Guía Práctica 4 – Complejidad

1. Mostrar usando la definición de $\mathcal{O}()$ tomada de [1] [2] :

$$\mathcal{O}(f(n)) = \{\ t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 \Rightarrow t(n) \leq cf(n)\ \}.$$

   a) $50n - 8 \in \mathcal{O}(n)$

   b) $5n^2 - 3n + 6 \in \mathcal{O}(n^2)$

   c) $2^n + 7n + 3 \in \mathcal{O}(2^n)$

2. Determinar $\mathcal{O}()$ y $\Omega()$ de las siguientes funciones:

   a) $5n^2$

   b) $3n^2 + 2n$

   c) $3^n + 2^n$

   d) $n + \log_2(n)$

   e) $n^2 + \log_2(n)$

3. Determinar $\Theta()$ de las siguientes funciones:

   a) $n^2 + 2n$

   b) $t(n) = \begin{cases} 2n + 4 & \text{si n par} \\ 7n^2 + n & \text{si n impar} \end{cases}$

   c) $t(n) = \begin{cases} 4n + 5 & \text{si n par} \\ 5n + n & \text{si n impar} \end{cases}$

4. Reescribir los siguientes órdenes de complejidad de menor a mayor ($<$):
   $\mathcal{O}(n), \mathcal{O}(n \log n), \mathcal{O}(n!), \mathcal{O}(n^2), \mathcal{O}(5), \mathcal{O}(\log \log n), \mathcal{O}(3^n), \mathcal{O}(\sqrt{n}), \mathcal{O}(n^n),$
   $\mathcal{O}(n^3), \mathcal{O}(2^n), \mathcal{O}(\log n)$.

   Etiquete cada $\mathcal{O}$ como **sublineal**, **lineal** o **superlineal**. Dentro de éstas últimas puede decir cuáles son **polinómicas** (cuadráticas, cúbicas, etc.) o **exponenciales**.

5. Para cada uno de los fragmentos de código siguientes:
   - Diga qué $\mathcal{O}()$ poseen.
   - Impleméntelos y dé el tiempo de ejecución para $\neq$ valores de n.
   - Compare estimación teórica (primer ítem) con tiempos reales (segundo ítem).

[1] Marzal y Gracia. Introducción al análisis de algoritmos.
[2] https://arco.esi.uclm.es/public/mirror/Introducci%C3%B3n%20al%20an%C3%A1lisis%20de%20algoritmos.pdf

Algorithm 1:

```
sum ← 0
for i ← 1 to n do
    sum ← sum + 1
```

Algorithm 2:

```
sum ← 0
for i ← 1 to n do
    for j ← 1 to n do
        sum ← sum + 1
```

Algorithm 3:

```
sum ← 0
for i ← 1 to n do
    for j ← 1 to n^2 do
        sum ← sum + 1
```

Algorithm 4:

```
sum ← 0
for i ← 1 to n do
    for j ← 1 to i do
        sum ← sum + 1
```

Algorithm 5:

```
sum ← 0
for i ← 1 to n do
    for j ← 1 to i^2 do
        for k ← 1 to j do
            sum ← sum + 1
```

Algorithm 6:

```
i ← 1
while i ≤ n do
    j ← 1
    while j ≤ n do
        j ← j * 2
    i ← i + 1
```

Algorithm 7:

```
i ← 1
while i ≤ n do
    j ← 1
    while j ≤ i do
        j ← j + 1
    i ← i + 1
```

Algorithm 8:

```
i ← 1
while i ≤ 10 do
    j ← 1
    while j ≤ 10 do
        j ← j + 1
    i ← i + 2
```

6. ¿Cuál es el propósito de los siguientes algoritmos? Analizar el tiempo de ejecución en el peor caso y expresarlos en notación $\mathcal{O}$.

Algorithm 9:

Input: a, n: **integer**
Output: b
b: **long integer**
$b \leftarrow 1$
for $k \leftarrow 0$ ***to*** $k < n$ **do**
    $b \leftarrow b * a$

Algorithm 10:

Input: a, n: **integers**
Output: b
b: **long integer**
c: **long integer**
$b \leftarrow 1$
$c \leftarrow a$
for $k \leftarrow n$ ***to*** $k > 0$ **do**
    if k ***mod*** $2 == 0$ **then**
        $k \leftarrow k/2$
        $c \leftarrow c * c$
    else
        $k \leftarrow k - 1$
        $b \leftarrow b * c$

7. A partir de una secuencia de números A, calcular otra B tal que cada uno de sus elementos B_i sea el promedio de todos los anteriores en la secuencia original, incluído el $i - ésimo$.
Ejemplo:
$A = [1, 3, 5, 4, 2, 1]$
$B = [1, 2, 3, 3{,}25, 3, 2{,}66]$

8. Un arreglo A contiene $n - 1$ enteros **únicos** $\in [0, n - 1]$; es decir, hay un número de este intervalo que no está en A. Escriba un algoritmo para encontrar ese número faltante en $\mathcal{O}(n)$. Sólo es posible utilizar $\mathcal{O}(1)$ espacio adicional además del arreglo A mismo.
Ejemplo:
$N = 5$. El arreglo puede contener elementos (no repetidos) $\in [0, 4]$.
Si el vector $= [1, 4, 2, 0]$, entonces el elemento que falta es **3**.

9. Suponga que cada renglón de un arreglo $A_{n,n}$ posee valores 0 y 1 de tal manera que en cualquier renglón de A todos los 1's van antes que los 0's. Describa un método de $\mathcal{O}(n)$ para determinar el renglón de A que contenga la mayor cantidad de 1's.

Ejemplo:

1	1	0	0	0
1	1	1	0	0
1	**1**	**1**	**1**	**0**
1	0	0	0	0
1	1	1	0	0

10. Para cada solución que tenga de los problemas de las guías anteriores, comente, si es posible, el $\mathcal{O}()$ de su solución. Si no es posible (por ejemplo talla indeterminada del problema, o caminos de ejecución no determinísticos) explique por qué. Para los problemas que no tenga resuelto, trate de proponer un algoritmo (al menos en forma coloquial) y vea si puede determinar el $\mathcal{O}()$ de su propuesta.

11. Realice algoritmos (no es necesario implementarlos, suficiente con pseudocódigo claro) para:

    a) **Sumar** $N \in \mathbb{N}$ y $M \in \mathbb{N}$ tales que n y m representan la cantidad de dígitos cada uno. Piense que el número de dígitos puede ser muy grande.

    b) **Multiplicar** $N \in \mathbb{N}$ y $M \in \mathbb{N}$ tales que n y m representan la cantidad de dígitos cada uno. Piense que el número de dígitos puede ser muy grande.

    c) **Multiplicar** las **matrices** $A_{m,n}$ y $B_{n,q}$.

    NOTA: En todos los casos exprese el $\mathcal{O}()$ de su algoritmo.

12. Resolver en lenguage C++ y obtener **ACCEPTED** en los jueces correspondientes para los siguientes problemas:

    a) Integer Inquiry – http://uva.onlinejudge.org/external/4/424.html

    b) 500! – http://uva.onlinejudge.org/external/6/623.html

    c) Overflow – http://uva.onlinejudge.org/external/4/465.html

    NOTA: En todos los casos exprese el $\mathcal{O}()$ de su algoritmo.