Pauta Prueba 1

Analisis de Algoritmos (INF-648)
Segundo Semestre 2011

Ejercicio 1

Usando sustitucion hacia atras, resuelva la siguiente ecuacion de recurrencia:

$$T(n) = 8T\left(\frac{n}{2}\right) + n^3, \qquad T(1) = 1$$

Solucion

$$T(n) = 8T\left(\frac{n}{2}\right) + n^3$$

$$\begin{aligned} T\left(\frac{n}{2}\right) &= 8T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^3 \\ T(n) &= 8\left(8T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right)^3\right) + n^3 \\ &= 8^2 T\left(\frac{n}{2^2}\right) + 2n^3 \end{aligned}$$

$$\begin{aligned} T\left(\frac{n}{2^2}\right) &= 8T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^3 \\ T(n) &= 8^2\left(8T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^3\right) + 2n^3 \\ &= 8^3 T\left(\frac{n}{2^3}\right) + 3n^3 \qquad \textbf{0.5 pts} \end{aligned}$$

El caso general es:

$$T(n) = 8^k T\left(\frac{n}{2^k}\right) + kn^3 \qquad \textbf{0.2 pts}$$

Aplicando la condicion inicial $T(1) = 1 \rightarrow \frac{n}{2^k} = 1 \rightarrow k = \log_2 n$, por ende:

$$\begin{aligned} T(n) &= 8^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n^3 \log_2 n \\ &= \left(2^{\log_2 n}\right)^3 T(1) + n^3 \log_2 n \\ &= n^3(1 + \log_2 n) \qquad \textbf{0.3 pts} \end{aligned}$$

Ejercicio 2

Suponga que podemos calcular c^n en tiempo $O(\log n)$, con c constante, es posible calcular el n-esimo termino de la serie de fibonaci en tiempo $O(\log n)$? Si es asi, construya un algoritmo que lo haga. La serie de fibonacci esta definida por:

$$F_n = F_{n-1} + F_{n-2}, \qquad F_0 = 0, F_1 = 1$$

Solucion

Si resolvemos la ecuacion de recurrencia de Fibonacci usando la formula para ecuaciones lineales homogeneas, se obtiene:

$$T(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \qquad \textbf{0.5 pts}$$

Para evaluar esta ecuacion, se requiere calcular c^n 2 veces y realizar algunas operaciones de sumas y restas que toman una cantidad de tiempo constante. Como se supone que le costo de calcular c^n es $O(\log n)$, entonces el costo de evaluar $T(n)$ es:

$$\text{Costo}(T(n)) = 2\log n + O(1) \qquad \textbf{0.2 pts}$$

Aplicando la O:

$$O(2\log n + 1) = O(\log n) \qquad \textbf{0.3 pts}$$

Entones **si es posible** calcular la serie de Fibonacci en tiempo $O(\log n)$.

Ejercicio 3-a

Demuestre que:

$$n\log n \in O(n^2)$$

Solucion

Para demostrar $n\log n \in O(n^2)$, debemos calcular:

$$\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} \overset{L'Hopital}{=} \lim_{n\to\infty} \frac{1}{n} = 0 \qquad \textbf{0.5 pts}$$

Como el limite es cero, se concluye que $n\log n \in O(n^2)$, lo que completa la demostracion **0.5 pts**.

Ejercicio 3-b

Demuestre que:

$$n! \in \Omega(e^n)$$

Solucion

Recordemos la Aproximacion de Stirling:

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

Entonces, para demostrar $n! \in \Omega(e^n)$, debemos calcular:

$$\lim_{n\to\infty} \frac{n!}{e^n} = \lim_{n\to\infty} \frac{\left(\frac{n}{e}\right)^n \sqrt{2\pi n}}{e^n} = \lim_{n\to\infty} \left(\frac{n}{e^2}\right)^n \sqrt{2\pi n} = \infty \qquad \textbf{0.5 pts}$$

Como el limite es infinito, se concluye que $n! \in \Omega(e^n)$, lo que completa la demostracion **0.5 pts**.

Ejercicio 4

Para el siguiente algoritmo, plantee una ecuacion de recurrencia que le permita obtener la complejidad computacional de dicho algoritmo, y resuelvala. Presente la solucion final usando la notacion O.

```
template <typename T>
void fusion(Arreglo<T>& v, int inicio, int med, int fin) {
    Arreglo<T> aux(fin - inicio + 1);
    int i = inicio;
    int j = med + 1;
    int k = 0;

    while (i <= med && j <= fin) {
        if (v[i] < v[j]) {
            aux[k] = v[i];
            i++;
        } else {
            aux[k] = v[j];
            j++;
        }
        k++;
    }

    while (i <= med) {
        aux[k] = v[i];
        i++;
        k++;
    }

    while (j <= fin) {
        aux[k] = v[j];
        j++;
        k++;
    }

    for (int n = 0; n < aux.largo(); ++n) { v[ini + n] = aux[n]; }
}

template<typename T>
void mergeSort(Arreglor<T>& v, int inicio, int fin) {

    if (inicio < fin) {
        int med = (inicio + fin) / 2;

        mergeSort(v, inicio, med);
        mergeSort(v, med + 1, fin);
        fusion(v, inicio, med, fin);
    }
}
```

Solucion

La ecuacion de recurrencia que representa la complejidad algoritmica de MergeSort es:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad \textbf{0.7 pts}$$

Esta ecuacion se construye considerando las 2 llamadas recursivas, cada una de las cuales procesa exactamente la mitad de los elementos del arreglo, por ende se desprende el termino $2T\left(\frac{n}{2}\right)$. Finalmente, cada llamada a la funcion fusion toma tiempo lineal, lo cual contribuye el termino lineal de la ecuacion (Cualquier termino lineal no cambia la solucion en forma radical).

Podemos resolver la ecuacion de recurrencia usando sustitucion hacia atras:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$\begin{aligned} T\left(\frac{n}{2}\right) &= 2T\left(\frac{n}{2^2}\right) + \frac{n}{2} \\ T(n) &= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n \\ &= 2^2 T\left(\frac{n}{2^2}\right) + 2n \end{aligned}$$

$$\begin{aligned} T\left(\frac{n}{2^2}\right) &= 2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} \\ T(n) &= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n \\ &= 2^3 T\left(\frac{n}{2^3}\right) + 3n \qquad \textbf{0.3 pts} \end{aligned}$$

El caso general es:

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn \qquad \textbf{0.3 pts}$$

Aplicando la condicion inicial $T(1) = c \rightarrow \frac{n}{2^k} = 1 \rightarrow k = \log_2 n$, por ende:

$$\begin{aligned} T(n) &= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n\log_2 n \\ &= \left(2^{\log_2 n}\right) T(1) + n\log_2 n \\ &= n(c + \log_2 n) \qquad \textbf{0.3 pts} \end{aligned}$$

Finalmente, la complejidad de Mergesort es:

$$T(n) \in O(n\log_2 n) \qquad \textbf{0.4 pts}$$