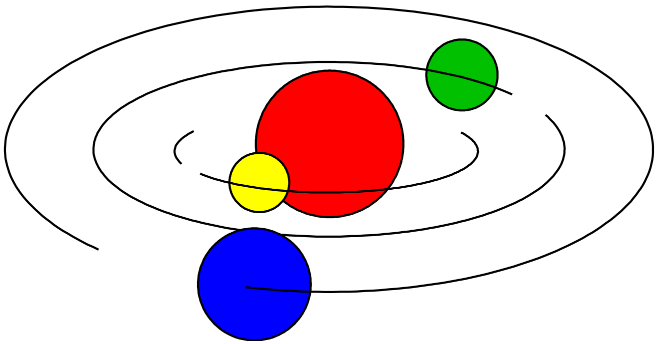
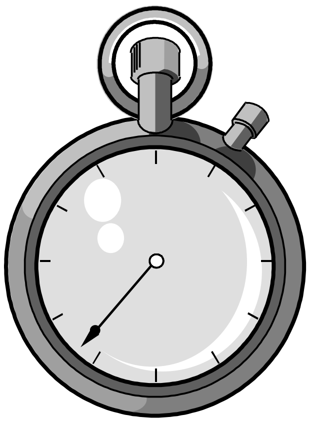
Técnicas avanzadas de análisis y diseño

- Análisis de algoritmos
  - Notaciones
  - Análisis de algoritmos
  - Análisis probabilístico
  - Análisis amortizado

Operaciones características y complejidad de cálculo. Funciones, sumatorias y recurrencias. Algoritmos recursivos e iterativos. Evaluación de la eficiencia y notación O.

Análisis de algoritmos

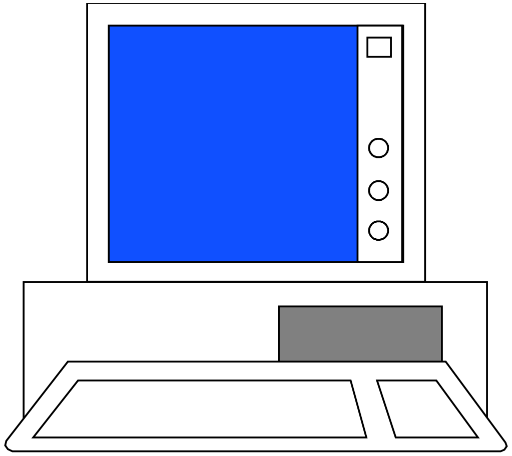




- La eficiencia de un programa tiene dos ingredientes fundamentales: espacio y tiempo.
- La eficiencia en espacio es una medida de la cantidad de memoria requerida por un programa.
- La eficiencia en tiempo se mide en términos de la cantidad de tiempo de ejecución del programa.

Análisis de algoritmos

- Ambas dependen del tipo de computador y compilador, por lo que **no** se estudiará aqui la eficiencia de los programas, sino la eficiencia de los algoritmos.



- Asimismo, este análisis dependerá de si trabajamos con máquinas de un solo procesador o de varios de ellos. Centraremos nuestra atención en los algoritmos para máquinas de un solo procesador que ejecutan una instruc-ción luego de otra.

Operaciones características y complejidad de cálculo

- La eficiencia de los algoritmos está basada en una operación característica que el algoritmo repite y que define su **complejidad en Tiempo (T(n))**.
- **T(n)** es el número de operaciones características que el algoritmo desarrolla para una entrada N dada.
- El máximo tiempo de ejecución de un algoritmo para todas las instancias de tamaño N, se denomina la complejidad en tiempo para el peor caso W(n). Asimismo, la complejidad promedio en tiempo es A(n), donde p_j es la probabilidad de que esta instancia ocurra.

$$W(n) = \max_{1 \leq j \leq k} T_j(n) \qquad A(n) = \sum_{j=1}^{k} p_j \, T_j(n)$$

- Normalmente se tendrán muchos algoritmos diferentes para resolver un mismo problema, por lo que debe existir un criterio para seleccionar el mejor.

Notaciones

- 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.
- La notación asintótica se describe por medio de una función cuyo dominio es los números naturales (N). Se consideran las funciones asintóticamente no negativas.

1.- **Notación Θ**, límite asintóticamente estrecho

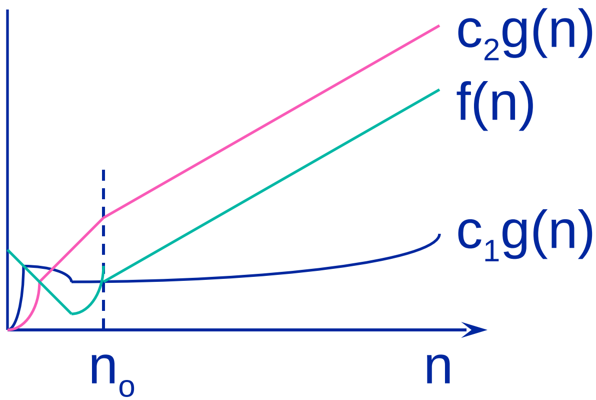
Para una función dada g(n),

$\Theta(g(n)) = \{ f(n) \,/\, \exists$ las constantes positivas c_1, c_2 y n_o /

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall n \ge n_o \}$$

Ejm: $n^2/2 - 3n = \Theta(n^2)$

Notaciones



Se denota
$f(n) = \Theta(g(n)) \equiv f(n) \in \Theta(g(n))$,
y se dice que g(n) es el límite asintóticamente estrecho de f(n).

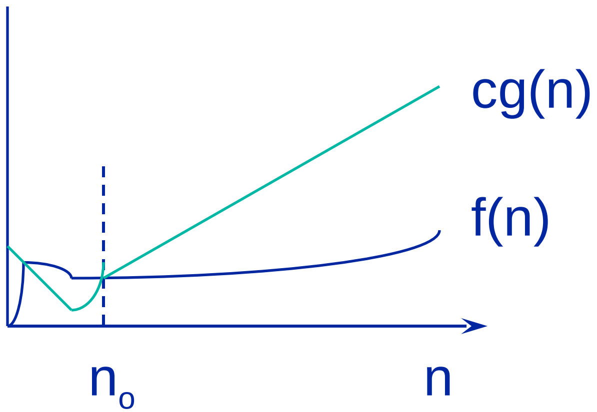
2.- **Notación O**, límite asintótico superior

$O(g(n)) = \{ f(n) \,/\, \exists$ las constantes positivas c y n_o /

$0 \leq f(n) \leq c\, g(n) \;\forall n \geq n_o\}$

Ejm: $an^2 + bn + c$ con $a > 0$ tiene $O(n^2)$

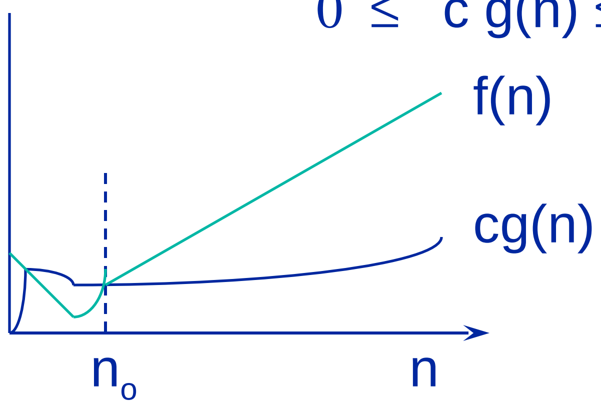
Notaciones



El hecho que $f(n) = \Theta(g(n))$ implica $f(n) = O(g(n))$, por lo que $\Theta(g(n)) \leq O(g(n))$.
Se dice que g(n) es el límite asintótico superior de f(n).

3.- **Notación Ω**, límite asintótico inferior

$$\Omega(g(n)) = \{ f(n) \,/\, \exists \text{ las constantes positivas } c \text{ y } n_o \,/\, 0 \leq c\, g(n) \leq f(n) \;\forall n \geq n_o \}$$



Ejm: $an^2 + bn + c$ con $a > 0$ tiene $O(n^2) = \Omega(n^2) = \Theta(n^2)$

Notaciones

Teorema: Para cualquier función f(n) y g(n), $f(n) = \Theta(g(n))$ si y solo si
$f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$

4.- **Notación o**, límite superior no asintóticamente estrecho

$o(g(n)) = \{ f(n)$ / para cualquier constante $c > 0$, $\exists$ una constante
$n_o > 0 \; / \; 0 \leq f(n) < c\, g(n) \; \forall n \geq n_o \}$

Ejm: $2n = o(n^2)$ pero $2n^2 \neq o(n^2)$

5.- **Notación ω**, límite inferior no asintóticamente estrecho

$\omega(g(n)) = \{ f(n)$ / para cualquier constante $c > 0$, $\exists$ una constante
$n_o > 0 \; / \; 0 \leq c\, g(n) < f(n) \; \forall n \geq n_o \}$

Ejm: $n^2/2 = \omega(n)$ pero $n^2/2 \neq \omega(n^2)$

Notaciones

Comparación de las funciones: f(n) y g(n) son asintóticamente +

1.- Transitividad

Si $f(n) = \Theta(g(n))$ y $g(n) = \Theta(h(n))$ entonces $f(n) = \Theta(h(n))$

Si $f(n) = O(g(n))$ y $g(n) = O(h(n))$ entonces $f(n) = O(h(n))$

Si $f(n) = \Omega(g(n))$ y $g(n) = \Omega(h(n))$ entonces $f(n) = \Omega(h(n))$

Si $f(n) = o(g(n))$ y $g(n) = o(h(n))$ entonces $f(n) = o(h(n))$

Si $f(n) = \omega(g(n))$ y $g(n) = \omega(h(n))$ entonces $f(n) = \omega(h(n))$

2.- Reflexibidad

$f(n) = \Theta(f(n))$, $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$

3.- Simetría

$f(n) = \Theta(g(n))$ si y solo si $g(n) = \Theta(f(n))$

4.- Simetría transpuesta

$f(n) = O(g(n))$ si y solo si $g(n) = \Theta(g(n))$

$f(n) = o\ (g(n))$ si y solo si $g(n) = \omega(g(n))$

Notaciones

Analogía entre la comparación asitótica de dos funciones f y g y la comparación entre dos números reales a y b:

$f(n) = O(g(n)) \approx a \leq b$

$f(n) = \Omega(g(n)) \approx a \geq b$

$f(n) = \Theta(g(n)) \approx a = b$

$f(n) = o(g(n)) \approx a < b$

$f(n) = \omega(g(n)) \approx a > b$

Para dos números reales cualesquiera una de las siguientes comparaciones es cierta: $a < b$, $a = b$ o $a > b$.

No todas las funciones son comparables asintóticamente. Ejemplo: $f(n) = n$ y $g(n) = n^{1+sen(n)}$, no pueden ser comparadas usando la notación sintótica pues el valor del exponente 1+sen(n) oscila entre 0 y 2 tomando todos los valores en ese rango.

Funciones y notaciones comunes

1.- Monotonía: Una función f(n) es *monótona creciente* si $m \leq n$ implica $f(m) \leq f(n)$. Es *monótona decreciente* si $m \leq n$ implica $f(m) \geq f(n)$
Una función f(n) es *estrictamente creciente* si $m < n$ implica $f(m) < f(n)$ y *estrictamente decreciente* si $m < n$ implica $f(m) > f(n)$.

2.- Pisos y techos: Para cualquier número real x, el *piso* de x $\lfloor x \rfloor$, es el número entero más grande menor o igual a x, y el *techo* de x $\lceil x \rceil$ es el número entero más pequeño mayor o igual a x.

$\forall$ número real x, $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$

Para cualquier entero n, $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$

Para cualquier entero n y los enteros $a \neq 0$ y $b \neq 0$, $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$ y $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$. Ambas funciones son monótonas crecientes.

3.- Polinomios: Dado un entero positivo d, un polinomio en n de grado d es una función $p(n) = \sum_{i=0}^{d} a_i n^i$ donde $a_0, a_1, ..., a_d$ son coeficientes del polinomio y $a_d \neq 0$. Un polinomio es asintóticamente positivo si $a_d > 0$.

Funciones y notaciones comunes

Para un polinomio asintóticamente positivo de grado d $p(n)=\Theta(n^d)$.
La función n^d es monótona creciente si $d \geq 0$, y monótona decreciente si $d \leq 0$. La función f(n) es polinomialmente limitada si $f(n) = n^{O(1)}$, que es equivalente a $f(n) = O(n^k)$ para alguna constante k.

4.- Exponenciales: Para todo real $a \neq 0$, m y n se tienen las siguientes identidades:

$a^0 = 1$	$a^1 = a$	$a^{-1} = 1/a$
$(a^m)^n = a^{mn}$	$(a^m)^n = (a^n)^m$	$a^m a^n = a^{m+n}$

para todo n y $a \geq 1$, la función a^n es monótona decreciente en n.
Para toda constante real a y b tal que $a > 1$, $\lim_{n\to\infty} \frac{n^b}{a^n} = 0$ por lo que $n^b = o(a^n)$, asi cualquier función exponencial positiva crece más rápido que cualquier polinomio. Siendo e = 2.71828..., la base de la función logaritmo natural, se tiene que para todo real x, $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$
Para todo real x, se tiene que $e^x \geq 1 + x$, cuando x = 0 es igual.

Funciones y notaciones comunes

Cuando $|x| \leq 1$, se tiene la aproximación $1 + x \leq e^x \leq 1 + x + x^2$.
Cuando $x \rightarrow 0$, la aproximación de e^x por $1 + x$ es buena,
$e^x = 1 + x + \Theta(x^2)$
Para todo x, $\lim_{n\rightarrow\infty}\left(1+\frac{x}{n}\right)^n = e^x$

5.- Logaritmos: Se asume que lg n + k = (lg n) + k.

$\lg n = \log_2 n$ (logaritmo binario) $\ln n = \log_e n$ (logaritmo natural)

$\lg^k n = (\lg n)^k$ (exponenciación) $\lg \lg n = \lg(\lg n)$

Para n > 0 y b > 1, la función $\log_b n$ es estrictamente creciente.

Para todo real a > 0, b > 0, c > 0 y n, se tiene que:

$$a = b^{\log_b a}$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_c a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b(1/a) = -\log_b a$$
$$\log_b a = \frac{1}{\log_a b}$$
$$a^{\log_b n} = n^{\log_b a}$$

Funciones y notaciones comunes

Para ln(1+x) cuando |x|<1, $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots$

Para x >-1, $\frac{x}{1+x} \leq \ln(1+x) \leq x$ cumpliendose la igualdad para x=0.

Una función es polilogaritmicamente limitada si $f(n)=\lg^{O(1)} n$. Para cualquier constante a>0, $\lg^b n=o(n^a)$, por lo que cualquier función polinomial positiva crece más rapidamente que cualquier función polilogarítmica.

6.- Factoriales: Para $n \geq 0$, n!=1x2x3x...n. Un límite superior débil es $n! \leq n^n$, ya que cada término es el producto de a lo sumo n. Usando la aproximación de Stirling se prueba:

$n! = o(n^n)$ $n! = \omega(2^n)$ $\lg(n!) = \Theta(n \lg n)$

$$\sqrt{2\mathrm{n}\pi}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\mathrm{n}\pi}\left(\frac{n}{e}\right)^{n+(1/12n)}$$

Funciones y notaciones comunes

7.- Números de Fibonacci: $F_0=0$, $F_1=1$, $F_i = F_{i-1} + F_{i+1}$ para $i \geq 2$.
Se prueba por inducción que

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

donde ϕ es la sección dorada igual a $(1+\sqrt{5})/2$. Los números de Fibonacci crecen exponencialmente.

8.- Interación de la función logarítmica: La notación lg*n denotará la función logarítmica iterada, la cual se define como:

$$\lg^* n = \min\{ i \geq 0 : \lg^{(i)} n \leq 1\}$$

donde $\lg^{(i)} n$ denota la función logaritmo aplicada **i** veces y definida como:

$$\lg^{(i)} n = \begin{cases} n \text{ si } i = 0, \\ \lg(\lg^{(i-1)} n) \text{ si } i \rangle 0 \text{ y } \lg^{(i-1)} n \rangle 0, \\ \text{no definida si } i \rangle 0 \text{ y } \lg^{(i-1)} n \leq 0 \text{ o } \lg^{(i-1)} n \text{ es indefinida.} \end{cases}$$

Esta función crece muy lentamente. $\lg^* 2 = 1$, $\lg^* 4 = 2$, $\lg^* 16 = 3$
$\lg^* 65536 = 4$, $\lg^* (2^{65536}) = 5$.

Sumatorias

Dada una secuencia a_1, a_2 ,... de números, la sumatoria finita $a_1+a_2+...+a_n$ se escribe como: $\sum_{k=1}^{n} a_k$ Si n=0, el valor de la sumatoria se define como 0. Si n no es entero, se asume un límite superior de $\lfloor n \rfloor$. Asimismo, si la suma comienza con k=x y x no es entero se asume como valor inicial de la suma $\lfloor x \rfloor$.

Dada una secuencia a_1, a_2 ,... de números, la sumatoria infinita $a_1+a_2+...$ se escribe como: $\sum_{k=1}^{\infty} a_k$ que es interpretado como: $\lim_{n\to\infty} \sum_{k=1}^{n} a_k$

Si el límite no existe, la serie **diverge**, de lo contrario **converge**.

Una serie que **converge absolutamente** es una serie $\sum_{k=1}^{\infty} a_k$ para la cual $\sum_{k=1}^{\infty} |a_k|$ también converge.

1.- Linealidad: Para cualquier real **c** y unas secuencias finitas a_1, a_2 ,... y b_1, b_2 ,... $\sum_{k=1}^{n}(ca_k+b_k)=c\sum_{k=1}^{n}a_k+\sum_{k=1}^{n}b_k$ propiedad que mantienen las series convergentes infinitas.

Sumatorias

La propiedad lineal se aplica a la notación asintótica, por ejemplo:

$\sum_{k=1}^{n}\Theta(f(k)) = \Theta\left(\sum_{k=1}^{n} f(k)\right)$ donde la notación Θ de la parte izquierda se aplica a la variable **k** y la de la derecha a **n**.

2.- Series aritméticas: $\displaystyle\sum_{k=1}^{n} k = 1+2+\cdots+n = \tfrac{1}{2}n(n+1) = \Theta\left(n^2\right)$

3.- Series geométricas o exponenciales: Para $x \neq 1$

$$\sum_{k=0}^{n} x^k = 1+x+x^2+\cdots+x^n = \frac{x^{n+1}-1}{x-1} \text{ cuando } |x|\langle 1 \quad \sum_{k=0}^{\infty} x^k = \tfrac{1}{1-x}$$

4.- Series harmónicas: Para n enteros positivos, el n-ésimo harmónico es $H_n = 1+\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{4}+\cdots+\tfrac{1}{n} = \sum_{k=1}^{n}\tfrac{1}{k} = \ln n + O(1)$

Para cualquier secuencia $a_1, a_2, \ldots, a_n$

$$\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0, \text{ asimismo } \sum_{k=0}^{n-1}(a_k - a_{k+1}) = a_0 - a_n$$

$$\text{Por ejemplo, } \sum_{k=1}^{n-1}\tfrac{1}{k(k+1)} = \sum_{k=1}^{n-1}\left(\tfrac{1}{k} - \tfrac{1}{k+1}\right) = 1 - \tfrac{1}{n}$$

5.- Productoria: El producto finito $a_1 a_2 \ldots a_n$ $\displaystyle\prod_{k=1}^{n} a_k$ si n=0 el valor del producto es 1. Para convertir de sumatoria a productoria se usa la identidad, $\displaystyle\lg\left(\prod_{k=1}^{n} a_k\right) = \sum_{k=1}^{n}\lg a_k$

Sumatorias

Los métodos más usados para describir el tiempo de corrida de los algoritmos se basan en encontrar una limitación para la suma de una secuencia mediante:

1.- Inducción matemática: Ejemplo: la serie geométrica dada es $\sum_{k=0}^{n+1} 3^k = \sum_{k=0}^{n} 3^k + 3^{n+1} \le c3^n + 3^{n+1} = \left(\frac{1}{3}+\frac{1}{c}\right)c3^{n+1} \le c3^{n+1}$ y está limitada con $O(3^n)$

2.- Limitando los términos: Normalmente se escoge el término más grande para limitar a los otros, en general si $a_{max}=\max_{1\le k\le n} a_k$ entonces $\sum_{k=0}^{n} a_k \le n a_{max}$. Este método no es muy recomendable si la serie puede ser limitada por una serie geométrica, en cuyo caso se prefiere el límite de esta última.

3.- Fisión: Se expresa la sumatoria en dos o más series separando el rango del índice y encontrando la limitación de cada serie.

En general, $\sum_{k=0}^{n} a_k = \sum_{k=0}^{k_0-1} a_k + \sum_{k=k_0}^{n} a_k = \Theta(1) + \sum_{k=k_0}^{n} a_k$

$$Ejemplo: \sum_{k=1}^{n} k = \sum_{k=1}^{n/2} k + \sum_{k=n/2+1}^{n} k \ge \sum_{k=1}^{n/2} 0 + \sum_{k=n/2+1}^{n} (n/2) \ge (n/2)^2 = \Omega(n^2)$$

Recurrencias

• Cuando un algoritmo contiene llamadas a él mismo (recursivo), su tiempo de ejecución se expresa normalmente con una **recurrencia**.
• Una **recurrencia** es una ecuación o desigualdad que describe una función en términos de sus valores para sus entradas más pequeñas.
• Para encontrar los límites asintóticos (Θ, O) se presentan tres mé-todos:

» Método por substitución
» Método por iteración
» Método maestro

Método por substitución: Se enuncia una forma de solución y se prueba por inducción que dicha solución asociada a una constante es buena.
Este método se aplica cuando se puede enunciar la solución posible de la recurrencia.

Recurrencias

Ejm: Encontrar el límite superior de la recurrencia $T(n)=2T(\lfloor n/2 \rfloor)+n$
Suponemos que la solución es $T(n)=O(n \lg n)$. Se probará que
$T(n) \leq cn \lg n$ para una constante apropiada $c>0$. Se inicia asumiendo que el límite se da para $\lfloor n/2 \rfloor$. Substituyendo,

$T(n) \leq 2(c \lfloor n/2 \rfloor \lg (\lfloor n/2 \rfloor)) + n$

$T(n) \leq c\, n \lg (n/2) + n$

$T(n) = c\, n \lg n - c\, n \lg 2 + n$

$T(n) = c\, n \lg n - c\, n + n$

$T(n) \leq c\, n \lg n$ se cumple para $c \geq 1$

La inducción matemática requiere que se pruebe la solución encontrada para condiciones límites. Si se asume $T(1) = 1$, no se puede escoger c grande, ya que $T(1) \leq c\, 1 \lg 1 = 0$. Esta dificultad se resuelve tomando $n =2$ o $n= 3$. Así, $T(2)=4$ y $T(3)=5$ y la prueba inductiva $T(n) \leq c\, n \lg n$ para $c \geq 2$ es suficiente.

Recurrencias

- Se pueden usar cambios de variables para simplificar, ejemplo:

$T(n)=2T(\lfloor \sqrt{n} \rfloor) + \lg n$. Se hace $m=\lg n$ y $T(2^m) = 2T(2^{m/2})+m$ se hace $S(m) = T(2^m)$ y se obtiene $S(m) = 2\,S(m/2) + m$ que es parecida a la anterior por lo cual $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$

Método iterativo: Se expande la recurrencia y se expresa como una sumatoria de términos que depende solo de n y de condiciones iniciales. Ejemplo:

$T(n) = 3\,T(\lfloor n/4 \rfloor) + n$

$T(n) = n + 3\,(\lfloor n/4 \rfloor + 3\,T(\lfloor n/16 \rfloor))$

$T(n) = n + 3\,(\lfloor n/4 \rfloor + 3\,(\lfloor n/16 \rfloor + 3\,T(\lfloor n/64 \rfloor)))$

$T(n) = n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27\,T(\lfloor n/64 \rfloor)$

$T(n) = n + 3\,n/4 + 9\,n/16 + 27\,n/64 + \ldots + 3^{\log_4 n}\,\Theta(1)$

$T(n) \le n\sum_{i=٠}^{\infty}\left(\frac{٣}{٤}\right)^{i} + \Theta\left(n^{\log_{٤} ٣}\right)$

$T(n) = 4n + o(n) = O(n)$

Recurrencias

Método maestro: Se usa para resolver recurrencias del tipo $T(n) = a\,T(n/b) + f(n)$, donde $f(n)$ es una función asintóticamente positiva y las constantes $a \geq 1$, $b > 1$. Ella expresa que el problema se divide en a subproblemas de tamaño n/b. Los a subproblemas se resuelven recursivamente en tiempo T(n/b). El costo de dividir el problema y combinar sus soluciones es descrito por f(n). Normal-mente en estos casos no se consideran los techos y pisos.

Teorema maestro: Sean $a \geq 1$, $b > 1$ constantes, f(n) una función y T(n) una recurrencia para números enteros no negativos, $T(n) = a\,T(n/b) + f(n)$, donde n/b puede ser $\lfloor n/b \rfloor$ o $\lceil n/b \rceil$ entonces T(n) está limitado asintóticamente según:

1.- Si $f(n) = O(n^{\log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_b a})$

2.- Si $f(n) = \Theta(n^{\log_b a})$, entonces $T(n) = \Theta(n^{\log_b a} \lg n)$

3.- Si $f(n) = \Omega(n^{\log_b a+\epsilon})$ para alguna constante $\epsilon > 0$, y si $a\,f(n/b) \leq c f(n)$ para alguna constante $c < 1$ y n suficientemente grande, entonces $T(n) = \Theta(f(n))$.

Evaluación de la eficiencia

- La mejor técnica para diferenciar la eficiencia de los algoritmos es el estudio de los órdenes de complejidad.
- El orden de complejidad se expresa generalmente en términos de la cantidad de datos procesados por el programa, denominada N.
- Este N puede ser el tamaño dado o estimado.

Ejemplo 4: Un algoritmo que procese un vector V(N) tendrá un orden de complejidad de N, ya que si N crece, en esa misma proporción crece el orden de complejidad de él.

- La cantidad de tiempo de procesamiento de un algoritmo (T(n)), por lo general viene dado en función de N, y puede expresarse en base a los casos típicos de ese N, caso promedio A(n), o en base a casos extremos no deseables, como el peor de los casos W(n).

Notación O

- El orden de complejidad se define como una función que **domina** la ecuación que expresa en forma exacta el tiempo de ejecución del programa.

    g(x) domina a f(x), si dada una constante C cualquiera

    $C*g(x) \geq f(x)\ \forall x$

- g(x) domina asintóticamente a f(x), si g(x) domina a f(x) para los valores muy grandes de x.

    $f(N) = N^2 + 5N + 100$ $\qquad$ $g(N) = N^2$

    entonces g(N) domina a f(N)

- El orden de complejidad se denota con una o mayúscula, O(g(N))

    $O(N^2)$, O(N)

- Un orden O(N) indica que el tiempo de ejecución decrece suave-mente en proporción al decrecimiento de N.

Notación O

- Aunque dos algoritmos tengan el mismo orden O(N), ellos pueden tener diferentes tiempos de ejecución para iguales valores de N.
- Reglas para determinar el orden de complejidad:

  1.- $O(C*g) = O(g)$

  2.- $O(f * g) = O(f) * O(g)$ $\qquad$ $O(f/g) = O(f) / O(g)$

  3.- $O(f+g)$ = función dominante entre $O(f)$ y $O(g)$

  Ejemplos: $O(2456 * N) = O(N)$

  $O((20 * N) * N) = O(20 * N) * O(N) = O(N^2)$

- Algunas funciones de dominación más comunes:

N^b domina a $N!$	$N!$ domina a b^N
b^N domina a c^N si $b \geq c$	b^N domina a N^a si $a \geq 0$
N^n domina a N^m si $n \geq m$	N domina a $\log_a N$ si $a \geq 1$
$\log_a N$ domina a $\log_b N$ si $b \geq a \geq 1$	$\log_a N$ domina a 1 si $a \geq 1$

Notación O

- Un algoritmo de O(1) tiene complejidad constante y por ello son los más eficientes y los preferidos.
- La mayoría de los programas tienen complejidad polinomial $O(N^a)$, donde N es la variable y a es una constante mayor que 1.

  $O(N)$, $O(N^2)$, $O(N^3)$, $O(\log N)$
- La **complejidad no polinomial** (NP) es aquella que tiene un orden mayor que la polinomial.

  Ejm: La complejidad exponencial $O(a^N)$
- Nombres de las más usadas:

log N	complejidad logarítmica ($\log_2 N \equiv \lg N$)
N	complejidad lineal
N^2	complejidad cuadrática
N^3	complejidad cúbica
2^N	complejidad exponencial

Notación O

Comparación entre diferentes complejidades

N	logN	NlogN	N^2	N^3	2^N	3^N
1	0	0	1	1	2	3
2	1	2	4	8	4	9
4	2	8	16	64	16	81
8	3	24	64	512	256	6.561
16	4	64	256	4.096	65.536	43.046.721
32	5	160	1.024	32.768	4.294.967.296	??
64	6	384	4.096	262.144	*	??
128	7	896	16.384	2.097.152	**	??

* el número de instrucciones que puede ejecutar un supercomputador de 1 GFLOP en 500 años.
** sería 500 billones de veces la edad del universo (20 billones de años) en nanosegundos.

- Los algoritmos sin lazos y sin recursión tienen complejidad constante
- La determinación del orden de complejidad de un algoritmo se inicia por los lazos y las recursiones. Los lazos anidados tendrán complejidad poínómica.

Notación O

- Correspondencia entre la estructura de programación y el orden de complejidad.

Estructura	Orden
Secuencial (S)	O(1)
S1 S2	La función dominante entre O(S1) y O(S2)
Si (Condición) entonces S1 sino S2 fsi	La función dominante entre O(S1), O(S2) y O(Condición), en el peor de los casos
[S1] i = 1, N	O(N * S1)

Ejemplo: [[m(i,j) = 0] j = 1, N] i = 1, N tiene $O(N^2)$

[a = a + b_i] i = 3,8 tiene O(1)

Análisis probabilístico

- Paradoja del cumpleaños
  - ¿Cuántas personas deben haber en una sala para que exista la posibilidad de que dos de ellas cumplan años el mismo día?

k: # de personas en la sala

n: # de días de un año = 365 (no se consideran los bisiestos)

Bi: el día del i-ésimo cumpleaños, con $1 \leq Bi \leq n$

$Pr\{Bi=r\}=1/n$ para $i = 1,.., k$ y $r = 1,.., n$ (cumpleaños uniformemente distribuidos)

Si los cumpleaños son independientes, $Pr\{Bi = r \text{ y } Bj = r\} = 1/n * 1/n=1/n^2$

Que el cumpleaños de ambos sea el mismo día,

$Pr\{Bi = Bj\} = \sum_{r=1}^{n} Pr\{Bi = r \text{ y } Bj = r\} = \sum_{r=1}^{n} (1/n^2) = 1/n$

Análisis probabilístico

- Paradoja del cumpleaños: ¿Cuántas personas deben haber en una sala para que exista la posibilidad de que dos de ellas cumplan años el mismo día?

  $\forall$ Pares (i, j) de las K personas en la sala, se define la variable aleatoria Xij

  Xij = 1, si i y j tienen el mismo cumpleaños y 0, si no

  Pr{Bi=r}=1/n

  La esperanza E[Xij] = 1 * 1/n + 0 * (1 – 1/n) = 1/n

  Suma de la esperanza de los pares $\sum_{i=2}^{k}\sum_{j=1}^{i-1} E[X_{ij}] = \binom{k}{2} 1/n = (k(k-1))/2n$

  Cuando k(k-1) ≥ 2n, el número esperado de pares de cumpleaños es al menos 1. Si se tienen al menos √2n personas en una sala, se espera tener 2 personas que cumplan años el mismo día. Si n=365, k=28. $\Theta(\sqrt{n})$