $n \geq N$. Thus $2h(n) \leq c\big(f(n) + g(n)\big)$ for all $n \geq N$, so we may take $c' = c/2$ and $N' = n$.