UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 02: Complejidad de Algoritmos

ALUMNOS:

Castañon Maldonado Carlos Emilio
Nepomuceno Escarcega Arizdelcy Lizbeth

PROFESOR

María de Luz Gasca Soto

AYUDANTES

Brenda Margarita Becerra Ruíz
Enrique Ehecatl Hernández Ferreiro (Link)

ASIGNATURA

Análisis de Algoritmos

1 Sea Π un problema. El desempeño computacional en el peor de los casos para Π es $O(n^2)$ y también es $\Omega(n \log_2 n)$.
Sea $\mathcal{A}$ un algoritmo que soluciona Π. ¿Cuáles de las siguientes afirmaciones resultan consistentes con la información sobre Π?

Justifique su respuesta.

a) $\mathcal{A}$ tiene en el peor caso complejidad $O(n^3)$.
Sea $f(n)$ la función de complejidad del algoritmo $\mathcal{A}$. Como $f(n)$ es $O(n^2)$, implica que existen dos constantes c, k tal que $c \in \mathbb{R}$ y $k \in \mathbb{N}$, tal que $f(n) \leq cn^2$ para todo $n \geq k$, como $cn^2 \leq cn^3$ para todo $n \geq k$ entonces, tenemos que: $f(n) \leq cn^2 \leq cn^3$ para todo $n \geq k$, por lo que, $f(n) \leq cn^3$ para todo $n \geq k$. Por lo tanto $f(n)$ es $O(n^3)$.

b) $\mathcal{A}$ tiene en el peor caso complejidad $O(n)$.
Por el inciso anterior, sabemos que $f(n) \leq cn^2$, además también sabemos que existen dos constantes c', k' tal que para todo $n \geq k$ $f(n) \geq c'nlog(n)$, entonces tenemos que $c'nlog(n)n \leq f(n) \leq c(n^2)$. Por lo tanto $f(n)$ no puede ser $O(n)$.

c) $\mathcal{A}$ tiene en el peor caso complejidad $\Theta(n \log n)$.
Para que $f(n)$ sea $\Theta(nlog(n))$ tiene que pasar que $f(n)$ sea $\Omega(nlog(n))$ y $f(n)$ se $O(nlog(n))$, pero $f(n)$ no necesariamente está en $O(nlog(n))$, por que $f(n)$ representa la complejidad en el peor de los casos y esta es $O(n^2)$.

d) $\mathcal{A}$ tiene en el peor caso complejidad $\Theta(n^2)$.
Para que $f(n)$ sea $\Theta(n^2)$ tiene que pasar que $f(n)$ sea $O(n^2)$ y $f(n)$ tiene que ser $\Omega(n^2)$, pero $f(n)$ no necesariamente va a ser $\Omega(n^2)$, porque al estár $f(n)$ en $\Omega(nlog(n))$, quiere decir que $f(n)$ toma tiempo al menos $nlog(n)$, por lo tanto $f(n)$ no es $\Theta(n^2)$.

2 Supongamos que un algoritmo $\mathcal{A}$ se ejecuta en el peor de los casos con tiempo $f(n)$ y el algoritmo $\mathcal{B}$ toma tiempo $g(n)$, en el peor caso.
Responda las siguientes preguntas con **sí, no** o **tal vez** y **justifica formalmente tu respuesta**.
¿Es $\mathcal{B}$ más rápido que $\mathcal{A}$, para toda n mayor que alguna n_o **a)** ... si $g(n) \in \Omega(f(n) \log n)$?

Nota: podemos afirmar que $cf(n) < cf(n)log(n)$ para todo $c \in \mathbb{R}^+$ y $n > 1$.

Si $g(n)$ es $\Omega(f(n)log(n))$ entonces significar que existe algún $c \in \mathbb{R}^+$ tal que $g(n) \geq cf(n)log(n)$ para todo $n \geq k$ para algún $k \in \mathbb{N}$, entonces $g(n) > cf(n)$, por lo que el algoritmo $\mathcal{A}$ es más rápido.

b) ... si $g(n) \in \Theta(f(n) \log n)$?
Como $c_1, c_2 \in \mathbb{R}^+$ tal que $c_1 f(n)log(n) \leq g(n) \leq c_2 f(n)log(n)$ para todo $n \geq k$ para algún $k \in \mathbb{N}$, entonces $g(n) > c_1 f(n)$. Por lo tanto el algoritmo $\mathcal{A}$ es más rápido.

3 Considera los siguientes ciclos anidados:

```
...
    i <-- n;
    while i > 0 do
        j <-- i;
            while not (j > n)
                <cuerpo del repeat> // requiere O(1)
                j <-- j * 2
            end_w
        i <-- i / 2
    end_w
...
```

a) Determina el desempeño computacional $T(n)$ de los ciclos anidados.

Primero consideremos que el peor de los casos para el while interno es cuando $j = 1$ y, considerando que $n = 2^x$, tenemos que este ciclo se ejecuta x veces para que $j > n$, porque en cada iteración multiplicamos a j por 2, por lo que $x = log_2(n)$(donde x representa las iteraciones del ciclo interior).
Ahora consideremos el primer while, este se ejecuta cuando $i > 0$, notemos que si $n > 0$ este ciclo nunca va a acabar, ya que si $n = 2^x$ podemos dividir esto entre 2 hasta llegar a $i = 1$, pero notemos que si seguimos dividiendo, nunca vamos a llegar a 0 por lo que este ciclo se ejecutará infinitamente, por lo tanto el desempeño computacional es ∞.

b) Si en el código anterior cambiamos la asignación $i \leftarrow i/2$ por $i \leftarrow i$ div 2, ¿Cuál sería el desempeño computacional $T(n)$ del proceso? **Justifica** Para facilitar las operaciones aritméticas, en ambos incisos, puedes suponer que n es potencia de 2.

Como el ciclo interno no fue modificado este mantiene el desempeño computacional anterior o sea en el peor de los casos realiza $log_2(n)$ operaciones. Ahora siguiendo el razonamiento del inciso anterior el desempeño computacional del primer ciclo es $log_2(n)$, porque a diferencia del insiso anterior este sí acaba ya que al llegar a $i = 1$ el resultado de realizar la operación $i\ div\ 2$ es 0, por lo que el ciclo terminaría, como $n = 2^x$ este ciclo se ejecuta x veces donde $x = log_2(n)$, entonces, el desempeño computacional es

$$T(n) = \sum_{i=1}^{log_2(n)} (log_2(n) - i) = (log_2(n) * (log_2(n) + 1))/2$$

Lo cual tiende a $(log_2(n))^2$.

4 Proporciona un algoritmo (código) cuyo desempeño computacional sea $\Theta(n^3 \log n)$. Debes usar operaciones básicas, **no** debes usar procesos. **Justifica** formalmente que tu algoritmo alcanza el tiempo pedido.

```
def algoritmo_De_Majora(arr):
    n = len(arr)
    result = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                for _ in range(int(math.log(n, 2))):
                    result += arr[i] * arr[j] * arr[k]
    return result
```

Empezando desde del bucle mas profundo, el cual es for _ in range(int(math.log(n, 2))):, podemos notar que este por definicion tiene complejidad logarítmica $O(\log n)$ debido al logaritmo base 2 con el que vamos a operar a n (además de que esto también sucede por que es la operación aritmética de mayor jerarquía en este bucle), el siguiente bucle, el cual es for k in range(n):, podemos observar que tiene una complejidad de $O(n)$ porque itera desde 0 hasta $n - 1$, hasta este momento la complejidad es de $O(n) * O(\log n)$, siguiendo con el siguiente bucle el cual es for j in range(n):, podemos darnos cuenta que este también cuenta con una complejidad de $O(n)$, ya que se comporta igual al bucle anterior a el, con esto tendremos hasta el momento una complejidad de $O(n) * O(n) * O(\log n)$, con esto pasaremos a nuestro ultimo bucle el cual es for i in range(n):, a lo que podemos notar que nuevamente este tendrá una complejidad de $O(n)$, quedándonos por definicion que nuestro código tiene complejidad de $O(n) * O(n) * O(n) * O(\log n)$, lo cual se traduce en $O(n^3 \log n)$.

Ahora, utilizando la definicion de O y de Ω, si tenemos una $f(n)$, $g(n)$ y $O(g(n))$ en las que si $f(n) \in O(g(n))$, entonces $g(n)$ es una cota superior para $f(n)$, esto implica que $f(n)$ no crece mas rápido que $g(n)$, por ende si $f(n)$ está acotada superiormente por $O(n^3 \log n)$ también está acotada superiormente por $O(n^3 \log n)$, por lo tanto $f(n) \in \Omega(n^3 \log n)$.

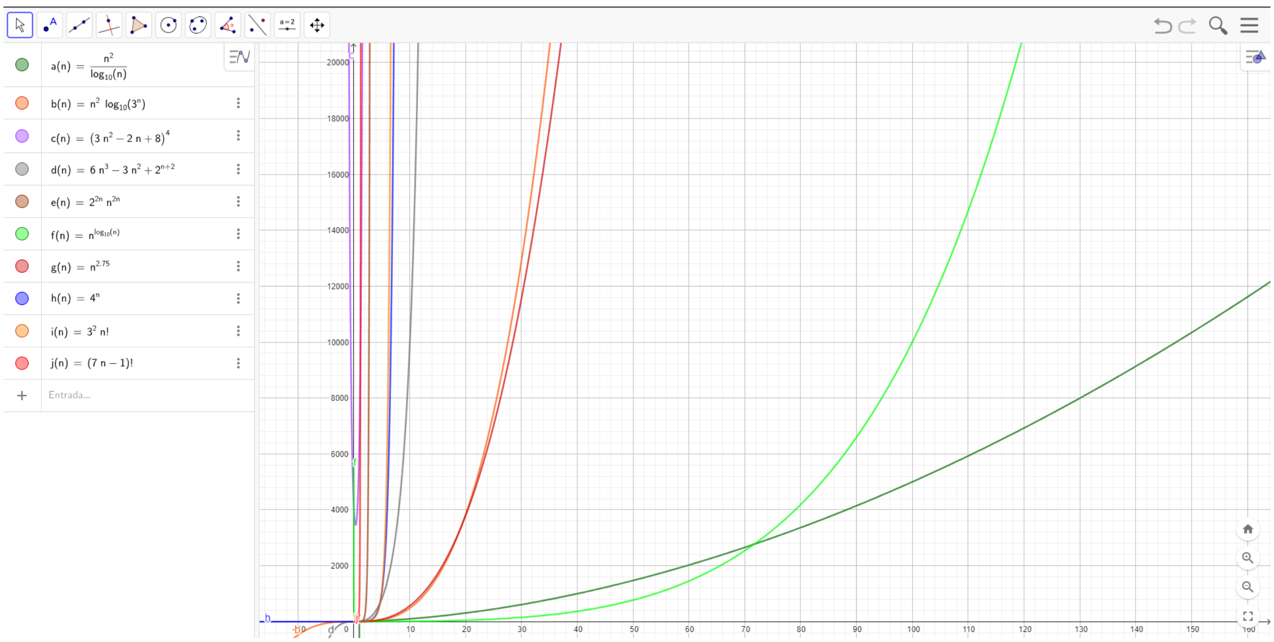
Y recordando que para que algo pertenezca a la complejidad de Θ, este debe también pertenecer a O y a Ω, podemos decir con toda certeza que el algoritmo tiene complejidad $\Theta(n^3 \log n)$.

5 Considera las siguientes funciones de complejidad:

$$\begin{array}{ccccc} n^2log3^n & n^{2.75} & 3^2 * n! & n^2/logn & (3n^2-2n+8)^4 \\ (7n-1)! & 4^n & 2^{2n} * n^{2n} & n^{logn} & 6n^3-3n^2+2^{n+2} \end{array}$$

Usando la definición formal de O, Ω, Θ, o y ω así como las relaciones *estar contenido*, $\subset$, y *ser igual*, $=$, **ordenar** las funciones de complejidad dadas en términos de O, Θ, y Ω.

a) $n^2/\log n$ **b)** $n^{\log n}$ **c)** $n^{2.75}$ **d)** $n^2 \log 3^n$ **e)** 4^n
f) $6n^3-3n^2+2^{n+2}$ **g)** $3^2 * n!$ **h)** $2^{2n} * n^{2n}$ **i)** $(3n^2-2n+8)^4$ **j)** $(7n-1)!$



6 **Usando la definición** de O y Ω, para los siguientes incisos, demuestra formalmente, si $g(n) \in O(f(n))$ o si $g(n) \in \Omega(f(n))$.

	$g(n)$	$f(n)$
a)	2^n	$5^{\log n}$
b)	$n^2/\log n$	$n(\log n)^2$
c)	$\log^3(n)$	$n^{0.5}$
d)	$n!$	2^n
e)	3^n	2^n
f)	$\log 3^n$	$\log 2^n$

a) $g(n) = 2^n$ $\quad f(n) = 5^{\log n}$
Antes de comenzar notemos que por las propiedades de los logaritmos tenemos que: $f(n) = 5^{\log n} = n^{\log 5}$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $2n$ esté acotada superiormente por $c * n^{log5}$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$2n \leq 1 * n^{log5}$
A lo que podemos observar que es verdadero, ya que $2n$ es siempre menor o igual a n^{log5} cuando $n \geq 1$.
$\therefore g(n) = 2n \in O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $2n$ esté acotada inferiormente por $c * n^{log5}$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$2n \geq 1 * n^{log5}$
A lo que podemos observar que es verdadero, ya que $2n$ es siempre mayor o igual a n^{log5} cuando $n \geq 1$.
$\therefore g(n) = 2n \in \Omega(f(n))$

b) $g(n) = n^2/\log n \qquad f(n) = n(\log n)^2$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $n^2/\log n$ esté acotada superiormente por $c * n(\log n)^2$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 2$
$n^2/\log n \leq 1 * n(\log n)^2$
A lo que podemos observar que es verdadero, ya que $n^2/\log n$ es siempre menor o igual a $n(\log n)^2$ cuando $n \geq 2$.
$\therefore g(n) = n^2/\log n \in O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $n^2/\log n$ esté acotada inferiormente por $c * n(\log n)^2$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = \frac{1}{2}$ y $n_0 = 2$
$n^2/\log n \geq \frac{1}{2} * n(\log n)^2$
A lo que podemos observar que es verdadero, ya que $n^2/\log n$ es siempre mayor o igual a $\frac{1}{2} * n(\log n)^2$ cuando $n \geq 2$.
$\therefore g(n) = n^2/\log n \in \Omega(f(n))$

c) $g(n) = \log^3(n) \qquad f(n) = n^{0.5}$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $\log^3(n)$ esté acotada superiormente por $c * n^{0.5}$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$\log^3(n) \leq 1 * n^{0.5}$
A lo que podemos observar que es verdadero, ya que el crecimiento de $\log^3(n)$ es menor que el de $n^{0.5}$ cuando $n \geq 1$.
$\therefore g(n) = \log^3(n) \in O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $\log^3(n)$ esté acotada inferiormente por $c*n^{0.5}$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$\log^3(n) \geq 1 * n^{0.5}$
A lo que podemos observar que es verdadero, ya que $\log^3(n)$ es siempre mayor o igual a $n^{0.5}$ cuando $n \geq 2$.
$\therefore g(n) = \log^3(n) \in \Omega(f(n))$

d) $g(n) = n! \qquad f(n) = 2^n$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $n!$ esté acotada superiormente por $c*2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 4$
$n! \leq 1 * 2^n$
A lo que podemos observar que es verdadero, ya que el crecimiento de $n!$ es mas lento que el de 2^n cuando $n \geq 4$.
$\therefore g(n) = n! \in O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $n!$ esté acotada inferiormente por $c * 2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 4$
$n! \geq 1 * 2^n$
A lo que podemos observar que es verdadero, ya que $n!$ es siempre mayor o igual a 2^n cuando $n \geq 4$.
$\therefore g(n) = n! \in \Omega(f(n))$

e) $g(n) = 3^n \qquad f(n) = 2^n$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que 3^n esté acotada superiormente por $c*2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$3^n \leq 1 * 2^n$
A lo que podemos observar que es falso, y que por ende la cota superior que buscábamos no existe.
$\therefore g(n) = 3^n \notin O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que 3^n esté acotada inferiormente por $c * 2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$3^n \geq 1 * 2^n$
A lo que podemos observar que es verdadero, ya que 3^n es siempre mayor a 2^n.
$\therefore g(n) = 3^n \in \Omega(f(n))$

f) $g(n) = \log 3^n \qquad f(n) = \log 2^n$

$g(n) \in O(f(n))$
Si $g(n) \in O(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $\log 3^n$ esté acotada superiormente por $c * 2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = 1$ y $n_0 = 1$
$\log 3^n \leq 1 * 2^n$
Simplificamos
$n \log 3 \leq 2^n$
A lo que podemos observar que es verdadero, ya que el crecimiento de $n \log 3$ es mas lento que el crecimiento de 2^n.
$\therefore g(n) = \log 3^n \in O(f(n))$

$g(n) \in \Omega(f(n))$
Si $g(n)\Omega(f(n))$, entonces debe haber una constante positiva c y un valor de n_0 tal que $\log 3^n$ esté acotada inferiormente por $c * 2^n$ para todo n mayor o igual a n_0.
Entonces si tenemos $c = \frac{1}{log3}$ y $n_0 = 1$
$\log 3^n \geq \frac{1}{log3} * 2^n$
Simplificamos
$n \log 3 \geq \frac{1}{log3} * 2^n$
A lo que podemos observar que es verdadero, ya que el crecimiento de $n \log 3$ es mayor que el crecimiento de $\frac{1}{log3} * 2^n$
$\therefore g(n) = \log 3^n \in \Omega(f(n))$