Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

Arturo Olivares Martos

Granada, 2023-24

Índice general

1. Relaciones de Problemas

1.1. La eficiencia de los algoritmos

Ejercicio 1.1.1. Demostrar las siguientes propiedades:

a) $k \cdot f(n) \in O(f(n)), \qquad \forall k > 0.$

b) $n^r \in O(n^k)$ si $0 \leqslant r \leqslant k$.

c) $O(n^k) \subset O(n^{k+1})$.

d) $n^k \in O(b^n) \quad \forall b > 1, k \geqslant 0.$

e) $\log_b n \in O(n^k) \quad \forall b > 1, k > 0.$

f) Si $f(n) \in O(g(n))$ y $h(n) \in O(g(n))$, entonces $f(n) + h(n) \in O(g(n))$.

g) Si $f(n) \in O(g(n))$, entonces $f(n) + g(n) \in O(g(n))$.

h) *Reflexividad*: $f(n) \in O(f(n))$.

i) *Transitividad*: Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$, entonces $f(n) \in O(h(n))$.

j) *Refla de la suma*: Si $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$, entonces:

$$T1(n) + T2(n) \in O(\text{máx}\{f(n), g(n)\}).$$

k) *Refla del producto*: Si $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$, entonces:

$$T1(n) \cdot T2(n) \in O(f(n) \cdot g(n)).$$

Ejercicio 1.1.2. Expresar, en notación $O(\cdot)$, el orden que tendrí un algoritmo cuyo tiempo de ejecución fuera $f_i(n)$, donde:

1. $f_1(n) = n^2$.

2. $f_2(n) = n^2 + 1000n$.

3. $f_3(n) = \begin{cases} n & \text{si } n \text{ es par} \\ n^3 & \text{si } n \text{ es impar} \end{cases}$

4. $f_n4(n) = \begin{cases} n & \text{si } n \leqslant 100 \\ n^3 & \text{si } n > 100 \end{cases}$

5. $f_5(n) = (n-1)^3$.

6. $f_6(n) = \sqrt{n^2 - 1}$.

7. $f_7(n) = \log(n!)$.

8. $f_8(n) = n!$.

Ejercicio 1.1.3. Usando la notación $O(\cdot)$, obtener el tiempo de ejecución de las siguientes funciones:

1. Código Fuente 1 (`ejemplo1`).

```
void ejemplo1 (int n)
{
    int i, j, k;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        {
            C[i][j] = 0;
            for (k = 0; k < n; k++)
                C[i][j] += A[j][k] * B[k][j];
        }
}
```

Código fuente 1: Función del Ejercicio 1.1.3 apartado 1.

2. Código Fuente 2 (`ejemplo2`).

```
long ejemplo2 (int n)
{
    int i, j, k;
    long total = 0;

    for (i = 0; i < n; i++)
        for (j = i+1; j <= n; j++)
            for (k = 1; k <= j; k++)
                total += k*i;

    return total;
}
```

Código fuente 2: Función del Ejercicio 1.1.3 apartado 2.

3. Código Fuente 3 (`ejemplo3`).

```
void ejemplo3 (int n)
{
    int i, j, x=0, y=0;

    for (i = 1; i <= n; i++)
        if (i % 2 == 1)
        {
            for (j = i; j <= n; j++)
                x++;
            for (j = 0; j < i; j++)
                y++;
        }
}
```

Código fuente 3: Función del Ejercicio 1.1.3 apartado 3.

4. Código Fuente 4 (`ejemplo4`).

```
int ejemplo4 (int n)
{
    if (n <= 1)
        return 1;
    else
        return (ejemplo4(n - 1) + ejemplo4(n-1));
}
```

Código fuente 4: Función del Ejercicio 1.1.3 apartado 4.

5. Código Fuente 5 (`ejemplo5`).

```
int ejemplo5 (int n)
{
    if (n == 1)
        return n;
    else
        return (ejemplo5(n/2) + 1);
}
```

Código fuente 5: Función del Ejercicio 1.1.3 apartado 5.

Ejercicio 1.1.4. Resolver las siguientes recurrencias:

a) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2T(n-1) + 1 & \text{en otro caso} \end{cases}$

b) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2T(n-1) + n & \text{en otro caso} \end{cases}$

c) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ T(n-1) + T(n-2) & \text{en otro caso} \end{cases}$

d) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 3T(n-1) + 4T(n-2) & \text{en otro caso} \end{cases}$

e) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 5T(n-1) - 8T(n-2) + 4T(n-3) & \text{en otro caso} \end{cases}$

f) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 36 & \text{si } n = 1 \\ 5T(n-1) + 6T(n-2) + 4 \cdot 3^n & \text{en otro caso} \end{cases}$

g) $T(n) = 2T(n-1) + 3^n$.

h) $T(n) = 2T(n-1) + n + 2^n$.

i) $T(n) = 2T\left(n/2\right) + \log n$.

j) $T(n) = 4T\left(n/2\right) + n$.

k) $T(n) = 4T\left(n/2\right) + n^2$.

l) $T(n) = 2T\left(n/2\right) + n\log n$.

m) $T(n) = \begin{cases} 1 & \text{si } n = 2 \\ 2T\left(\sqrt{n}\right) + \log n & \text{si } n \geqslant 4 \end{cases}$

n) $T(n) = \begin{cases} 1 & \text{si } n = 2 \\ 2T\left(\sqrt{n}\right) + \log\log n & \text{si } n \geqslant 4 \end{cases}$

o) $T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 5T\left(n/2\right) + (n\log n)^2 & \text{si } n \geqslant 2 \end{cases}$

p) $T(n) = \sqrt{n}T\left(\sqrt{n}\right) + n, \qquad n \geqslant 4.$

q) $T(n) = \begin{cases} 6 & \text{si } n = 1 \\ nT^2\left(n/2\right) & \text{si } n > 1 \end{cases}.$

r) $T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 4 & \text{si } n = 2 \\ T\left(n/2\right) \cdot T^2\left(n/2\right) & \text{si } n \geqslant 4 \end{cases}$

Ejercicio 1.1.5. El tiempo de ejecución de un Algotimo A viene descrito por la recurrencia
$$T(n) = 7T\left({}^{n}/_{2}\right) + n^2$$

Otro algoritmo B tiene un tiempo de ejecución descrito por la recurrencia

$$T'(n) = aT'\left({}^{n}/_{4}\right) + n^2$$

¿Cuál es el mayor valor de la constante $a \in \mathbb{R}$ que hace al algoritmo B asintóticamente más eficiente que A?

Ejercicio 1.1.6. Resuelva la siguiente recurrencia:

$$T(n) = aT\left(\frac{n}{b}\right) + n^k$$

con $a, b, k \in \mathbb{R}$, $a \geqslant 1$, $b \geqslant 2$, $k \geqslant 0$.