Algoritmos y Estructuras de Datos

Guía Práctica 6
Complejidad

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. Probar utilizando las definiciones que $f \in O(h)$, sabiendo que:

a) $f, h : \mathbb{N} \to \mathbb{N}$ son funciones tales que $f(n) = n^2 - 4n - 2$ y $h(n) = n^2$.

b) $f, g, h : \mathbb{N} \to \mathbb{N}$ son funciones tales que $g(n) = n^k$, $h(n) = n^{k+1}$ y $f \in O(g)$.

c) $f, g, h : \mathbb{N} \to \mathbb{N}$ son funciones tales que $g(n) = \log n$, $h(n) = n$ y $f \in O(g)$.

Ejercicio 2. Probar utilizando las definiciones que:

a) $n^2 - 4n - 2 = O(n^2)$.

b) Para todo $k \in \mathbb{N}$ y toda función $f : \mathbb{N} \to \mathbb{N}$, si $f \in O(n^k)$, entonces $f \in O(n^{k+1})$.

c) Si $f : \mathbb{N} \to \mathbb{N}$ es tal que $f \in O(\log n)$, entonces $f \in O(n)$.

Ejercicio 3. Determinar la verdad o falsedad de cada una de las siguientes afirmaciones. **Justificar**.

a) $2^n = O(1)$.

b) $n = O(n!)$.

c) $n! = O(n^n)$.

d) $2^n = O(n!)$.

e) Para todo $i, j \in \mathbb{N}$, $i \cdot n = O(j \cdot n)$.

f) Para todo $k \in \mathbb{N}$, $2^k = O(1)$.

g) $\log n = O(n)$.

h) $n! = O(2^n)$.

i) $2^n n^2 = O(3^n)$.

j) Para toda función $f : \mathbb{N} \to \mathbb{N}$, $f = O(f)$.

Ejercicio 4.

a) ¿Qué significa, intuitivamente, $O(f) \subseteq O(g)$? ¿Qué se puede concluir acerca del crecimiento de f y g cuando, simultáneamente, tenemos $O(f) \subseteq O(g)$ y $O(g) \subseteq O(f)$?

b) ¿Cómo ordena por inclusión las siguientes familias de funciones?

- $O(1)$
- $O(x+1)$
- $O(x^2)$
- $O(\sqrt{x})$
- $O(1/x)$
- $O(x^x)$
- $O(\sqrt{2})$
- $O(\log x)$
- $O(\log^2 x)$
- $O(\log(x^2))$
- $O(\log \log x)$
- $O(x!)$
- $O(\log(x!))$
- $O(x \log x)$
- $O(1 + \sin^2 x)$

Ejercicio 5. Determinar el orden de complejidad temporal de peor caso de los siguientes algoritmos, asumiendo que todas las operaciones sobre arreglos y matrices toman tiempo $O(1)$. La complejidad se debe calcular en función de una medida de los parámetros de entrada, por ejemplo, la cantidad de elementos en el caso de los arreglos y matrices y el valor en el caso de parámetros naturales.

a) SUMATORIA, que calcula la sumatoria de un arreglo de enteros:

```
1: function SUMATORIA(arreglo A)
2:     int i, total;
3:     total := 0;
4:     for i := 0 ... Long(A) - 1 do
5:         total := total + A[i];
6:     end for
7: end function
```

b) SumatoriaLenta, que calcula la sumatoria de n, definida como la suma de todos los enteros entre 1 y n, de forma poco eficiente:

```
1: function SumatoriaLenta(nat n)
2:     int i, total;
3:     total := 0;
4:     for i := 1 ... n do
5:         for j := 1 ... i do
6:             total := total + 1;
7:         end for
8:     end for
9: end function
```

c) ProductoMat, que dadas dos matrices A (de $p \times q$) y B (de $q \times r$) devuelve su producto AB (de $p \times r$):

```
1: function ProductoMat(matriz A, matriz B)
2:     int fil, col, val, colAFilB;
3:     matriz res(Filas(A), Columnas(B));
4:     for fil := 0 ... Filas(A) - 1 do
5:         for col := 0 ... Columnas(B) - 1 do
6:             val := 0;
7:             for colAFilB := 0 ... Columnas(A) - 1 do
8:                 val := val + (A[fil][colAFilB] * B[colAFilB][col]);
9:             end for
10:            res[fil][col] := val;
11:        end for
12:    end for
13:    return res;
14: end function
```

Ejercicio 6. Determinar el orden de complejidad temporal de mejor y peor caso de los siguientes algoritmos, asumiendo que todas las operaciones sobre arreglos y matrices toman tiempo $O(1)$. La complejidad se debe calcular en función de una medida de los parámetros de entrada, por ejemplo, la cantidad de elementos en el caso de los arreglos y matrices y el valor en el caso de parámetros naturales.

a) InsertionSort, que ordena un arreglo pasado como parámetro:

```
1: function InsertionSort(arreglo A)
2:     int i, j, valor;
3:     for i := 0 ... Long(A) - 1 do
4:         valor := A[i];
5:         j := i - 1;
6:         while j ≥ 0 ∧ a[j] > valor do
7:             A[j+1] := A[j];
8:             j := j - 1;
9:         end while
10:        A[j+1] := valor;
11:    end for
12: end function
```

b) BúsquedaBinaria, que determina si un elemento se encuentra en un arreglo, que debe estar ordenado:

```
1: function BúsquedaBinaria(arreglo A, elem valor)
2:     int izq := 0, der := Long(A) - 1;
3:     while izq < der do
4:         int medio := (izq + der) / 2;
5:         if valor < A[medio] then
6:             der := medio;
7:         else
8:             izq := medio;
9:         end if
10:    end while
11:    return A[izq] = valor;
12: end function
```

c) AlgoritmoQueHaceAlgo:

```
1: function AlgoritmoQueHaceAlgo(arreglo A)
2:     int i := 1; int j := 1;
3:     int suma := 1; int count := 0;
4:     while i ≤ tam(A) do
5:         if i ≠ A[i] then
6:             count := count+1;
7:         end if
8:         j := 1;
9:         while j ≤ count do
10:            int k := 1;
11:            while k ≤ tam(A) do
12:                suma := suma + A[k];
13:                k := k * 2;
14:            end while
15:            j := j+1;
16:        end while
17:        i := i+1;
18:    end while
19:    return suma
20: end function
```

Ejercicio 7. Pensar un algoritmo que resuelva cada uno de los siguientes problemas y, en cada caso, determinar su complejidad temporal. No es necesario escribir formalmente el algoritmo, basta con delinear los pasos importantes que permitan estimar su complejidad.

a) Calcular la media de un arreglo de enteros.

b) Calcular la mediana[1] de un arreglo de una cantidad impar de enteros.

c) Determinar, dado un n natural, si n es (o no) primo.

¿Le parece que en alguno de los casos anteriores la complejidad del algoritmo propuesto es *óptima* (en sentido asintótico, ignorando constantes)? Sacarse la duda consultando.

Ejercicio 8. Para cada una de las siguientes afirmaciones, decida si son verdaderas o falsas y justifique su decisión.

a) $O(n^2) \cap \Omega(n) = \Theta(n^2)$

b) $\Theta(n) \cup \Theta(n \log n) = \Omega(n \log n) \cap O(n)$

c) Existe una $f : \mathbb{N} \to \mathbb{N}$ tal que para toda $g : \mathbb{N} \to \mathbb{N}$ se cumple que $g \in O(f)$.

d) Sean $f, g : \mathbb{N} \to \mathbb{N}$, entonces se cumple que $O(f) \subseteq O(g)$ o $O(g) \subseteq O(f)$ (es decir, el orden sobre funciones dado por la inclusión de la O es total).

Ejercicio 9. Para cada una de las siguientes afirmaciones, decida si son verdaderas o falsas y justifique su decisión.

a) $n + m = O(nm)$.

b) $n^2 + m^2 = O(nm)$.

c) $n + m^5 = O(m^5)$.

d) $n \log n + m \log m = O(n \log m + m \log n)$.

e) $nm = O(n + m)$.

f) $nm = O(n^2 + m^2)$.

g) $m^5 = O(n + m^5)$.

h) $n \log m + m \log n = O(n \log n + m \log m)$.

Ejercicio 10. Sean $f, g : \mathbb{N} \to \mathbb{N}$, y supongamos que está definido el límite $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \ell \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$. Probar que:

a) $0 < \ell < +\infty$ si y sólo si $f \in \Theta(g)$.
b) $\ell = +\infty$ si y sólo si $f \in \Omega(g)$ y $f \notin O(g)$.
c) $\ell = 0$ si y sólo si $f \in O(g)$ y $f \notin \Omega(g)$.

Recordar las definiciones de límite:

- $\lim_{n\to+\infty} a_n = \ell \in \mathbb{R}$ si $\forall\, \varepsilon > 0.\, \exists n_0 \in \mathbb{N}.\, \forall n > n_0.\, |a_n - \ell| < \varepsilon$.
- $\lim_{n\to+\infty} a_n = +\infty$ si $\forall\, M > 0.\, \exists n_0 \in \mathbb{N}.\, \forall n > n_0.\, a_n > M$.

[1] La *mediana* es el valor que deja a cada lado (por encima y por debajo) la mitad de los valores de la muestra.