CSOR W4246 – Fall, 2015

## Homework 1

1. (10 points)

   (a) Let $f_1(n) = n$. Then $2n \leq c \cdot n$ for all $n \geq n_0 = 1$ and $c = 2$.

   (b) Let $f_1(n) = 2^n$. Then $\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty$. Hence $2^{2n} = \omega(2^n)$.

2. (20 points)

   (a) $T(n) = O(n^2 \log n)$

   (b) $T(n) = O(n^3 \log n)$

   (c) $T(n) = O(n^2)$

   (d) $T(n) = O(n^{\log_3 7})$

3. (20 points) The recurrence for this algorithm is

$$T(n) = 3T(2n/3) + \Theta(1) = \Theta(n^{\log_{3/2} 3}) = \Theta(n^{\frac{\log 3}{\log 1.5}}) = \omega(n^2)$$

   Hence both insertion sort and merge Sort are faster than this algorithm.

4. (25 points)

| $f$ | $g$ | $O$ | $o$ | $\Omega$ | $\omega$ | $\Theta$ |
|---|---|---|---|---|---|---|
| $10 \log n$ | $\log^3 n$ | yes | yes | no | no | no |
| $n \log(2n)$ | $n \log n$ | yes | no | yes | no | yes |
| $\sqrt{\log n}$ | $\log \log n$ | no | no | yes | yes | no |
| $10n^2 + \log n$ | $n^2 + 11 \log^3 n$ | yes | no | yes | no | yes |
| $\sqrt{n} + \log n$ | $n^{2/3} + 10$ | yes | yes | no | no | no |
| $n^2 2^n$ | $3^n$ | yes | yes | no | no | no |
| $n^{1/3}$ | $(\log n)^2$ | no | no | yes | yes | no |
| $n \log n$ | $\frac{n^2}{\log n}$ | yes | yes | no | no | no |
| $n!$ | $n^n$ | yes | yes | no | no | no |
| $\log n!$ | $\log n^n$ | yes | no | yes | no | yes |

5. (30 points)

   (a) Fibonacci($n$)

   **if** $n \leq 1$ **then**
   
   `return` $n$
   
   **else**
   
   `return` Fibonacci($n-1$)+ Fibonacci($n-2$)
   
   **end if**

The running time can be expressed as

$$T(n) = T(n-1) + T(n-2) + O(1) \geq 2T(n-2) + O(1)$$

Note that $F_n = F_{n-1} + F_{n-2} \geq 2F_{n-2}$. Hence $T(n) \geq F_n = \Omega(2^{n/2})$. (A formal proof of this inequality follows by strong induction on $n$.)

(b) Fibonacci($n$)

**if** $n \leq 1$ **then**
  `return` $n$
**end if**
$a = 0$
$b = 1$
**for** $i = 2, \ldots, n$ **do**
  Fi = a+b
  b= Fi
  a = b
**end for**
`return` Fi

The running time of this algorithm is $O(n)$: for every value of $i$, the body of the while loop requires $O(1)$ time and $i$ ranges from 2 to $n$.

(c) We have

$$\begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^2 \begin{bmatrix} F_{n-3} \\ F_{n-2} \end{bmatrix} = \cdots = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}$$

If we use divide and conquer,

- for odd $n$, $M^n = M \times M^{\frac{n-1}{2}} \times M^{\frac{n-1}{2}}$
- for even $n$, $M^n = M^{\frac{n}{2}} \times M^{\frac{n}{2}}$

We keep shrinking the size of the subproblems by 2 until we get subproblems of size 1. Therefore the recurrence is

$$T(n) = T(n/2) + O(1) = O(\log n).$$

(d) Since $F_n \geq 2^{n/2}$, it has at least $n/2$ bits. Recall that the time to add 2 $\Theta(n)$-bit numbers is $\Theta(n)$, while the time required to multiply 2 $\Theta(n)$-bit numbers is $\Theta(n^2)$.

i. $T(n) = T(n-1) + T(n-2) + \Theta(n) = \Omega(2^{n/2})$. So this approach is still inefficient. (Tighter lower bounds can be computed.)

ii. $T(n) = T(n-1) + \Theta(n) = O(n^2)$.

iii. To multiply two matrices, 8 multiplications and 4 additions are required. Since the time to multiply 2 $n$-bit numbers ($O(n^2)$) **dominates** the time to add 2 $n$-bit numbers ($O(n)$), we only need consider the number of multiplications. Since the elements in the matrices have $\Theta(n)$ bits, it takes $\Theta(n^2)$ time to multiply 2 matrices. Therefore

$$T(n) = T(n/2) + \Theta(n^2) = O(n^2).$$