Universidad Nacional Autónoma de México

Facultad de Ciencias

Departamento de Matemáticas

México, Septiembre 2020.

Introducción al Análisis de Algoritmos

Notas de Clase

(Primera Parte, Segunda Versión)

Dra. María De Luz Gasca Soto

Licenciatura en Ciencias de la Computación,
Departamento de Matemáticas,
Facultad de Ciencias, UNAM.

Prefacio

Estas notas forman parte del material que se revisa en el curso de Análisis de Algoritmos I que se imparte en la Facultad de Ciencias de la UNAM, para la Licenciatura en Ciencias de la Computación.

He tenido la oportunidad de impartir este curso los últimos años y he estado preparando y corrigiendo las presentes notas de clase para el curso, de las cuales esta resulta ser la segunda versión.

Considero que las áreas de Análisis, Diseño y Justificación de algoritmos debe ser tomado más en serio por las personas que de una u otra manera diseñan programas o bien están involucradas con la computación.

El presente trabajo, pretende dar una panorámica general de lo que es el análisis de algoritmos. Enfatizando que el análisis, diseño y justificación de algoritmos se puede realizar de manera formal usando como herramienta a la Inducción Matemática.

La bibliografía básica, para el material presentado en estas notas, está basada en los libros:

- **Udi Manber** [13]
  Introduction to Algorithms. A Creative Approach.
- **J. Kingston** [12]
  Algorithms and Data Structures: Design, Correctness and Analysis.
- **R. Neapolitan & K. Naimipour** [16].
  Foundations of Algorithms.

Dra. María De Luz Gasca Soto
Profesor Asociado, T.C.
Departamento de Matemáticas
Facultad de Ciencias, UNAM.

Índice general

1. Conceptos Básicos **1**
1.1. Problemas y Algoritmos. 1
1.2. Características de los Algoritmos. 6
1.3. Tipos de Problemas. 7

2. Análisis de Algoritmos **9**
2.1. Introducción. 9
2.2. Complejidad. 10
2.3. Cálculo del Tiempo de Ejecución. 12
2.3.1. Tiempo Constante. 13
2.3.2. Ciclos Simples. 13
2.3.3. Ciclos Anidados. 15
2.3.4. Otros Ciclos. 16
2.3.5. Llamadas a Procesos. 17
2.4. Introducción al Orden 18
2.4.1. Intuitiva Introducción al Orden. 18
2.4.2. Rigurosa Introducción al Orden. 21
2.4.3. Propiedades del Orden. 26
2.5. Ejercicios. 28

3. Inducción Matemática **33**
3.1. Introducción. 33
3.2. Principio de Inducción. 34
3.3. Ejemplos de Inducción Matemática. 35
3.3.1. Ejemplos de Álgebra 35
3.3.2. Ejemplos de Geometría Computacional 37
3.3.3. Ejemplos de Teoría de Gráficas 39
3.3.4. Otros Ejemplos 45
3.4. Ejercicios 48

4. Justificación de Algoritmos **51**
4.1. Algoritmos Recursivos. 52
4.1.1. Números de Fibonacci. 53
4.1.2. Factorial de n 53

2.4. Introducción al Orden

A continuación formalizaremos los conceptos relacionados con el orden de las funciones de complejidad. Se inicia dando una versión intuitiva y concluimos dando una versión muy formal. Este material está basado en el Capítulo 1 del libro *Foundations of Algorithms* de R. Neapolitan y K. Naimipour[16].

2.4.1. Intuitiva Introducción al Orden.

En esta sección presentaremos de manera breve e intuitiva el concepto de orden, proporcionamos una serie de ejercicios resueltos para alcanzar este objetivo.

Algoritmos con desempeño computacional n o $100n$ son llamados *algoritmos de tiempo lineal* ya que su complejidad es una función lineal en el tamaño de la entrada. Algoritmos con desempeño computacional n^2 o $0{,}01n^2$ o $5n^2+3n+10$ son llamados *algoritmos de tiempo cuadrático* pues su complejidad es una función cuadrática en el tamaño de la entrada. De manera similar, algoritmos con desempeño computacional n^3 o $0{,}01n^3$ o $4n^3+5n^2+3n+10$ son llamados *algoritmos de tiempo cúbico* dado que su complejidad es una función cúbica en el tamaño de la entrada. Esto resulta ser un principio fundamental, ya que algoritmos de tiempo lineal son eventualmente más eficientes que cualquier algoritmo de tiempo cuadrático y estos a su vez serán eventualmente mejores que cualquier algoritmo de tiempo cubico.

En el análisis teórico de un algoritmo estamos interesados en el eventual comportamiento del algoritmo. A continuación mostramos cómo los algoritmos pueden ser agrupados de acuerdo a su eventual comportamiento, de esta manera podremos determinar si el comportamiento de un algoritmo es eventualmente mejor que el de otro.

Funciones de la forma $3n^2$ o $3n^2+100$ son llamadas funciones *cuadráticas puras* ya que no poseen término lineal, mientras que las funciones como $0{,}2n^2+0{,}5n+10$ y $0{,}2n^2+5n+10$ son denominadas funciones *cuadráticas completas* pues cuentan con un término lineal. La siguiente tabla muestra cómo, eventualmente, el término cuadrático domina la función.

n	$0{,}2n^2$	$0{,}2n^2+0{,}5n+10$	$0{,}2n^2+2n+10$	$0{,}2n^2+5n+10$
10	20	35	50	80
20	80	100	130	190
50	500	535	610	760
100	2000	2060	2210	2510
200	8000	8110	8410	9010
500	50000	50260	51010	52510
1000	200000	200510	202010	205010
2000	800000	801010	804010	810010
5000	5000000	5002510	5010010	5025010
100000	20000000	20005010	20020010	20050010

Como se puede observar en la tabla, los otros términos serán eventualmente insignificantes comparados con el valor del término cuadrático. Esto significa que si un algoritmo tiene este tiempo de complejidad, podemos decir que es un algoritmo de tiempo cuadrático.

Intuitivamente, parece que deberíamos tirar los términos de orden menor cuando clasificamos las funciones de complejidad. Por ejemplo, la función $4n^3+5n^2+3n+10$ debe ser clasificada sólo como una función cúbica pura. Más adelante estableceremos formalemente que esto puede hacerse, primero lo haremos intuitivamente.

El conjunto de todas las funciones que pueden ser clasificadas con funciones cuadráticas puras es llamado $\Theta(n^2)$. Si una función es miembro del conjunto $\Theta(n^2)$, diremos que la función es de **orden** n^2. Por ejemplo, como podemos tirar los términos de orden menor, se tiene que $g(n) = 5n^2 + 120n + 20 \in \Theta(n^2)$, es decir $g(n)$ es de orden n^2.

Cuando el tiempo de complejidad de un algoritmo está en el conjunto $\Theta(n^2)$ es llamado algoritmo de tiempo cuadrático.

Similarmente, el conjunto de todas las funciones que pueden ser clasificadas con funciones cúbicas puras es llamado $\Theta(n^3)$ y se dice que la función es de **orden** n^3 y se puede generalizar este concepto de acuerdo al máximo término de la función polinomial. A estos conjuntos se les denomina **Categorias de Complejidad**. A continuación listamos algunas de las más comunes:

$$\Theta(\log n) \quad \Theta(n) \quad \Theta(n \log n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n)$$

En este orden, si $f(n)$ está en una categoría a la izquierda de la categoría que contiene a $g(n)$, entonces $f(n)$ eventualmente está debajo de $g(n)$ en una gráfica. La siguiente tabla muestra algunos valores puntuales para estas funciones, mientras que la Figura 2.3 muestra las gráficas de estas funciones clásicas de complejidad.

n	$\log_2 n$	$n \log_2 n$	n^2	n^3	2^n
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	$\sim 42 \times 10^8$
64	6	384	4096	262144	$\sim 18 \times 10^{18}$
128	7	896	16384	2097152	$\sim 34 \times 10^{37}$
256	8	2048	65536	16×10^6	$\sim 11 \times 10^{77}$
512	9	4608	262144	13×10^7	$\sim 13 \times 10^{155}$
1024	10	10240	1048576	10×10^8	-

La siguiente tabla muestra la tasa de crecimiento, en nanosegundos, de las funciones clásicas de complejidad: $\log_2 n,\ n,\ n \log_2 n,\ n^2,\ 2^n,\ n!$ [18].

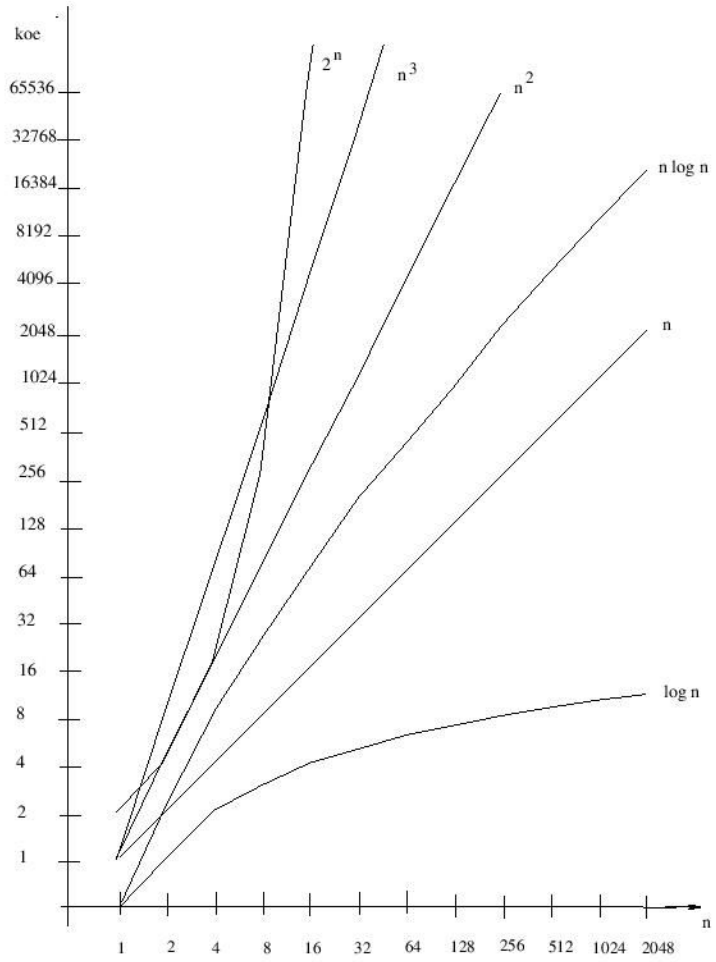

Figura 2.3: Gráficas de las funciones clásicas de complejidad

n	$f(n) = \log_2 n$	$f(n) = n$	$f(n) = n \log_2 n$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = n!$
10	0.003μs	0.01μs	0.033μs	0.1μs	1.0μs	3.63ms
20	0.004μs	0.02μs	0.086μs	0.4μs	1.0ms	77.1años
30	0.005μs	0.03μs	0.147μs	0.9μs	1.0seg	$8{,}4 \times 10^{15}$ años
40	0.005μs	0.04μs	0.213μs	1.6μs	18.3min	
50	0.006μs	0.05μs	0.282μs	2.5μs	13 dias	
10^2	0.007μs	0.10μs	0.644μs	10.0μs	4×10^{13} años	
10^3	0.010μs	1.00μs	9.966μs	1.0ms		
10^4	0.013μs	10.00μs	130.0μs	100.0ms		
10^5	0.017μs	0.10ms	1.67ms	10.0seg		
10^6	0.020μs	1.00ms	19.93ms	16.7min		
10^7	0.023μs	0.01seg	0.23seg	1.16dias		
10^8	0.027μs	0.10seg	2.66seg	115.7dias		
10^9	0.030μs	1.00seg	29.90seg	31.7años		

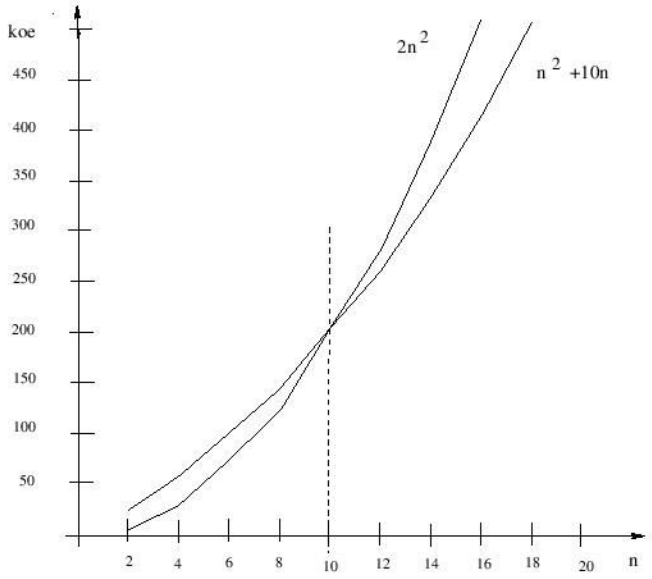

Figura 2.4: Gráficas de las funciones $n^2 + 10n$ y $2n^2$.

2.4.2. Rigurosa Introducción al Orden.

A continuación formalizaremos los conceptos relacionados con el orden de las funciones de complejidad y definimos otros conceptos fundamentales como "O", o-grande, "o", o-pequeña y Ω, omega.

* O(f(n))

Definición 2.1 Dada una función de complejidad $f(n)$, $O(f(n))$ es el conjunto de funciones de complejidad $g(n)$ para las cuales *existe alguna* constante c, real y positiva, y algún entero no negativo N, tal que

$$\forall n, \quad n \geq N, \qquad g(n) \leq c \cdot f(n).$$

Si $g(n) \in O(f(n))$, decimos que $g(n)$ es O *grande* de $f(n)$. Aunque $g(n)$ inicie por encima de $c \cdot f(n)$, eventualmente, a partir de N, será menor que ella.

Ejemplo 2.1 Sea $g(n) = n^2 + 10n$. Por demostrar que $g(n)$ es $O(n^2)$.

Solución Si graficamos las funciones $g(n)$ y $f(n) = 2 \cdot n^2$, para $n \geq 0$ tenemos que a partir de $n \geq 10$, $g(n) = n^2 + 10n \leq 2 \cdot n^2 = f(n)$. Por lo tanto, tomando $c = 2$ y $N = 10$, concluimos que $n^2 + 10n \in O(n^2)$, esto es: $g(n) \in O(f(n))$. La Figura 2.4 muestra las gráficas de estas funciones, se observa como a partir de $N = 10$ la función $f(n) = 2n^2$ acota superiormente a la función $g(n) = n^2 + 10n$.

Esto significa que si $g(n)$ representa el desempeño computacional de algún algoritmo, eventualmente el tiempo de ejecución del algoritmo será al menos cuadrático. Para propósitos de Análisis, podemos decir que $g(n)$ es tan bueno como una función cuadrática.

Se dice que *O grande* describe el comportamiento asintótico de una función, de hecho representa una cota superior de la función.

Ejemplo 2.2 Observe que:

a) $3n^2 + 10n + 7 \in O(n^2)$,
ya que $4n^2 > 3n^2 + 10n + 7$ a partir de $N = 11$

b) $3n^2 + 10n + 7 \in O(n^3)$,
ya que $n^3 > 3n^2 + 10n + 7$ a partir de $N = 6$

c) $3n^2 + 10n + 7 \notin O(n)$,
ya que $c \cdot n < 3n^2 + 10n + 7$ cuando $n > c$.

d) $3n \log n + 5n + 10 \in O(n^2)$
ya que $3n^2 > 3n \log n + 5n + 10$ a partir de $N = 5$.

e) $3n \log n + 5n + 10 \in O(n^3)$
ya que $n^3 > 3n \log n + 5n + 10$ a partir de $N = 2$.

f) $3n \log n + 5n + 10 \in O(n)$
ya que $n < 3n \log n + 5n + 10$ para $n \geq 0$.

g) $5n^2 + 3\sqrt{n} + 123 \in O(n^3)$
ya que $n^3 > 5n^2 + 3\sqrt{n} + 123$ a partir de $N = 8$.

* Ω(f(n))

Definición 2.2 Dada una función de complejidad $f(n)$, $\Omega(f(n))$ es el conjunto de funciones de complejidad $g(n)$ para las cuales *existe alguna* constante c, real y positiva, y algún entero no negativo N, tal que $\forall n,\ n \geq N, g(n) \geq c \cdot f(n)$. Si $g(n) \in \Omega(f(n)$, decimos que $g(n)$ es *Omega* $f(n)$.

Ejemplo 2.3 Sea $g(n) = 5n^2$. Por demostrar que $g(n)$ es $\Omega(n^2)$.

Solución. Para $n \geq 0, 5n^2 \geq 1 \cdot n^2$, así que tomamos $c = 1$ y $N = 0$.

Ejemplo 2.4 Se tiene que:

a) $3n^2 + 10n + 7 \in \Omega(n^2)$,
ya que $3n^2 < 3n^2 + 10n + 7$ a partir de $N = 1$

b) $3n^2 + 10n + 7 \notin \Omega(n^3)$,
ya que $n^3 > 3n^2 + 10n + 7$ a partir de $N = 6$

c) $3n^2 + 10n + 7 \in \Omega(n)$,
ya que $n < 3n^2 + 10n + 7$ cuando $n \geq 0$.

d) $3n \log n + 5n + 10 \notin \Omega(n^2)$
ya que $cn^2 > 3n \log n + 5n + 10$ para $c > 0$.

e) $3n \log n + 5n + 10 \in \Omega(n^3)$
ya que $cn^3 > 3n \log n + 5n + 10$ para $c > 0$.

f) $3n \log n + 5n + 10 \in \Omega(n)$
ya que $n < 3n \log n + 5n + 10$ para $n \geq 0$.

g) $5n^2 + 3\sqrt{n} + 123 \in \Omega(n^2)$
ya que $n^2 < 5n^2 + 3\sqrt{n} + 123$ a partir de $N = 0$.

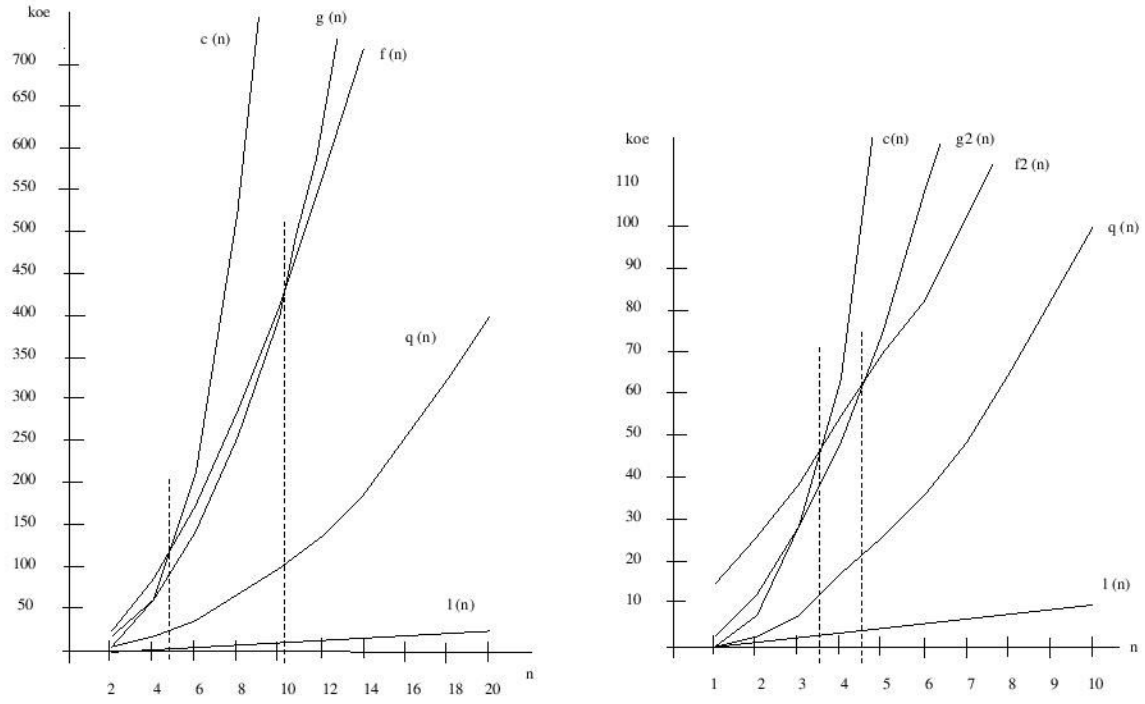

Figura 2.5: Comparación de las funciones de los Ejemplos 2.2, 2.4 y 2.5.

* Θ(f(n))

Definición 2.3 Dada una función de complejidad $f(n)$, $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$.

Es decir el $\Theta(f(n))$ es el conjunto de funciones de complejidad $g(n)$ para las cuales *existen algunas* constantes reales positivas c y d, y algún entero no negativo N, tal que $\forall n \geq N, c \cdot f(n) \leq g(n) \leq d \cdot f(n)$. Si $g(n) \in \Theta(f(n)$, decimos que $g(n)$ es *orden* de $f(n)$. $\Theta(f(n)$ determina una *categoría* de complejidad.

Ejemplo 2.5 Observe que:

a) $3n^2 + 10n + 7 \in \Theta(n^2)$, pues es $O(n^2)$ y $\Omega(n^2)$
b) $3n^2 + 10n + 7 \notin \Theta(n^3)$, aunque es $O(n^3)$ no es $\Omega(n^3)$
c) $3n^2 + 10n + 7 \notin \Omega(n)$, pues sólo es $\Omega(n)$.
d) $3n \log n + 5n + 10 \notin \Theta(n^2)$ ya que no es $\Omega(n^2)$
e) $3n \log n + 5n + 10 \in \Omega(n^3)$, aunque es $O(n^3)$ no es $\Omega(n^3)$
f) $5n^2 + 3\sqrt{n} + 123 \in \Theta(n^3)$ pues es $O(n^3)$ y $\Omega(n^3)$

La Figura 2.5(a), ilustra gráficamente la comparación entre las funciones $g(n) = 4n^2$, $f(n) = 3n^2 + 10n + 7$, $c(n) = n^3$, $q(n) = n^2$ y n. La Figura 2.5(b), ilustra gráficamente la comparación entre las funciones $f_2(n) = 3n \log n + 5n + 10$, $g_2(n) = 3n^2$, $c(n) = n^3$, $q(n) = n^2$ y $l(n) = n$.

Ejemplo 2.6 Demuestre que n no es $\Omega(n^2)$.
Solución. Para demostrar esta afirmación usaremos una *prueba por contradicción*.
Suponemos que $n \in \Omega(n^2)$, es decir, existe alguna constante positiva c y un entero no negativo N tal que para $n \geq N$, $n \geq c \cdot n^2$. Si dividimos ambos lados de la desigualdad por $c \cdot n$, tenemos que para $n \geq N, 1/c \geq n$. Sin embargo, para $n > 1/c$ esta desigualdad nunca se satisface, lo cual significa que no se cumple para $n \geq N$. Esta contradicción prueba que n no es $\Omega(n^2)$.

* $\omega(\mathbf{f(n)})$

Definición 2.4 Dada una función de complejidad $f(n)$, $\omega(f(n))$ es el conjunto de funciones de complejidad $g(n)$ que satisfacen la siguiente condición:

> *Para toda* constante real positiva c, *existe* un entero no negativo N,
> tal que $\forall n,\ n \geq N,\ c \cdot g(n) > f(n)$.

Si $g(n) \in \omega(f(n)$, decimos que $g(n)$ es *omega pequeña* de $f(n)$.

* $o(\mathbf{f(n)})$

Definición 2.5 Dada una función de complejidad $f(n)$, $o(f(n))$ es el conjunto de funciones de complejidad $g(n)$ que satisfacen la siguiente condición:

> *Para toda* constante real positiva c, *existe* un entero no negativo N,
> tal que $\forall n,\ n \geq N, g(n) \leq c \cdot f(n)$.

Si $g(n) \in o(f(n)$, decimos que $g(n)$ es *o pequeña* de $f(n)$.

Ejemplo 2.7 Demuestre que n es $o(n^2)$.
Solución. Sea $c > 0$ un valor dado. Requerimos encontrar una N tal que para $n \geq N$ se tiene que $n \leq c \cdot n^2$. Si dividimos ambos lados de la desigualdad por $c \cdot n$, obtenemos $1/c \leq n$. Por lo tanto, es suficiente elegir cualquier $N \geq 1/c$.

Nótese que el valor de N depende de la constante c. Por ejemplo, si $c = 0{,}00001$, debemos tomar $N \geq 10000$. Es decir, para $n \geq 10000$ se tiene que $n \leq 0{,}00001 \cdot n^2$.

Teorema 2.1 Si $g(n)$ es $o(f(n))$, entonces $g(n)$ es $O(f(n)) \setminus \Omega(f(n))$.
Esto es, $g(n)$ es $O(f(n))$ pero no es $\Omega(f(n))$.
Demostración.

> Ya que $g(n) \in o(f(n))$, para cada constante real positiva existe una N tal que para toda $n \geq N$, $g(n) \leq c \cdot f(n)$, lo cual significa que, en efecto, la cota se satisface para alguna c. Por lo tanto, $g(n) \in O(f(n))$.
> Por demostrar que $g(n) \notin \Omega(f(n))$, lo haremos por contradicción.
> Si $g(n) \in \Omega(f(n))$ entonces existe alguna constante real $c > 0$ y alguna N_1 tal que para toda $n \geq N_1$, $g(n) \geq c \cdot f(n)$. Pero, tenemos que $g(n) \in o(f(n))$, es decir, existe alguna N_2 tal que para toda $n \geq N_2$, $g(n) \leq (c/2) \cdot f(n)$. Ambas desigualdades deberían cumplirse para toda n mayor a N_1 y N_2, esto no es posible. Por lo que esta contradicción prueba que $g(n)$ no está en $\Omega(f(n))$.

Ejemplo 2.8 Considere la función $g(n) = \begin{cases} n & si\ n\ es\ par; \\ 1 & si\ n\ es\ impar. \end{cases}$

Observación. Se tiene que $g(n) \in O(n) \setminus \Omega(n)$, pero $g(n) \notin o(n)$.
Este ejemplo es un tanto controvertido. Cuando las funciones de complejidad si representan tiempos de complejidad de algoritmos reales, entonces, en efecto se tiene que las funciones en $g(n) \in O(n) \setminus \Omega(n)$ son las mismas que están en $o(n)$. La función g dada en el Ejemplo 2.8 no representa el tiempo de ejecución de ningún algoritmo, por lo cual se tiene tan controvertido resultado.

Ejemplo 2.9 Observe que:

a) $3n^2 + 10n + 7 \notin o(n^2)$, ya que es $\Theta(n^2)$
b) $3n^2 + 10n + 7 \in o(n^3)$, pues es $O(n^3)$ y no es $\Omega(n^3)$
c) $3n^2 + 10n + 7 \notin o(n)$, ya que no es $O(n)$.
d) $3n \log n + 5n + 10 \in o(n^2)$ pues es $O(n^2)$ y no es $\Omega(n^2)$
e) $3n \log n + 5n + 10 \in o(n^3)$ ya que es $O(n^3)$ y no es $\Omega(n^3)$
f) $5n^2 + 3\sqrt{n} + 123 \notin o(n^3)$ pues es $\Theta(n^3)$.

Teorema 2.2 $g(n) \in \Theta(f(n))$ *si sólo sí* $f(n) \in \Theta(g(n))$.

Ejemplo 2.10 $n^2 + 10n \in \Theta(n^2)$ *y* $n^2 \in \Theta(n^2 + 10n)$

Esto significa que Θ separa las funciones de complejidad en conjuntos disjuntos, a los cuales se les llama **Categoría de Complejidad**. Cualquier función de una categoría dada puede representar a la categoría. Por conveniencia, respresentamos a la categoría por su elemento más simple. La complejidad del Ejemplo 2.10 es representada por $\Theta(n^2)$.

Como hemos visto en algunos ejemplos, la complejidad de algunos algoritmos nos se incrementa con el valor de n, el tamaño de la entrada. Hemos indicado que estos algoritmos requieren tiempo constante, por simplicidad lo representaremos por $\Theta(1)$.

2.4.3. Propiedades del Orden.

A continuación se establecen algunas de las más importantes propiedades de orden que facilitar el determinar el orden de diversas funciones de complejidad.

1. $g(n)$ es $O(f(n))$ si y sólo sí $f(n)$ es $\Omega(g(n))$.
2. $g(n)$ es $\Theta(f(n))$ si y sólo sí $f(n)$ es $\Theta(g(n))$.
3. Si $b > 1$ y $a > 1 \implies \log_a n$ es $\Theta(\log_b n)$.
   Esto implica que todas las funciones de complejidad logarítmica están en la misma categoría o clase de complejidad, que se denota por $\Theta(\lg n)$.
4. Si $b > a > 0 \implies a^n$ es $o(b^n)$.
   Esto implica que todas las funciones de complejidad exponencial no están en la misma categoría de complejidad.
5. $\forall\ a > 0,\ a^n$ es $o(n!)$.
   Esto implica que $n!$ es *peor* que cualquier función de complejidad exponencial.
6. Considere la siguiente clasificación de categorías de complejidad:

   $$\Theta(\lg n)\ \ \Theta(n)\ \ \Theta(n \lg n)\ \ \Theta(n^2)\ \ \Theta(n^j)\ \ \Theta(n^k)\ \ \Theta(a^n)\ \ \Theta(b^n)\ \ \Theta(n!)$$

   donde, $k > j > 2$ y $b > a > 1$. Si una función de complejidad $g(n)$ está en una categoría a la izquierda de la categoría que contiene a $f(n)$, entonces $g(n) \in o(f(n))$. Esta propiedad establece que cualquier función logarítmica es eventualmente mejor que cualquier función polinomial, que a su vez es eventualmente mejor que cualquier función exponencial, la cual a su vez, es eventualmente mejor que cualquier función factorial.
7. Si $c \geq 0, g(n) \in O(f(n))$ y $h(n) \in O(f(n))$, entonces $c \cdot g(n) + d \cdot h(n) \in O(f(n))$.

Ejemplo 2.11 La Propiedad 3 establece que todas las funciones de complejidad logarítmica están en la misma categoría de orden.
Por ejemplo, $\Theta(\log_4 n) = \Theta(\log_2 n)$

Ejemplo 2.12 La Propiedad 6 establece que:

- cualquier función logarítmica es eventualmente mejor que cualquier función polinomial;
- cualquier función polinomial es eventualmente mejor que cualquier función exponencial;
- cualquier función exponencial es eventualmente mejor que cualquier función factorial.

Por ejemplo, $\log_2 n \in o(n)$, $n^{10} \in o(2^n)$, $2^n \in o(n!)$.

Ejemplo 2.13 Las Propiedad 6 y 7 pueden ser usadas repetidamente.

Por ejemplo, para demostrar que $f(n) = 5n + 3\log_2 n + 10n\log_2 n + 7n^2 \in \Theta(n^2)$ Aplicamos varias veces las Propiedad 6 y 7, como sigue:

Tenemos que	$7n^2 \in \Theta(n^2)$,
lo cual significa que	$10n\log_2 n + 7n^2 \in \Theta(n^2)$,
es decir	$3\log_2 n + 10n\log_2 n + 7n^2 \in \Theta(n^2)$,
por lo tanto	$5n + 3\log_2 n + 10n\log_2 n + 7n^2 \in \Theta(n^2)$.

En realidad en la práctica no hacemos esto, más bien tiramos los términos de menor orden, para calcular el tiempo de complejidad de un algoritmo. Cuando esto no es posible, regresamos a las definiciones de O y Ω para determinar el orden del algoritmo.

Notación. Para concluir esta sección, haremos un comentario sobre la notación. Algunos autores usan:

$$f(n) = \Theta(n^2) \quad \text{en vez de} \quad f(n) \in \Theta(n^2)$$

Ambas notaciones son correctas y significan lo mismo: que $f(n)$ es un miembro del conjunto $\Theta(n^2)$ Similarmente, es común usar:

$$f(n) = O(n^2) \quad \text{en vez de} \quad f(n) \in O(n^2).$$

2.5. Ejercicios.

Ejercicio 2.1 Sea P un problema. El desempeño computacional en el peor de los casos para P es $O(n^2)$. El tiempo de ejecución en el peor de los casos para P también es $\Omega(n \log_2 n)$. Sea A un algoritmo que soluciona P. ¿ Cuales de las siguientes afirmaciones resultan ser consistentes con la información dada sobre P? *Justifique su respuesta.*

a) A tiene en el peor caso complejidad $O(n^2)$.
b) A tiene en el peor caso complejidad $O(n^{3/2})$.
c) A tiene en el peor caso complejidad $O(n)$.
d) A tiene en el peor caso complejidad $\Theta(n^2)$.
e) A tiene en el peor caso complejidad $\Theta(n^3)$.

Ejercicio 2.2 Suponga que un algoritmo A se ejecuta en el peor caso con tiempo $f(n)$ y que el algoritmo B se ejecuta en el peor caso con tiempo $g(n)$. Responda a las siguientes preguntas con **sí, no** o **no sé**. *Justifique su respuesta.*

a) ¿ Es A es más rápido que B para toda n mayor que alguna n_o si $g(n) = \Omega(f(n) \log n)$?
b) ¿ Es A es más rápido que B para toda n mayor que alguna n_o si $g(n) = O(f(n) \log n)$?
c) ¿ Es A es más rápido que B para toda n mayor que alguna n_o si $g(n) = \Theta(f(n) \log n)$?
d) ¿ Es B es más rápido que A para toda n mayor que alguna n_o si $g(n) = \Omega(f(n) \log n)$?
e) ¿ Es B es más rápido que A para toda n mayor que alguna n_o si $g(n) = O(f(n) \log n)$?
f) ¿ Es B es más rápido que A para toda n mayor que alguna n_o si $g(n) = \Theta(f(n) \log n)$?

Ejercicio 2.3 Conteste las siguientes preguntas. *Explique brevemente su respuesta.*

a) Si se prueba que un algoritmo toma, en el peor de los casos, tiempo $O(n^2)$, ¿ es posible que tome tiempo $O(n)$ para algunos ejemplares ?
b) Si se prueba que un algoritmo toma, en el peor de los casos, tiempo $O(n^2)$, ¿ es posible que tome tiempo $O(n)$ para todos los ejemplares ?
c) Si se prueba que un algoritmo toma, en el peor de los casos, tiempo $\Theta(n^2)$, ¿ es posible que tome tiempo $O(n)$ para algunos ejemplares ?
d) Si se prueba que un algoritmo toma, en el peor de los casos, tiempo $\Theta(n^2)$, ¿ es posible que tome tiempo $O(n)$ para todos los ejemplares ?
e) Sea $f(n) = 100n^2$ para n par y $f(n) = 20n^2 - n \log n$ para n impar. ¿ Es $f(n) = \Theta(n^2)$?

Ejercicio 2.4 Conteste las siguientes preguntas *Justifique su respuesta.*

a) ¿ Es $3^n = O(2^n)$? **b)** ¿ Es $\log 3^n = O(\log 2^n)$?
c) ¿ Es $3^n = \Omega(2^n)$? **d)** ¿ Es $\log 3^n = \Omega(\log 2^n)$?

Ejercicio 2.5 Demuestre o de un contraejemplo de la siguiente afirmación.
Si $f(n) = O(F(n))$ y $g(n) = O(G(n))$ entonces $f(n)/g(n) = O(F(n)/G(n))$.

Ejercicio 2.6 Será cierto que $f(n) = O(g(n))$ implica que $2^{f(n)} = O(2^{g(n)})$? *Justifique su respuesta.*

Ejercicio 2.7 Demuestre o de un contraejemplo de la siguiente afirmación. Para todas las funciones $f(n)$ y $g(n)$ se tiene que $f(n) = O(g(n))$ o bien $g(n) = O(f(n))$

Ejercicio 2.8 Demuestre las siguientes afirmaciones.

a) Sea $f(n) \in O(g(n))$, se tiene además que $g(n) > 1 \forall n$.
Entonces $f(n) + c$, $\forall c$ está también en $O(g(n))$.
b) Sea $f(n) \in O(g(n))$, se tiene además que $g(n) > 1 \forall n$.
Entonces $c \times f(n)$, $\forall c$ está también en $O(g(n))$.

Ejercicio 2.9 Sea A un conjunto ordenado de n elementos. Escriba un algoritmo que imprima todos los subconjuntos de tres elementos de A. Calcule el desempeño computacional de su algoritmo.

Ejercicio 2.10 Determine el desempeño computacional en el peor de los casos y en el mejor de los casos para una versión del algoritmo InsertionSort que usa Búsqueda Binaria.

Ejercicio 2.11 Proporcione un algoritmo que calcule el máximo común divisor entre dos enteros. Calcule el desempeño computacional de su algoritmo.

Ejercicio 2.12 Proporcione un algoritmo que requiera tiempo lineal para ordenar n enteros distintos cuyos valores están entre 1 y 500.

Ejercicio 2.13 El algoritmo A realiza $10n^2$ operaciones elementales y el algoritmo B ejecuta $300 \ln n$ operaciones elementales. ¿Para qué valor de n el algoritmo B empieza a mostrar un mejor desempeño?

Ejercicio 2.14 Se tienen dos algoritmos $Alg1$ y $Alg2$ para un problema específico de tamaño n. $Alg1$ se ejecuta en n^2 microsegundos y $Alg2$ corre en $100n \log n$ microsegundos. El algoritmo $Alg1$ puede ser implementado usando 4 horas de tiempo de programador y requiere de 2 minutos de tiempo de CPU. Por otro lado, el $Alg2$ requiere de 15 horas de tiempo de programador y 6 minutos de tiempo de CPU. Si a los programadores se les paga \$20.00 la hora y el tiempo de CPU está cotizado a \$50.00 por minuto.
¿Cuántas veces debe el ejemplar de un problema de tamaño 500 ser resuelto usando el $Alg2$ para justificar su costo de desarrollo?

Ejercicio 2.15 Use las definiciones de O y Ω para mostrar que:

a) $f(n) = n^2 + 3n^3 \in \Theta(n^3)$
b) $6n^2 + 20n \in O(n^3)$ pero $6n^2 + 20n \notin \Omega(n^3)$

Ejercicio 2.16 Sea $p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$, donde $a_k > 0$. Usando las Propiedades de Orden muestre que $p(n) \in \Theta(n^k)$.

Ejercicio 2.17 Agrupe las siguientes funciones por categorías de complejidad:

$n \ln n$	$(\log n)^2$	$5n^2 + 7n$	$n^{5/2}$	$n!$
$2^{n!}$	4^n	n^n	$n^n + \ln n$	$5^{\log n}$
$\log(n!)$	$\sqrt{n}$	e^n	$8n + 12$	$10^n + n^{20}$

Ejercicio 2.18 Establecer las propiedades 1,2,6 y 7 de las Propiedades de Orden.

Ejercicio 2.19 Discutir las propiedades reflexiva, simétrica y transitiva de las comparaciones asintóticas (O, Ω, Θ, o).

Ejercicio 2.20 Suponga que tiene una computadora que requiere un minuto para resolver ejemplares de tamaño $n = 1000$. ¿Qué ejemplares pueden ser ejecutados en 1 minuto si tú compras una nueva computadora 1000 veces más rápida que la anterior, suponiendo las siguientes complejidades $T(n)$?

a. $T(n) \in \Theta(n)$ ***b.*** $T(n) \in \Theta(n^3)$ ***c.*** $T(n) \in \Theta(10^n)$

Ejercicio 2.21 Indique si los siguientes argumentos son correctos
Justifique su respuesta.

a. $\ln n$ *es* $O(n)$; ***b.*** n *es* $O(n \log n)$; ***c.*** $n \log n$ *es* $O(n^2)$;
d. 2^n *es* $\Omega(5^{\ln n})$; ***e.*** $\log^3(n)$ *es* $o(n^{0,5})$

Ejercicio 2.22 Actualmente podemos resolver problemas de tamaño 100 en 1 minuto usando el algoritmo A, que es un algoritmo $\Theta(2^n)$. Por otro lado, pronto podremos resolver problemas del doble de tamaño en 1 minuto. ¿Ayudaría comprar una computadora más rápida (y más cara)? *Justifique su respuesta.*

Ejercicio 2.23 Determine el desempeño computacional T(n) de los siguientes ciclos anidados. Para facilitar las operaciones aritméticas, puede suponer que n es potencia de 2. Esto es, $n = 2^k$ para algún entero positivo k.

```
...
    for i=1 to n do
        j <-- n;
        while j > 1 do
            <cuerpo del while>          {requiere Theta(1)}
            j <-- j div 2
        end_w
    end_for
...
```

Ejercicio 2.24 Dar un algoritmo para el siguiente problema y determinar sus desempeño computacional.

Problema: Dada una lista de n entreros positivos distintos, particionar la lista en dos sublistas, cada una de tamaño $n/2$, tal que la diferencia entre las sumas de sus elementos en las dos sublistas sea máxima. Se puede suponer que n es un múltiplo de 2.

Ejercicio 2.25 Determine el desempeño computacional T(n) de los siguientes ciclos anidados. Para facilitar las operaciones aritméticas, puede suponer que n es potencia de 2. Esto es, $n = 2^k$ para algún entero positivo k.

```
...
    i <-- n;
    while i > 0 do
        j <-- i;
        repeat
            <cuerpo del repeat>       {requiere Theta(1)}
            j <-- j * 2
        until j > n
        i <-- i div 2
    end_w
...
```

Ejercicio 2.26 Justifique la validez de las siguientes afirmaciones, suponiendo que $f(n)$ y $g(n)$ son funciones asintóticamente positivas y donde $o(fn))$ representa cualquier función $g(n)$ en $o(f(n))$.

a. $f(n) + g(n)$ *es* $O(máx\{f(n), g(n)\})$ ***b.*** $f^2(n)$ *es* $\Theta(f(n))$
c. $f(n) + o(f(n))$ *es* $\Theta(f(n))$

Ejercicio 2.27 Explique con palabras cuáles funciones están en los siguientes conjuntos

a. $n^{O(1)}$ ***b.*** $O(n^{O(1)})$ ***c.*** $O(O(n^{O(1)}))$

Ejercicio 2.28 Dar un algoritmo para el siguiente problema y determinar sus desempeño computacional.

Problema: Dada una lista de n entreros positivos distintos, particionar la lista en dos sublistas, cada una de tamaño $n/2$, tal que la diferencia entre las sumas de sus elementos en las dos sublistas sea mínima. Se puede suponer que n es un múltiplo de 2.

Ejercicio 2.29 Proporcione un algoritmo con desempeño computacional $\Theta(n^4 \log n)$.

Ejercicio 2.30 Determine el desempeño computacional de los siguientes programas. Suponga que el cuerpo del ciclo requiere tiempo constante.

a)

```
for (int i=0; i<n; i++) {
   ...
}
for (int j=n; j>=0; j--) {
   ...
}
```

b)

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    ...
  }
}
```

c)

```
for (int i=0; i<n; i++) {
  for (int j=0; j<i; j++) {
     ...
  }
}
```

d)

```
for (int i=n; i>0; i= i/2) {
   ...
}
```

e)

```
for (int i=0; i<n; i++) {
  for (int j=0; j*j<n; j++) {
    ....
  }
}
```

f)

```
for (int i=n; i>0; i=i>>1) {
  for (int j=0; j<n; j++) {
    ...
  }
}
```