



## Práctica 7

En teoría se han visto varios conceptos: complejidad de una función, cotas de la complejidad ( $\Theta$ ,  $O$ ), análisis en peor (o mejor) caso. Asegúrese comprender la diferencia entre estos conceptos.

En general, queremos saber cuál es la fc. complejidad y tener una descripción del comportamiento asintótico ( $\Theta$ ). En caso de no poder dar una respuesta general, buscaremos una cota superior en su comportamiento asintótico ( $O$ ), empeorando la función. Además, muchas veces podemos estudiar el problema restringiéndonos a cuál sería el peor caso de entrada, obteniendo también una cota superior del algoritmo original.

1. Siguiendo la formalización matemática, probar:

- $\Theta$  es simétrica: si  $f \in \Theta(g)$  entonces  $g \in \Theta(f)$ .
- $O$  es transitiva: si  $f \in O(g)$  y  $g \in O(h)$  entonces  $f \in O(h)$ .
- Si  $f \in O(g)$  entonces  $kf \in O(g)$  con  $k > 0$ .
- Existen  $f$  y  $g$ , tales que  $f \notin O(g)$  y  $g \notin O(f)$ .

2. Considere la siguiente notación para el conjunto de endofunciones sobre  $\mathbb{N}$ :

$$\mathbb{N}^{\mathbb{N}} = \{f \mid f : \mathbb{N} \rightarrow \mathbb{N}\}$$

Definimos a  $\Theta(g)$  como:

$$\Theta(g) = \{f \in \mathbb{N}^{\mathbb{N}} \mid \exists c_1, c_2 > 0 \cdot \exists n_0 \in \mathbb{N} \cdot \forall n \in \mathbb{N} \cdot \text{si } n_0 \leq n, \text{ entonces } c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Dar el dominio y codominio de  $\Theta$  como función.

3. Así como  $O$  refleja la cota superior de  $\Theta$ , se suele introducir un operador  $\Omega$  para reflejar una cota inferior. Defina  $\Omega$  formalmente, y pruebe que es transitivo.

4. El teorema maestro nos brinda la solución para calcular el comportamiento asintótico de una gran cantidad de algoritmos recursivos. Dada una ecuación de la forma

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \text{ con } a \geq 1, b > 1$$

el teorema maestro nos asegura que:

- Si  $f \in \Theta(n^c)$  donde  $c < \log_b a$ , entonces  $T \in \Theta(n^{\log_b a})$ .
- Si  $f \in \Theta(n^c \log^k n)$  donde  $c = \log_b a$  y  $k \geq 0$ , entonces  $T \in \Theta(n^c \log^{k+1} n)$ .
- Si  $f \in \Theta(n^c)$  donde  $c > \log_b a$ , entonces  $T \in \Theta(n^c)$ .

Resolver, utilizando el teorema maestro, las siguientes ecuaciones:

- $T(n) = 2T\left(\frac{n}{2}\right) + 5$
- $U(n) = 2U\left(\frac{n}{2}\right) + n + 2$
- $V(n) = 3V\left(\frac{n}{2}\right) + n^2 + 435n$

5. Calcular, para el peor y mejor caso, la complejidad del ordenamiento burbuja, inserción y selección.

6. Analizar la complejidad de las funciones vistas para:

- a) Comparar dos cadenas.
- b) Hashear un natural.
- c) Hashear una cadena.
- d) Agregar un elemento al inicio de una lista simplemente enlazada.
- e) Agregar un elemento al final de una lista simplemente enlazada.
- f) Visitar cada nodo de una lista simplemente enlazada.

7. Analizar la complejidad de cada una de estas funciones:

```
int f(int data[], int sz) {
    int i, r = 0;
    for (i = 0; i < sz; i++)
        r += data[i];
    return r;
}
int g(int data[], int sz) {
    return f(data, sz)/sz;
}
void d(int data[], int sz) {
    int i;
    for (i = 0; i < sz; i++)
        data[i]--;
    for (i = 0; i < sz; i++)
        data[i]++;
}
int h(int n) {
    int i, j, k, r = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                r += i + j + k;
    return r;
}
int i(int x) {
    int r = 0;
    if (x % 2 == 0)
        return 42;
    for (; x > 0; x--)
        r++;
    return r;
}
int lo(int i, int b) {
    if (i % b != 0)
        return 0;
    return 1 + lo(i/b, b);
}
int bin(int d[], int i,
        int l, int x) {
    int m = (i + l)/2;
    if (l < i)
        return 0;
    if (i == l)
        return d[i] == x;
    if (d[m] >= x)
        return bin(d, i, m, x);
    else
        return bin(d, m + 1, l, x);
}
```

8. Analice la complejidad del algoritmo iterativo para calcular Fibonacci. Pruebe, utilizando inducción, que para algún  $n_0$  vale:

$$C_{\text{fibr}}(n) \geq 2^{n/2} \text{ para } n \geq n_0$$

donde  $C_{\text{fibr}}$  es el costo del algoritmo recursivo para calcular Fibonacci. Saque conclusiones.