Algoritmos
y
Estructuras de Datos

Cursada 2011

Resolver la recurrencia (Ej 12.4)

$$T(n) = \begin{cases} 1, n = 1 \\ 8T\left(\frac{n}{2}\right) + n^3, n \geq 2 \end{cases}$$

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

$$T(n) = 8\left[8T\left(\frac{\frac{n}{2}}{2}\right) + \left(\frac{n}{2}\right)^3 \right] + n^3, n \geq 4$$

$$T(n) = 8\left[8T\left(\frac{n}{4}\right) + \frac{n^3}{2^3} \right] + n^3, n \geq 4$$

Resolver la recurrencia (Ej 12.4)

$$T(n) = 8^2 T\left(\frac{n}{4}\right) + 8\frac{n^3}{2^3} + n^3, n \geq 4$$

$$T(n) = 8^2 T\left(\frac{n}{2^2}\right) + n^3 + n^3, n \geq 4$$

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

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

Resolver la recurrencia (Ej 12.4)

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

$$\frac{n}{2^i} = 1$$

$$n = 2^i$$

$$i = Log_2(n)$$

Resolver la recurrencia (Ej 12.4)

$$T(n) = 8^{Log_2(n)} T\left(\frac{n}{2^{Log_2(n)}}\right) + Log_2(n) n^3$$

$$T(n) = n^{Log_2(8)} T\left(\frac{n}{n}\right) + Log_2(n) n^3$$

$$T(n) = n^3 T(1) + Log_2(n) n^3$$

$$T(n) = n^3 + Log_2(n) n^3$$

Peor caso

```
/* Secuencia creciente continua de max longitud. */
public static int f5a(int[] datos, int n) {
        int m = 0;
        for (int i = 0; i < n; i++) {
        /* La secuencia puede comenzar en cualquiera de las posiciones*/
                int k = i + 1;
                while (k < n && datos[k] >= datos[k - 1]) {
                        k = k + 1;
                /* es necesario verificar donde termina, para cada posible
comienzo*/
                }

                if (k - i > m)
                        m = k - i;
        }
        return m;
}
```

Peor caso

$$T_1(n) = c + \sum_{i=1}^{n} \left(d + \sum_{k=i+1}^{n} e \right) =$$

$$= c + nd + e \sum_{i=1}^{n} (n - i) =$$

$$= c + nd + en^2 - e\frac{n(n+1)}{2} =$$

$$= c + nd + en^2 - \frac{en^2}{2} - \frac{en}{2} =$$

$$= c + \frac{en^2}{2} + \left(d - \frac{e}{2} \right) n$$

Peor caso

- Si bien en la diapositiva previa se calculó el T(n) del algoritmo, el tiempo real depende del orden de los datos, puesto que si la secuencia está en orden decreciente, el while interno nunca se itera (Recordar la noción del peor de los casos).
- A continuación se brinda otro algoritmo que no posee el while interno.

Peor caso

```
/* El while interno era redundante. */
public static int f5b(int[] datos, int n) {
        int m = 0, ant, act;
        ant = 1;
        for (int i = 1; i < n; i++) {
                if (datos[i] >= datos[i - 1]) {
                        act = ant + 1;
                } else {
                        act = 1;
                }

                if (act > m)
                        m = act;
                ant = act;
        }
        return m;
}
```

Peor caso

- El T(n) de este algoritmo es lineal.

$$T_2(n) = c + \sum_{i=1}^{n} d = c + nd$$

Tiempo real de ejecución

- Sean 2 algoritmos de ordenación:
  - Selección ($O(n^2)$)
  - Mergesort ($O(n*Log(n))$)

- A continuación se muestran datos reales de tiempo que ejecución que confirman los tiempos teóricos

Tiempo real de ejecución

Cantidad (N)	Selección (milisegundos)	Merge (milisegundos)
5000	47	0
80000	9391	15
155000	25188	31
230000	55782	47
305000	100312	62
380000	158875	78
455000	229985	109

Tiempo real de ejecución