Tiempo de Ejecución

Arturo Díaz Pérez

Sección de Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. CP 07300

Tel. 5061 3800 Ext. 6562, 6660
e-mail: adiaz@cs.cinvestav.mx

Midiendo el Tiempo de Ejecución

- Benchmarking
  - *Benchmarks*: una colección de entradas típicas representativas de una carga de trabajo para un programa
- Profiling
  - Asociar a cada instrucción de un programa un número que representa la fracción del tiempo total tomada para ejecutar esa instrucción particular
  - Regla del 90-10
    - Regla informal que afirma que el 90% del tiempo de ejecución se invierte en el 10% del código

Midiendo el Tiempo de Ejecución

- Análisis
  - Agrupar las entradas de acuerdo a su tamaño, n, y estimar el tiempo de ejecución del programa en entradas de ese tamaño, $T(n)$

El Problema de Ordenamiento

- **Entrada**: una secuencia de números

  $<a_1, a_2, ..., a_n>$
- **Salida**: una permutación de la secuencia

  $<a'_1, a'_2, ..., a'_n>$
- tal que**,**

  $a'_1 = a'_2 = ... = a'_n$

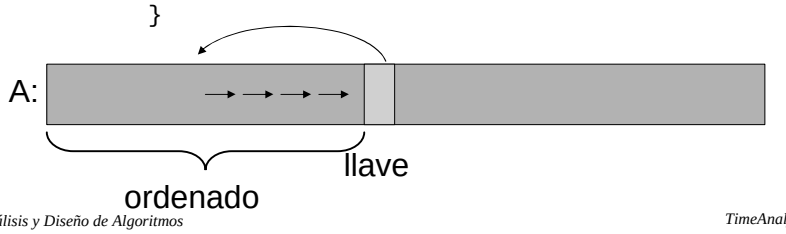
- **Ejemplo:**

25	**3**	**12**	**19**	**2**	**1**	**9**	**6**
1	**2**	**3**	**6**	**9**	**12**	**19**	**25**

Método de Inserción

```
void Inserción( int A[], int n )
{
 int i,j;

 for( i=1; i < n; i++ ) {
   j:= i;
   while( j > 0  &&  A[j] < A[j-1] ) {
     Intercambia( &A[j], &A[j-1] );
     j--
   }
 }
}
```



Método de Inserción: Ejemplo

25 3 12 19 2 1 9 6

3 25 12

3 12 25 19

3 12 19 25 2

2 3 12 19 25 1

1 2 3 12 19 25 9

1 2 3 9 12 19 25 6

1 2 3 6 9 12 19 25

La Entrada del Problema

- Tiempo de Ejecución
  - Debe ser definido como una función de la entrada
    - $T_p = f(E)$
  - Con frecuencia el tiempo de ejecución depende del *tamaño de la entrada*
    - $T(n)$: El tiempo de ejecución de un programa en entradas de tamaño n
      - Ejemplo: $T(n) = c\ n^2$, para alguna constante c

Tipos de Análisis

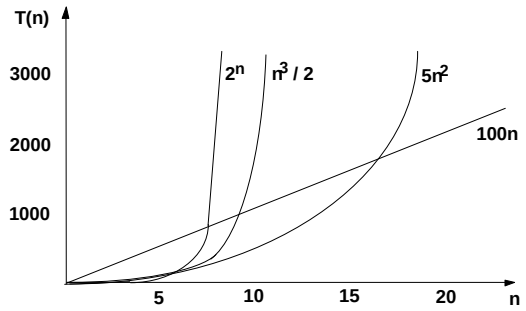
- El tiempo de ejecución del peor caso, $\boldsymbol{T_w(n)}$
  - El máximo de los tiempos de ejecución sobre todas las entradas de tamaño n
  - Puede no ser muy fiel
- El tiempo promedio de ejecución: $\boldsymbol{T_a(n)}$
  - El promedio de los tiempos de ejecución sobre todas las entradas de tamaño n
  - Puede ser más fiel
  - En algunas ocasiones puede ser difícil de determinar
- El tiempo de ejecución del mejor caso: $\boldsymbol{T_b(n)}$
  - El menor de los tiempos de ejecución sobre todas las entradas de tamaño n
  - Puede ser engañoso en un algoritmo lento que trabaja rápido sobre algunas entradas

Tiempo independiente de la computadora

- El tiempo de ejecución no debe ser expresado en unidades de tiempo estándar
- ¿Qué significa el tiempo del peor caso para el método de inserción?
  - ← Depende de la velocidad de una computadora
    - Velocidad relativa (en la misma computadora)
    - Velocidad absoluta (en computadoras diferentes)
- Ignore las constantes dependientes de la computadora
- Observe el crecimiento de $T(n)$ conforme $n \to a$.
  - El tiempo de ejecución de un algoritmo es proporcional a $f(n)$

Análisis Asintótico

Comparando Tiempos de Ejecución



Running Time T(n)	Max. Problem Size for 10^3 sec	Max. Problem Size for 10^4 sec	Increase in Max. Problem Size
100 n	10	100	10.0
$5n^2$	14	45	3.2
$n^3 / 2$	12	27	2.3
2^n	10	13	1.3

Función de Complejidad

- Definición:
  - Una *función de complejidad* puede ser cualquier función de los enteros no negativos a los reales no negativos
  - f: $\mathbf{N} \rightarrow \mathbf{R}^{\geq 0}$
- Ejemplos:
  - $f\ (n) = n$
  - $f\ (n) = n^2$
  - $f\ (n) = \log n$
  - $f\ (n) = 3n\ + 4n^2$

Ordenes de Crecimiento

- Dada f: N $\rightarrow$ R, una función con valores no negativos

$$O\ (f(n)) = \{\ g: \mathbf{N} \rightarrow \mathbf{R}^{\geq 0} \mid \exists\ c > 0 \text{ y } N_0 \in \mathbf{N}: (n \geq N_0 \Rightarrow g(n) \leq c\ * f(n))\ \}$$

$$\Theta\ (f(n)) = \{\ g: \mathbf{N} \rightarrow \mathbf{R}^{\geq 0} \mid [g \in O(f)] \wedge [f \in O(g)]\ \}$$

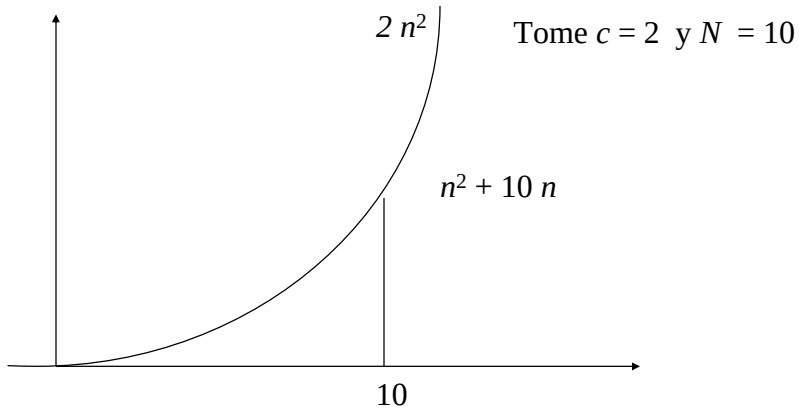
$$\Omega\ (f(n)) = \{\ g: \mathbf{N} \rightarrow \mathbf{R}^{\geq 0} \mid \exists\ c > 0 \text{ y } N_0 \in \mathbf{N}: (n \geq N_0 \Rightarrow f(n) \leq c\ * g\ (n))\ \}$$

Ordenes de Crecimiento

- Se escribe $g(n) = \mathcal{O}(f(n))$ en lugar de $g \in \mathcal{O}(f)$
- El crecimiento de g está dominado por el de f, si y solo si, $g(n) = O(f(n))$
- g y f poseen el mismo orden de crecimiento, si y solo si, $g(n) = \Theta(f(n))$
- El crecimiento de g es al menos el de f, si y solo si, $g(n) = \Omega(f(n))$

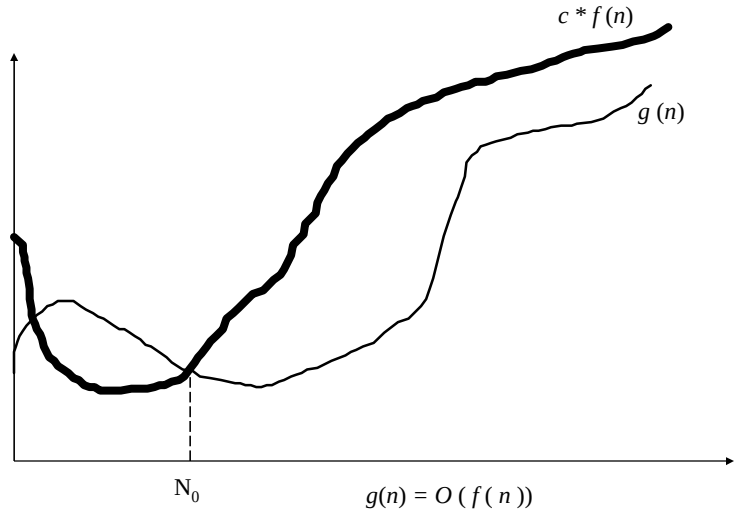
$O(f(n))$: De orden $f(n)$

$n^2 + 10\,n$ es $O(n^2)$ ¿Por qué?



Tome $c = 2$ y $N = 10$

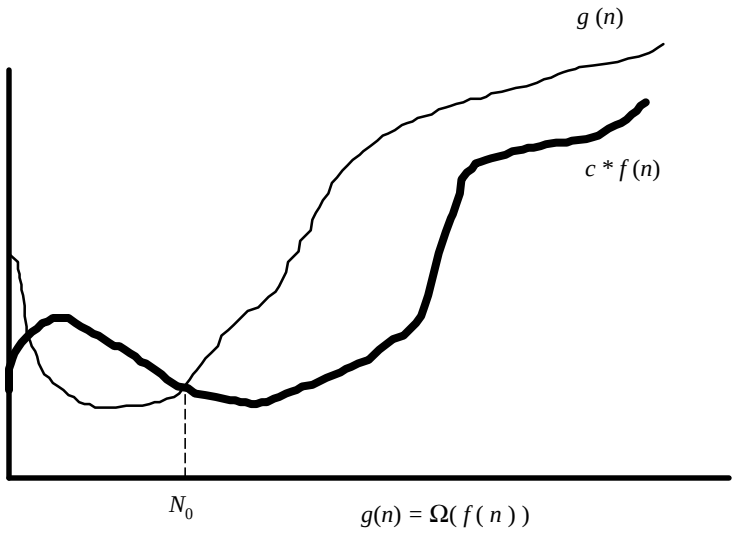
Cota Superior Asintótica: *O*



O: Ejemplos

- ¿1,000,000n^2 es $O(n^2)$?
- ¿$(n - 1)n / 2$ es $O(n^2)$?
- ¿$n / 2$ es $O(n^2)$?
- ¿log $(n^{1000000})$ es $O(n)$?
- ¿n^2 es $O(n)$?

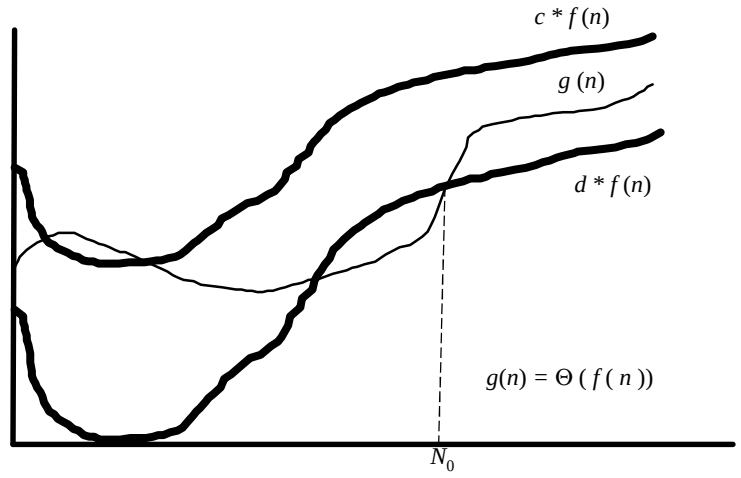
Cota Inferior Asintótica: W



W: Ejemplos

- ¿1,000,000 n^2 es $\Omega(n^2)$?
- ¿$(n - 1)n / 2$ es $\Omega(n^2)$?
- ¿$n / 2$ es $\Omega(n^2)$?
- ¿log $(n^{1000000})$ es $\Omega(n)$?
- ¿n^2 es $\Omega(n)$?

Θ: Mismo Orden de Crecimiento



Θ: Ejemplos

- ¿1,000,000 n es $\Theta(n^2)$?
- ¿$(n - 1)n / 2$ es $\Theta(n^2)$?
- ¿$n / 2$ es $\Theta(n^2)$?
- ¿$\log (n^{1000000})$ es $\Theta(n)$?
- ¿n^2 es $\Theta(n)$?

Observaciones

- Para cualesquiera $f, g: \mathbf{N} \to \mathbf{R}$:
  - $g(n) = O(f(n)) \Leftrightarrow f(n) = \Omega(g(n))$
  - $g(n) = \Theta(f(n)) \Leftrightarrow [g(n) = O(f(n))] \wedge [g(n) = \Omega(f(n))]$

Observaciones

- También se define
  - $o(f(n)) = \{ g: \mathbf{N} \to \mathbf{R}^{\geq 0} \mid \forall c > 0,\ \exists N_0 \in \mathbf{N}: (n \geq N_0 \Rightarrow g(n) \leq c * f(n)) \}$
  - $w(f(n)) = \{ g: \mathbf{N} \to \mathbf{R}^{\geq 0} \mid f \in o(g(n)) \}$
  - esto es,
    - $g(n) = o(f(n)) \Leftrightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
    - $g(n) = w(f(n)) \Leftrightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = +\infty$

Propiedades

- Transitividad: $\forall \lozenge \in \{O, \Theta, \Omega, o, w\}$:
  - $g(n) = \lozenge(f(n)) \wedge f(n) = \lozenge(h(n)) \Rightarrow g(n) = \lozenge(h(n))$
- Reflexividad: $\forall \lozenge \in \{O, \Theta, \Omega\}$:
  - $f(n) = \lozenge(f(n))$
- Simetría:
  - $g(n) = \Theta(f(n)) \Leftrightarrow f(n) = \Theta(g(n))$
  - Por lo tanto, $g(n) = \Theta(f(n))$ define una *relación de equivalencia* en el espacio de funciones

Propiedades

- Simetría transpuesta:
  - $f(n) = O(g(n))$, si y solo si, $g(n) = \Omega(f(n))$
  - $f(n) = O(g(n))$, si y solo si, $g(n) = \omega(f(n))$

Diferencia entre *O* y *o*

$O(f(n))$ = $\{g: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} |$: **existen constantes** positivas c and N_0 tal que $g(n) \leq c * f(n)$ para toda $n \geq N_0\}$

$o(f(n))$ = $\{g: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0} |$: **para toda constante** positiva c existe una constante $N_c > 0$, tal que, $g(n) \leq c * f(n)$ para toda $n \geq N_c$ $\}$

- ← Para *o* la desigualdad se mantiene para todas las constantes positivas
- ← Mientras que para *O* la desigualdad se mantiene para algunas constantes positivas

Analogía con los Números Reales

- ☞ $f(n) = O(g(n)) \approx a \leq b$
- ☞ $f(n) = \Omega(g(n)) \approx a \geq b$
- ☞ $f(n) = \Theta(g(n)) \approx a = b$
- ☞ $f(n) = o(g(n)) \approx a < b$
- ☞ $f(n) = \omega(g(n)) \approx a > b$

Observaciones

- A diferencia de los números reales NO todas las funciones son asintóticamente comparables
  - ← Ejemplo: $n^{1+\sin n}$ y n
- El conjunto $o(f(n))$ NO es el mismo que el conjunto $O(f(n)) - \Omega(f(n))$
  - ← Ejemplo:

$$g(n) = \begin{cases} n & \text{si } n \text{ es par} \\ 1 & \text{si } n \text{ es impar} \end{cases}$$

Observaciones

- Los límites se pueden usar para determinar el orden

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \begin{cases} c & \text{entonces, } g(n) = O(f(n)) \text{ si } c > 0 \\ 0 & \text{entonces, } g(n) = o(f(n)) \\ \infty & \text{entonces, } f(n) = o(g(n)) \end{cases}$$

Big O Revisada

- $T(n)$ es un $O(f(n))$, leído como "O de $f(n)$", si existe una constante positiva c y n_0, tales que, $T(n) \leq cf(n)$ para todo $n \geq n_0$
- Factores constantes "no importan"
  - Para cualquier constante positiva d y cualquier función $T(n)$, $T(n)$ es $O(dT(n))$
  - Si $T(n)$ es $O(f(n))$, entonces, $T(n)$ es $O(df(n))$ para cualquier $d > 0$
- Términos de orden inferior "no importan"
  - Si $T(n)$ es un polinomio de la forma $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$, tal que, $a_k > 0$, entonces, $T(n)$ es $O(n^k)$

Big O Revisada

- Si $p(n)$ son $q(n)$ y son polinomios y el grado de $q(n)$ es mayor o igual al grado de $p(n)$, entonces, $p(n)$ es $O(q(n))$
- $p(n)$ es $O(a^n)$, exponenciales, a^n, $a > 1$, crecen más rápido que cualquier polinomio, $p(n)$
- $f(n)$ es una **cota *O* ajustada** de $T(n)$ si
  - $T(n)$ es $O(f(n))$
  - Si $T(n)$ es $O(g(n))$, entonces, $f(n)$ es $O(g(n))$

Regla de la Suma

- Supongamos que $T_1(n)$ y $T_2(n)$ son los tiempos de ejecución de dos fragmentos de programa P_1 y P_2, respectivamente y que $T_1(n)$ es $O(f(n))$ y $T_2(n)$ es $O(g(n))$, entonces, $T_1(n) + T_2(n)$ es $O(\max(f(n), g(n)))$

  - Si $T_1(n)$ es $O(f(n))$, $\exists\ c_1, n_1$, tales que, $T_1(n) \leq c_1 f(n)$, $\forall\ n \geq n_1$
  - Si $T_2(n)$ es $O(g(n))$, $\exists\ c_2, n_2$, tales que, $T_2(n) \leq c_2 g(n)$, $\forall\ n \geq n_2$
  - Sea $n_0 = \max(n_1, n_2)$, $\forall\ n \geq n_0$, $T_1(n) + T_2(n) \leq c_1 f(n) + c_2 g(n)$
  - De aquí que, $\forall\ n \geq n_0$, $T_1(n) + T_2(n) \leq (c_1 + c_2) \max(f(n), g(n))$
  - Por lo tanto, $T_1(n) + T_2(n)$ es un $O(\max(f(n), g(n)))$

Regla del Producto

- Regla del producto
  - Si $T_1(n)$ y $T_2(n)$ son $O(f(n))$ y $O(g(n))$, respectivamente, entonces, $T_1(n) * T_2(n)$ es $O(f(n) * g(n))$
- Transitividad
  - Si $g(n) = O(f(n)) \wedge f(n) = O(h(n)) \Rightarrow g(n) = O(h(n))$

Reglas Generales de Análisis

- Un asignamiento e instrucciones de lectura y escritura que no contengan llamados a funciones toman un tiempo constante O(1)
  - Cuando existen llamados a funciones, se debe tomar en cuenta el tiempo que toma la ejecución de la función
- El tiempo de ejecución de una secuencia de proposiciones se determina por la regla de la suma

  $T(S_1; S_2; \ldots; S_n)$ es un $O(\max(T(S_1), T(S_2), \ldots, T(S_n)))$

Reglas Generales de Análisis

- El tiempo de ejecución de una proposición ***if*-*then*** esta determinado por el tiempo de ejecución de las proposiciones condicionales más el tiempo para evaluar la condición

  $T(\textbf{if } C \textbf{ then } S) = T(C) + T(S)$

- El tiempo de ejecución de una proposición ***if*-*then*-*else*** es el tiempo para evaluar la condición más el mayor de los tiempos necesarios para evaluar las proposiciones cuando la condición es verdadera y el tiempo para las proposiciones ejecutadas cuando la condición es falsa

  $T(\textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2) = T(C) + \max(T(S_1), T(S_2))$

Reglas Generales de Análisis

- El tiempo para ejecutar un ciclo es la suma, sobre todas las veces que se ejecuta el ciclo del tiempo para ejecutar el ciclo más el tiempo para evaluar la condición de terminación

$$T(\textbf{while } C \textbf{ do } S) = \sum_{\substack{\text{las veces que} \\ \text{se ejecuta el ciclo}}} [T(C) + T(S)] + T(C)$$

Reglas Generales de Análisis

- Ejemplo

```
void Inserción( int A[], int n )
{
 int  i,j;

 for( i=1; i < n; i++ ) {
   j:= i;
   while( j > 0  &&  A[j] < A[j-1] ) {
     Intercambia( &A[j], &A[j-1] );
     j--
   }
 }
}
```

$O(i)$ $O(n)$