Práctica 1 - Complejidad algorítmica

Varios de los ejercicios aquí incluidos son idea de Francisco Soulignac.

NOTAS PRELIMINARES
Los objetivos de esta práctica son:

- Introducir la noción de complejidad algorítmica.
- Desarrollar intuiciones sobre las clases de complejidades más frecuentes.

Ejercicio 1

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

a f, h **son funciones tales que** $f(n) = n^2 - 4n - 2$ **y** $h(n) = n^2$

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

c f, g, h **son funciones tales que** $g(n) = \log n$**,** $h(n) = n$ **y** $f \in O(g)$

Para evitar tener que definir las funciones a la hora de escribir los conjuntos O, Θ y Ω, vamos a utilizar una notación más cómoda y que se utiliza usualmente en la literatura. Para cualquier par de funciones f y g, vamos a decir que $f(n) = O(g(n))$ si y slo si $f \in O(g)$ Análogamente, podemos decir que $f \in O(g(n))$, o que $f(n) = O(g)$ De esta forma, el ejercicio anterior podra reescribirse de la siguiente forma:

Probar utilizando las definiciones que:

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

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

c Si f **es tal que** $f \in O(\log n)$**, entonces** $f \in O(n)$

Esta notación se extiende a funciones con más de un parámetro, y a los conjuntos Θ y Ω. Sin embargo, a pesar de ser una notación cómoda, tiene ciertas inconvenientes de los que hay que cuidarse. En primer lugar, la relación $=$ que utilizamos para decir que $f(n) = O(g(n))$ **NO** es una relación de equivalencia. De hecho, el lado izquierdo del $=$ representa una función, mientras que el lado derecho representa una familia de funciones (y el $=$ significa pertenencia). En segundo lugar, qué significa cuando escribimos $f \in O(n^k)$ para alguna función f? Podra significar que f crece: como un polinomio en n, como una función exponencial en k, o como una función de dos parametros n y k Para evitar esta ambiguedad, debemos explicitar aquellos parámetros que sean constantes. Luego, si queremos decir que f crece como un polinomio en n, debemos decir que f es una función tal que $f \in O(n^k)$ donde k **es una constante**. Por otra parte, qué significa cuando escribimos $2^k = O(1)$? Podrá significar que la función $k \to 2^k$ crece como una constante, o que 2^k es una constante. En estos casos, también hay que aclarar si k **es o no una constante**.

Ejercicio 2

Utilizando la nueva notación, determinar la verdad o falsedad de cada una de las siguientes afirmaciones. **JUSTIFICAR**.

a. $2^n = O(1)$
b. $n! = O(2^n)$
c. $2^n = O(n!)$
d. $2^n n^2 = O(3^n)$
e. Para todo $i, j \in \mathbb{N}$, $i \cdot n = O(j \cdot n)$
f. Para toda función $f : \mathbb{N} \to \mathbb{N}$, $f = O(f)$
g. Para todo $k \in \mathbb{N}$, $2^k = O(1)$

Ejercicio 3

a. Qué significa, intuitivamente, $O(f) \subseteq O(g)$? Qué se puede concluir acerca del crecimiento de f y g cuando, simultneamente, tenemos $O(f) \subseteq O(g)$ y $O(g) \subseteq O(f)$?
b. Cómo ordena por inclusión las siguientes familias de funciones?

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

Ejercicio 4

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

```
# Calcula la sumatoria de un arreglo de enteros.
1: def Sumatoria(arreglo A):
2:     int i, total;
3:     total := 0;
4:     for i := 0 ... Long(A) - 1:
5:         total := total + A[i];
6:     end
7: end
```

b. SumatoriaLenta

```
# Calcula la sumatoria de $n$, definida como la suma de todos los enteros.
# entre $1$ y $n$, de forma poco eficiente.
1: def SumatoriaLenta(natural N)
2:     int i, total;
3:     total := 0;
4:     for i := 1 ... n:
5:         for j := 1 ... i:
6:             total := total + 1;
7:         end
8:     end
9: end
```

c. BúsquedaBinaria

```
# Determina si un elemento se encuentra en un arreglo, que debe estar ordenado.
1:  def BúsquedaBinaria(arreglo A, elem valor):
2:      int izq := 0, der := Long(A) - 1;
3:      while izq $<$ der:
4:          int medio := (izq + der) / 2;
5:          if (valor $<$ A[medio]):
6:              der := medio;
7:          else
8:              izq := medio;
9:          end
10:     end
11:     return A[izq] $=$ valor;
12: end
```

Ejercicio 5

a. 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.

- Calcular la media (o promedio) de un arreglo de enteros.
- Calcular la mediana[1] de un arreglo de una cantidad impar de enteros.
- Determinar, dado un n natural, si n es (o no) primo.

b. 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.

[1] *La **mediana** es el valor que deja a cada lado (por encima y por debajo) la mitad de los valores de una muestra. Es decir, si los elementos de la muestra estuvieran ordenados, sera el elemento del medio.*

Ejercicio 6

Sean $f(n)$ y $g(n)$ dos funciones no-negativas. Usando la definición de Θ, demostrar que $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ Notar que eso implica que si dos bloques de codigo se ejecutan uno tras otro, el costo de la secuencia será la del máximo de ellos.

Ejercicio 7

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

n y m son variables que representan el tiempo de ejecucion de distintas funciones.

a. $n + m = O(nm)$
b. $nm = O(n + m)$
c. $n^2 + m^2 = O(nm)$
d. $nm = O(n^2 + m^2)$
e. $n + m^5 = O(m^5)$
f. $m^5 = O(n + m^5)$
g. $n \log n + m \log m = O(n \log m + m \log n)$
h. $n \log m + m \log n = O(n \log n + m \log m)$