UNIVERSIDAD NACIONAL DE SANTIAGO DEL ESTERO

Facultad de Ciencias Exactas y Tecnologías

Licenciatura en Sistemas de Información

PROGRAMACIÓN II

2011

COMPLEJIDAD Y
EFICIENCIA DE
ALGORITMOS

TEORÍA DE LA COMPLEJIDAD

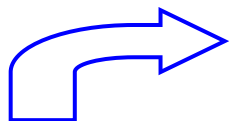
Dado un problema computacional:

- ¿Existe un algoritmo óptimo para resolver el problema? en tal caso,
- ¿Cuál será el criterio de optimalidad que deberá utilizarse?
- ¿Es posible medir el tiempo de ejecución o la memoria necesaria para cada posible algoritmo que resuelva el problema?

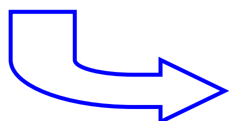
El área de las ciencias de la computación que se ocupa de estos temas se denomina **Teoría de la Complejidad** y los resultados que produce son medidas y criterios para determinar la eficiencia de los algoritmos.

ANÁLISIS DE ALGORITMOS

TIPO DE ANÁLISIS

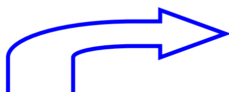


ANÁLISIS DE UN ALGORITMO EN PARTICULAR: se trata de investigar el costo de usar un determinado algoritmo para resolver un problema específico.



ANÁLISIS DE UNA CLASE DE ALGORITMOS: se investiga toda una familia de algoritmos que permiten resolver un problema, procurando obtener él que demande menor costo

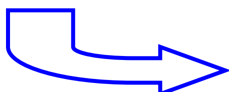
SEGÚN EL MOMENTO



ESTIMACIÓN A PRIORI

Hace uso de un modelo matemático, como lo es una función, basada en un computador idealizado y en un conjunto de operaciones con **costos** de ejecución perfectamente especificados.

Proporciona sólo un **RESULTADO APROXIMADO**.



COMPROBACIÓN A POSTERIORI

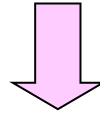
Se lleva a cabo en el momento de la ejecución del programa en un computador y consiste en medir los tiempos de corrida del programa en cuestión.

Proporciona **VALORES REALES**

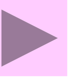
CLASIFICACIÓN DE LOS ALGORITMOS

¿CÓMO MEDIR LA EFICIENCIA DE LOS ALGORITMOS?

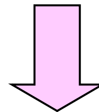
- Contando el número de pasos (**tiempo de ejecución** del algoritmo.)



COMPLEJIDAD EN TIEMPO



- Determinando el espacio utilizado por el agente computacional (máquina, persona) que lo ejecuta **(espacio** utilizado por el algoritmo.)

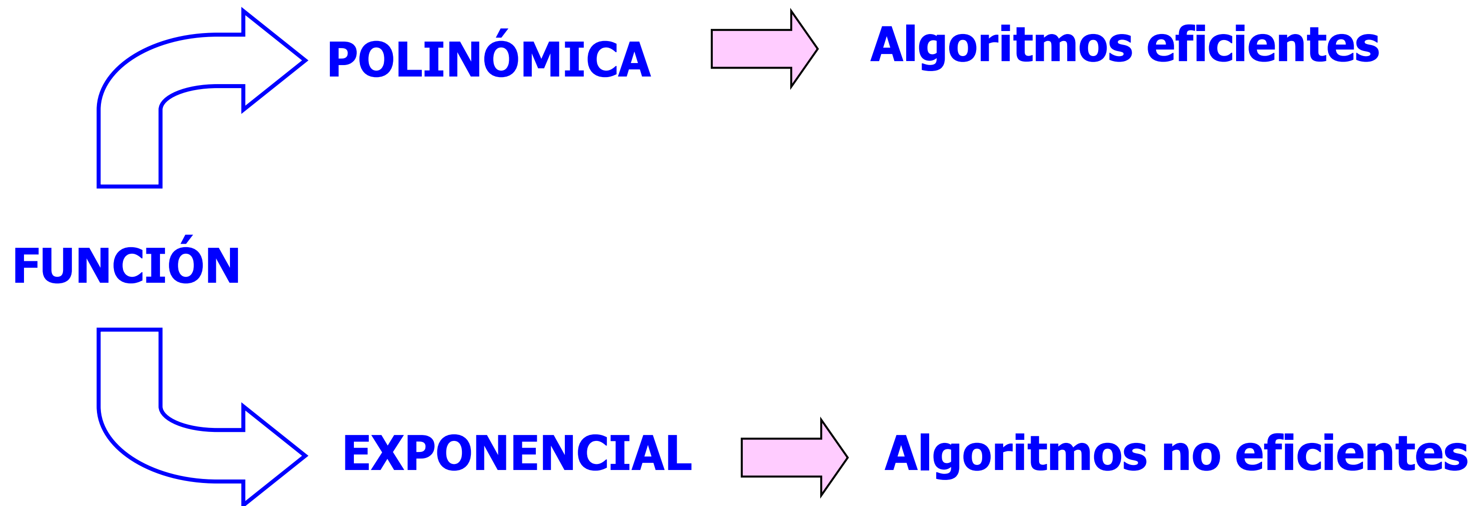


COMPLEJIDAD EN ESPACIO

FUNCIONES MATEMÁTICAS

Para evaluar la rapidez o viabilidad de un algoritmo (eficiencia) se imagina que al algoritmo se le suministra entradas cada vez mayores y se observa la razón de crecimiento del tiempo insumido para ejecutarlo.

Para analizar esto se consideran dos tipos de funciones matemáticas:



FUNCIONES POLINÓMICAS Y EXPONENCIALES

Función	Tipo	n=1	n=5	n =8	n= 10	n=11
$f(n)=n+1$	Pol	2	6	9	11	12
$f(n)=n^2+n+1$	Pol	3	31	73	111	133
$f(n)=n^3+n^2+n+1$	Pol	4	**156**	**585**	1111	**1464**
$f(n)=2^n$	Exp	2	**32**	**256**	1024	**2048**
$f(n)=n!$	Exp	1	120	40320	3628800	39916800
$f(n)=n^n$	Exp	1	3125	16777216	10 E 10	2,8 E 12

¿CÓMO MEDIR LA COMPLEJIDAD?

Ejemplo 1(search): problema de pertenencia a un conjunto.

```
procedure search (var X: int-set: y: integer, var found: boolean);
          var i: integer;
          begin
                i := 1;
                found := false;
                while i <= n  and not found do
                      if  X [ i ] = y
                        then
                            found := true;
                        else
                            i := i + 1;
                enddo
   end { search}
```

COMPLEJIDAD EN ESPACIO Y EN TIEMPO

Complejidad en espacio (número total de celdas requeridas): **E = |*X*| + k**
donde **|*X*|** es la longitud del arreglo (es decir, **n**) y
k es una constante que no depende del cardinal de **X (**celdas usadas para almacenar todas las otras variables y el código objeto).

Complejidad en tiempo*:* T = T1 + T2
donde **T1** es el tiempo insumido fuera del ciclo while y **T2** es el tiempo dentro de éste. Se observa que **T1** es un valor constante (**h**) que no depende del conjunto **X**, en cambio **T2** depende de él, así:

T = h + T2 (X)

El tiempo requerido por el procedimiento **search** es muy variable

- **mejor caso:** $T_b = h + r$
- **peor caso:** $T_w = h + r.\ n$
- **caso promedio:** $T_a = s.\ n + 1/2\ \ p\ .\ \ r$

donde:

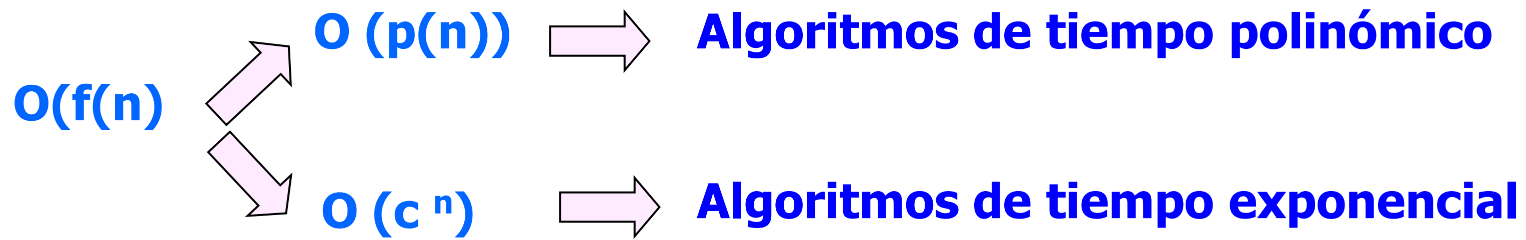
$$T_a = (1-p).n.r + \frac{p}{n}.r.\sum_{i=1}^{n} i = (1-p).r.n. + \frac{1}{2}.p.r.(n+1) = s.n + \frac{1}{2}.p.r \qquad s = r - \frac{1}{2}.p.r$$

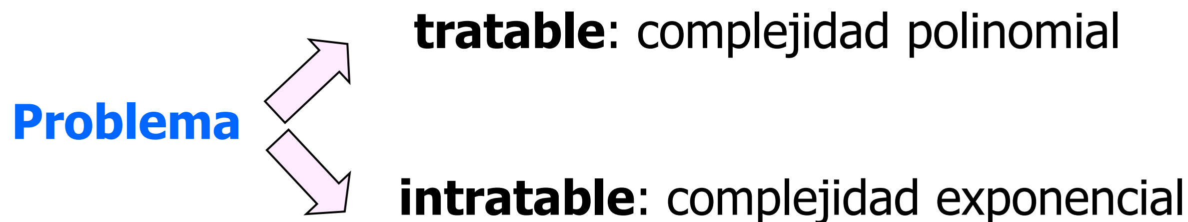
p : probabilidad que el **search** se ejecute para un valor **y** del conjunto
(1-p): es la probabilidad que ***y*** no esté en el conjunto
p / n es la probabilidad que el ciclo sea ejecutado **i** veces, $\forall 1 \leq i \leq n$

PROGRAMACIÓN II

DETERMINACIÓN DEL ORDEN DE UN ALGORITMO

Para diferenciar la eficiencia de los algoritmos se utilizan los **órdenes de complejidad**. Éstos se expresan en función de ***n*** que representa el tamaño (dado o estimado) de los datos de entrada.





Los algoritmos de complejidad mayor que **n·log n** son poco prácticos

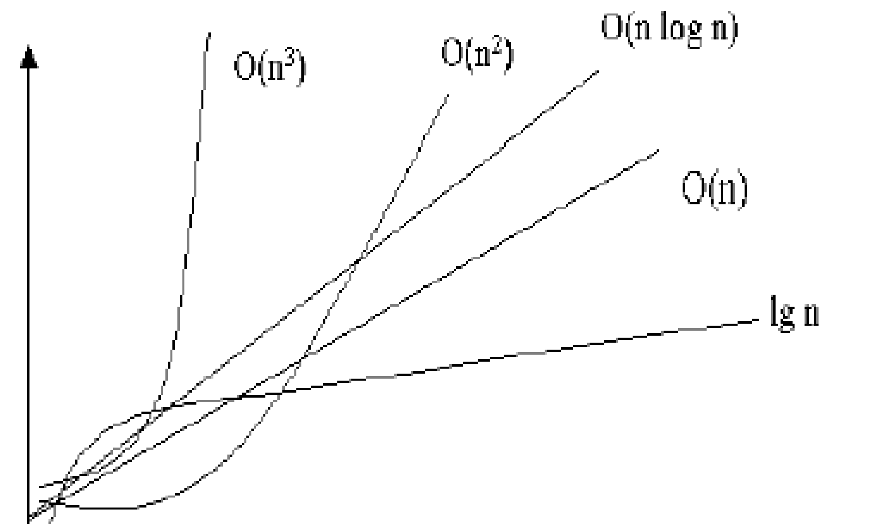
Los exponenciales son válidos para valores pequeños de n

OPERACIONES CON LA NOTACIÓN ORDEN

- **f(n) = 0(f(n))**: el orden de una función es igual al valor de la función.
- **c * f(n) = 0(f(n)):** el orden de una función multiplicada por una constante es igual al valor de la función.
- **0 (0(f(n))) = 0 (f(n)):** el orden del orden de una función es igual al orden de la función.
- **0 (f(n)) + 0 (g (n)) = 0 (max f(n),g (n)):** la suma de los ordenes de dos funciones es igual al orden máximo de las funciones.
- **0 (f(n)) * 0 (g (n)) = 0 (f(n) * g (n)):** el producto de dos ordenes de funciones es igual al orden del producto de las mismas.

ORDENES TÍPICOS

Notación	Denominación
$O(1)$	orden constante
$O(\log n)$	orden logarítmico
$O([\log n]^c)$	orden polilogarítmico
$O(n)$	orden lineal
$O(n \cdot \log n)$	orden lineal logarítmico
$O(n^2)$	orden cuadrático
$O(n^c)$	orden polinómico
$O(c^n)$, n > 1	orden exponencial
$O(n!)$	orden factorial
$O(n^n)$	orden doblemente exponencial



PRINCIPIOS PARA LA DETERMINACIÓN DEL ORDEN

TIPO DE INSTRUCCIÓN	ORDEN
➢ **Comando de asignación**, de **lectura** o de **escritura**	**0 (1)**
➢ **Secuencia de comandos**	**Instrucción de mayor tiempo de ejecución**
➢ **Instrucción de bifurcación**	**0 (1)** o bien el orden correspondiente al cuerpo de la bifurcación.
➢ **Lazo**	**Suma** de los tiempos de ejecución de: las instrucciones dentro del lazo + condición todo multiplicado por el número de veces que se repite el lazo.
➢ **Procedimientos no recursivos (subrutinas)**	Se evalúa por separado: 1. Procedimientos que no llaman a otros procedimientos. 2. A continuación deben ser evaluados los procedimientos que llaman a procedimientos que no llaman a otros procedimientos Este proceso debe repetirse hasta llegar al programa principal.

EJERCICIO 1

a) Enuncie el problema correspondiente.
b) Formalice el problema.
c) Analice la eficiencia y obtenga el tiempo de ejecución del peor caso
d) Exprese el orden correspondiente

```
procedure .................. (n: integer);
     var
          i, j, k : integer;
     begin
          for i:= 1 to n do
               for j:=1 to n do begin
                    C [ i, j ] := 0;
                    for k := 1 to n do
                    C [ i, j ] = C [ i ,j ] + A [ i, k] * B[ k,j ];
          end
           end
```

EJERCICIO 2

a) Enuncie el problema correspondiente.
b) Formalice el problema.
c) Analice la eficiencia y obtenga el tiempo de ejecución del peor caso
d) Exprese el orden correspondiente

```
begin
        s := 0
        k := 1
        for i = 1 to n-1 do
            for j = i + 1 to n do
                begin
                b[ k ] : = a[ i , j ]
                k := k + 1
                end
        for i = 1 to k-1 do
            s : = s + b[ i ]
        end
```

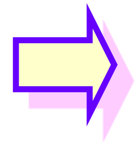
COMPORTAMIENTO ASINTÓTICO

El interés principal del análisis de algoritmos radica en saber cómo crece el tiempo de ejecución, cuando el tamaño de la entrada crece. Esto es la ***eficiencia asintótica*** del algoritmo. Se denomina "asintótica" porque analiza el comportamiento de las funciones en el *límite*, es decir, su tasa de crecimiento.

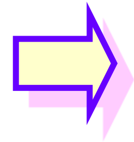
La **notación asintótica** se describe por medio de una función cuyo dominio es los números naturales (N) estimado a partir de tiempo de ejecución o de espacio de memoria de algoritmos en base a la longitud de la entrada. Se consideran las funciones asintóticamente no negativas.

La notación asintótica captura el **comportamiento** de la función para valores grandes de N.

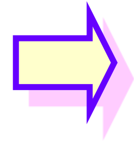
NOTACIONES ASINTÓTICAS



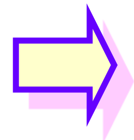
Notación "O grande" (O mayúscula) (Omicron, "Big-O") : se utiliza para manejar la cota superior del tiempo de ejecución.



Notación "o minúscula": denotar una cota superior que no es ajustada asintóticamente.



Notación "omega" (Ω): se utiliza para manejar la cota inferior del tiempo de ejecución



Notación "theta"(Θ): se utiliza para indicar el orden exacto de complejidad

NOTACIÓN ASINTÓTICA – COTAS SUPERIORES (I)

Notación "O grande" (O(f)), denota el conjunto de las funciones g que crecen a lo sumo tan rápido como f (es decir, f llega a ser en algún momento una cota superior para g).

Formalmente:

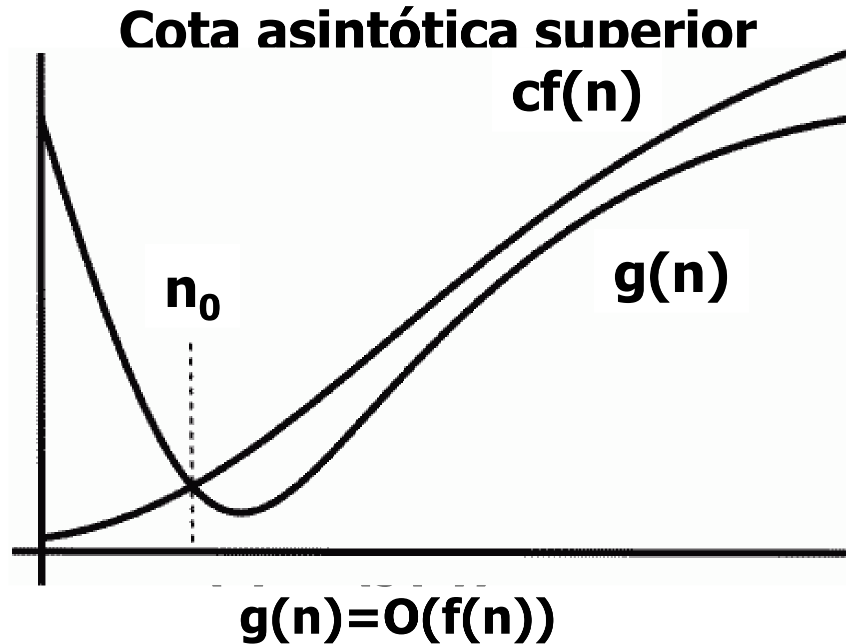
$$O(f) = \{g \mid \exists c_0 \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 \ g(n) \leq c_0 f(n)\}$$

Constante multiplicativa

n_0 indica a partir de que punto f es realmente una cota superior para g.

Se dice que la función *g(n)* "es de orden *f(n)*" [**O(f(n))**], si existen constantes positivas c_0 y n_0 tales que g*(n)* <= c_0 *f(n)* cuando *n* >= n_0

NOTACIÓN ASINTÓTICA – COTAS SUPERIORES (II)



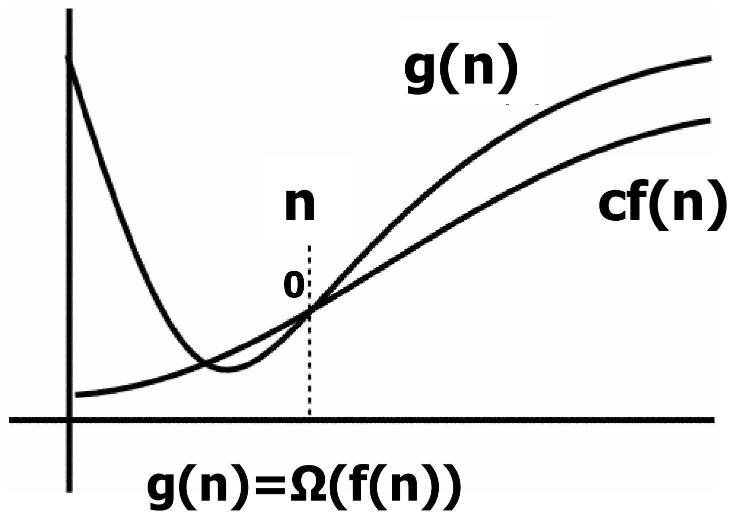
Se trata de buscar una función sencilla, f(n), que acote superiormente el crecimiento de otra g(n), en cuyo caso se notará como **g(n)∈ O(f(n)) (g es del orden de f).**

La función n^2+200n está **acotada superiormente** por n^2. Quiere decir que cuando n tiende a infinito el valor de 200 n se puede despreciar con respecto al de n^2.

Notación "o minúscula" (o(f)): La función n^2+200n puede ser acotada superiormente por la función n^3. Por tanto $n^2+200n = o(n^3)$, pero esta cota superior que **no está ajustada asintóticamente**.

NOTACIÓN ASINTÓTICA – COTA INFERIOR

Cota asintótica inferior

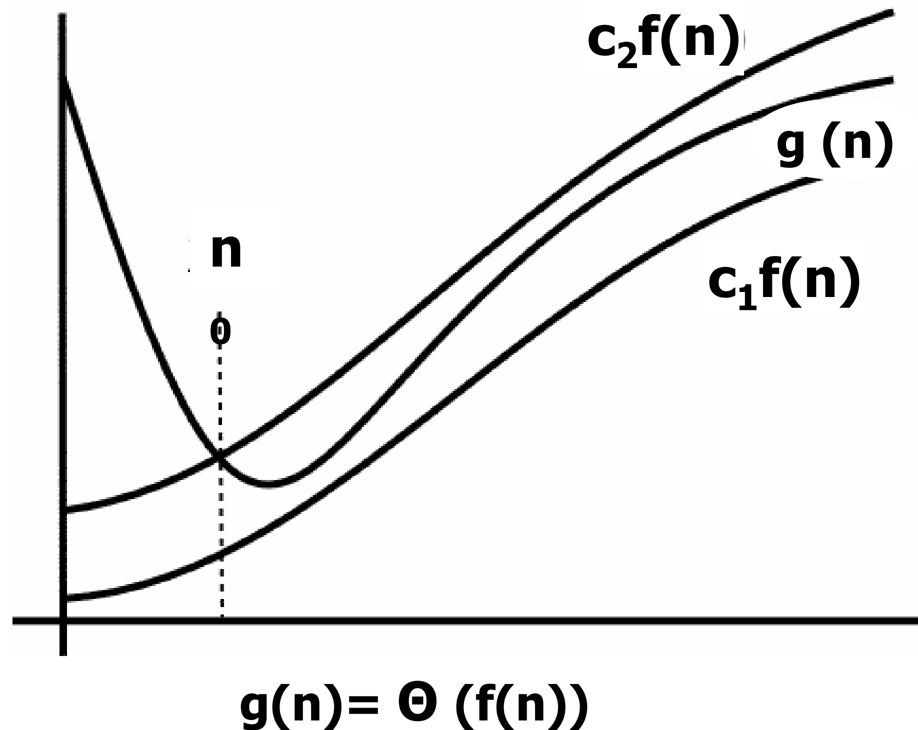


g(n) ≥ cf(n) para un número infinito de valores de **n**.

La función n^2 puede ser **acotada inferiormente** por la función n. Para demostrarlo basta notar que para todo valor de $n \geq 1$ se cumple $n \leq n^2$. Por tanto $n^2 = \Omega(n)$ (sin embargo, n no sirve como cota inferior para n^2).

La función $n^2+200\,n$ está **acotada inferiormente** por n^2. Quiere decir que cuando n tiende a infinito el valor de $200\,n$ se puede despreciar con respecto al de n^2.

NOTACIÓN ASINTÓTICA – COTA ASINTÓTICA AJUSTADA



$g(n)= \Theta\ (f(n))$

$g(n) = \Theta(f(n))$ **si y sólo si**
$g(n) = O(f(n))$ **y** $g(n) = \Omega(f(n))$

La función $n+10$ puede ser acotada por la función n. Para demostrarlo basta notar que para todo valor de $n \geq 1$ se cumple $g(n) \leq h(n) \leq 11g(n)$. Por tanto $n+10 = \Theta(x)$. La función $n^2+200\ n$ está **acotada de forma ajustada** por n^2. Quiere decir que cuando n tiende a infinito el valor de $200\ n$ se puede despreciar con respecto al de n^2.

COMPORTAMIENTO ASINTÓTICO

DEFINICIÓN 1: Sean **g**, **f** dos funciones de números naturales a números reales. La función **g** es **O(f(n))** si y sólo si existe una constante real **c > 0** tal que:

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = c$$

DEFINICIÓN 2: Sean **g** y **f** dos funciones sobre los naturales. **O(g(n)) < O(f(n))** si y sólo si

$$\lim_{n \to \infty} g(n) / f(n) = 0$$

Similarmente, **O (g(n)) > O (f(n))** si y sólo si $\lim_{n \to \infty} g(n) / f(n) = \infty$

DEFINICIÓN 3 Si **f(n)** domina asintóticamente a **g(n)** se dice que **g(n)=0(f(n))** , es decir, es de orden **f(n)** ya que **f(n)** es una **cota superior** a la tasa de crecimiento de **g(n)**. Para especificar una **cota inferior** para la velocidad de crecimiento de **g(n)** se usa la notación **Ω (h(n))**, que se lee "**g(n)** es omega mayúscula de **h(n)**" o "simplemente **g(n)** es omega de **h(n)**" lo cual significa que existe una constante **c** tal que: **g(n)** ≥ **ch(n)** para un número infinito de valores de **n**.

CONCLUSIONES (I)

- Existen dos clases de complejidad: la complejidad de tiempo y la complejidad de espacio. La **complejidad de tiempo** es más critica.

- Existen dos estudios posibles sobre el tiempo de ejecución de un algoritmo:
  - Obtener una **medida teórica a priori** mediante la función de complejidad
  - Obtener una **medida real del algoritmo**, a posteriori, para unos valores concretos en un computador determinado

Estimación a priori	Estimación a posteriori
Se analizan algoritmos	Se analizan programas
Se obtienen valores aproximados	Se obtienen valores exactos
Independiente de la máquina	Dependiente de la máquina
Permite hacer predicciones sobre el comportamiento del algoritmo en cualquier máquina	Poco predictivo

CONCLUSIONES (II)

- Los algoritmos eficientes se dice que son de orden de alguna función polinómica: **0(p(n))** (no interesa la forma exacta del polinomio **p**). Los del tiempo exponencial son **$0(c^n)$**, para alguna constante **c**.

- La complejidad depende del **tamaño de la entrada n.** Se puede realizar un análisis de complejidad para el peor, mejor y caso promedio.

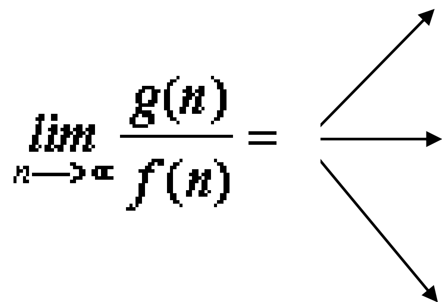
- La complejidad depende del **algoritmo diseñado** para solucionar un problema dado. Por ejemplo, el mismo problema (búsqueda en un conjunto ordenado) puede ser resuelto con diferentes complejidades por dos algoritmos diferentes (search y binary-search)

- Para determinar el tiempo de ejecución de un algoritmo se define una **función de costo o perfomance**, usando como parámetro de la misma el tamaño **n** de los datos del problema, escribiendo **f(n).**

CONCLUSIONES (III)

- El **orden** de una función es una **aproximación o estimación** del tiempo o espacio requerido para la ejecución de un algoritmo.

- La dominancia asintótica permite comparar dos algoritmos y determinar hasta que valor de **n** (datos de entrada) un algoritmo es más eficiente que otro.



$$\lim_{n \to \infty} \frac{g(n)}{f(n)} =$$

0 ⇒ O (g (n)) < O (f (n))

∞ ⇒ O (g (n)) > O (f (n))

c ⇒ O (g (n)) = O (f (n))

BIBLIOGRAFÍA

- KNUTH, Donald. ***Big Omicron and big Omega and big Theta***, ACM SIGACT News, Volume 8, Issue 2, 1976.
- KNUTH, Donald. ***The Art of Computer Programming***, **Volume 1**: *Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4.
- KNUTH, Donald. **El Arte de la Programación Computacional, Volumen 3**: Búsqueda y Ordenamiento, Segunda Edición. Addison-Wesley, p.158-168.
- Aho, J. Hopcrof y J. Ullman. ***The Design and Analysis of Computer Algorithms***. Addison-Wesley, 1974.
- Aho, J. Hopcrof y J. Ullman. ***Estructuras de Datos y Algoritmos***. Addison-Wesley, 1988.
- *Lewis*, Harry R. y *Papadimitriou*, Christos H. **Elements of the theory of computation**, Prentice Hall, 1981
- Paul E. Black, "**big-O notation**", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. Disponible en: http://www.itl.nist.gov/div897/sqg/dads/HTML/bigOnotation.html