Tema 1 - Análisis de la eficiencia de los Algoritmos

César González Ferreras

Departamento de Informática
Universidad de Valladolid

Curso 2022/23

Contenidos

Contenidos

Algoritmo

- Un **algoritmo** es un conjunto de instrucciones que permite realizar un cálculo o resolver un problema, bien sea a mano o, más frecuentemente, en una máquina.
- La notación más habitual para describir los algoritmos es el **pseudocódigo**.
- También se puede utilizar directamente algún lenguaje de programación concreto.
- El nombre de algoritmo proviene del matemático persa Al-Juarismi

Medida de algoritmos

- **Objetivo**: medir la *eficiencia* de los algoritmos.
- **Eficiencia**: *coste* en *recursos* que utiliza el algoritmo para llevar a cabo su tarea.
- Nos permite comparar algoritmos que realizan la misma tarea para determinar cuál es más eficiente.
- Los **recursos** más importantes son:
  - *Tiempo de ejecución*.
  - *Espacio de memoria*.

Medida de algoritmos - Ejemplo

Búsqueda secuencial

BusqSec($\mathbf{v}, n, x\,;\, p$)

$p \leftarrow 1$
while $(p <= n) \wedge (v_p \neq x)$ **do**
    $p \leftarrow p + 1$
end while
if $v_p \neq x$ **then**
    $p \leftarrow 0$
end if

- Busca un valor igual a x en el vector $v_{1\ldots n}$.
- Si existe uno (o varios) valores iguales a x, devuelve en p la posición de uno cualquiera de ellos.
- Si no existe ninguno igual devuelve $p = 0$.

Medida de algoritmos - Dificultades

- Existen varias dificultades para proporcionar el tiempo concreto que tarda un algoritmo (por ejemplo, 23 μseg.)

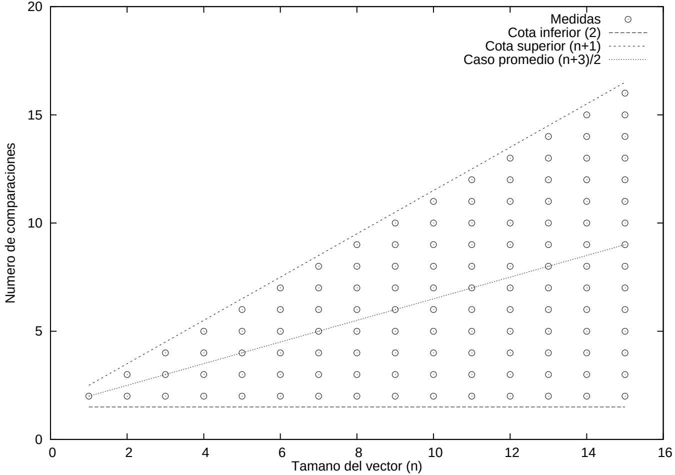
1. **Dependencia del procesador**: Cada procesador tarda un tiempo distinto en ejecutar cada tipo de instrucción.
   - **Solución**: expresar el resultado como el número de operaciones elementales que se realizan.
2. **Dependencia con el tamaño de la entrada**: según sea la entrada del algoritmo, más complejo es el proceso y más más operaciones hay que realizar.
   - En el ejemplo podemos ver que el número de operaciones depende del valor de n.
   - **Solución**: expresar el número de operaciones como una función del tamaño.
3. **Dependencia con los valores de la entrada**:
   - En el ejemplo, el algoritmo hace más o menos operaciones dependiendo de los valores de x y del vector.
   - **Solución**: *renunciar* a obtener una medida precisa y dividir el análisis en *casos*.

Medida de algoritmos

- **Peor caso**:
  - Valores de las entradas que, para un tamaño fijo, provocan que el algoritmo tenga el mayor coste posible.
  - Establece una cota superior al número de operaciones que realiza el algoritmo.
  - En el ejemplo, el peor caso se dará cuando el valor no se encuentra en el vector, y el número de comparaciones en las que interviene el vector será de $n + 1$.
- **Mejor caso**:
  - Valores de las entradas que, para un tamaño fijo, provocan que el algoritmo tenga el menor coste posible.
  - Establece una cota inferior al número de operaciones que realiza el algoritmo.
  - En el ejemplo, el mejor caso se dará cuando el valor buscado se encuentra en la primera posición del vector. Esto da lugar a que se realicen 2 comparaciones.

Medida de algoritmos

- Búsqueda secuencial: número de comparaciones en función de varios tamaños del vector y para distintos valores de las entradas.



Análisis del caso promedio

- Determinar una función dependiente del tamaño de la entrada que indique el número medio esperado de operaciones para una determinada distribución de las entradas de ese tamaño.
- Es necesario definir un **modelo estadístico** de los posibles valores de las entradas del algoritmo.
- Coste promedio, $\widehat{c}(n)$:

$$\widehat{c}(n) = \sum_{\forall e \in \mathcal{E}_n} \mathrm{prob}(e) \cdot \mathrm{coste}(e) \tag{1}$$

  - $\mathcal{E}_n$ espacio de distintas entradas de tamaño n del algoritmo.
  - $\mathrm{prob}(e)$ probabilidad de ocurrencia de la entrada $e \in \mathcal{E}_n$.
  - $\mathrm{coste}(e)$ coste (expresado en número de operaciones) de la entrada $e \in \mathcal{E}_n$.
- Si todas las entradas son *equiprobables* (*distribución uniforme*):

$$\widehat{c}(n) = \frac{1}{|\mathcal{E}_n|} \sum_{\forall e \in \mathcal{E}_n} \mathrm{coste}(e) \tag{2}$$

  - $|\mathcal{E}_n|$ indica el número de entradas posibles con tamaño n.

Análisis del caso promedio

- Ejemplo: búsqueda secuencial.
- Modelo de entradas: todos los valores del vector son distintos. Por tanto, podemos que restringir el estudio a las posibles permutaciones del conjunto $\{1 \dots n\}$.
- Dividimos el estudio en dos casos:
  - Búsqueda fallida: es equivalente al peor caso, y el número de operaciones es siempre el mismo ($n + 1$ comparaciones).
  - Búsqueda exitosa, se debe cumplir que $x \in \{1 \dots n\}$, lo que restringe el valor de x.
- El espacio $\mathcal{E}_n$ del problema consiste en las $n!$ permutaciones junto con los n posibles valores de x.
- Todas las entradas son equiprobables, por lo que se puede aplicar la ecuación (2).
- Tenemos que $|\mathcal{E}_n| = n \cdot n!$.

Análisis del caso promedio

- En la siguiente tabla se enumeran las 18 entradas (pares $\overline{\mathbf{v}} - x$) de $\mathcal{E}_3$ junto con el número de comparaciones de cada entrada:

$\overline{\mathbf{v}}$	x	coste	$\overline{\mathbf{v}}$	x	coste
$[1,2,3]$	1	2	$[2,3,1]$	1	4
$[1,2,3]$	2	3	$[2,3,1]$	2	2
$[1,2,3]$	3	4	$[2,3,1]$	3	3
$[1,3,2]$	1	2	$[3,1,2]$	1	3
$[1,3,2]$	2	4	$[3,1,2]$	2	4
$[1,3,2]$	3	3	$[3,1,2]$	3	2
$[2,1,3]$	1	3	$[3,2,1]$	1	4
$[2,1,3]$	2	2	$[3,2,1]$	2	3
$[2,1,3]$	3	4	$[3,2,1]$	3	2

- El número promedio de comparaciones será igual a la suma de los costes dividido por el número de entradas posibles:
$\widehat{c}(3) = \frac{54}{18} = 3.$

Análisis del caso promedio

- Una forma más sencilla de realizar el cálculo es hallar el número de entradas que dan lugar a un determinado coste y realizar una suma sobre los costes posibles.
- Supongamos que los costes para un determinado tamaño n pueden tomar valores del conjunto $\mathcal{C}_n$ (el análisis del mejor y peor caso nos proporciona éste conjunto).
- El espacio de entradas se puede particionar en los subconjuntos de entradas con un mismo coste: $\mathcal{E}_n = \bigcup_{\forall c \in \mathcal{C}_n} \mathcal{E}_n^{(c)}$ donde $\mathcal{E}_n^{(c)}$ representa el subconjunto de entradas cuyo coste es c.
- Podemos entonces modificar la ecuación (2) de la siguiente manera:

$$\widehat{c}(n) = \frac{1}{|\mathcal{E}_n|} \sum_{\forall c \in \mathcal{C}_n} |\mathcal{E}_n^{(c)}| \cdot c \tag{3}$$

Análisis del caso promedio

- En el problema tenemos que los costes posibles son 2,3 y 4, por lo que $\mathcal{C} = \{2, 3, 4\}$
- El conjunto de posibles entradas se puede particionar de la siguiente manera de acuerdo al coste:

$\mathcal{E}_3^{(2)}$	$\mathcal{E}_3^{(3)}$	$\mathcal{E}_3^{(4)}$
$([1,2,3],1)$	$([1,2,3],2)$	$([1,2,3],3)$
$([1,3,2],1)$	$([1,3,2],3)$	$([1,3,2],2)$
$([2,1,3],2)$	$([2,1,3],1)$	$([2,1,3],3)$
$([2,3,1],2)$	$([2,3,1],3)$	$([2,3,1],1)$
$([3,1,2],3)$	$([3,1,2],1)$	$([3,1,2],2)$
$([3,2,1],3)$	$([3,2,1],2)$	$([3,2,1],1)$

- Aplicando la ecuación (3) tenemos $\widehat{c}(3) = \frac{1}{18}(6 \cdot 2 + 6 \cdot 3 + 6 \cdot 4) = 3$.
- Podemos apreciar que el número de entradas con igual coste es el mismo, 6, en las tres particiones:
  - Los costes son *equiprobables*, existe la misma probabilidad de que una entrada cualquiera tenga un determinado coste.

Análisis del caso promedio

- Si los costes son equiprobables, se cumple que $\forall_{c \in \mathcal{C}_n} |\mathcal{E}_n^{(c)}| = u$, donde u es un valor constante.
- Como las particiones son disjuntas, se cumple que $|\mathcal{E}_n| = \sum_{\forall_{c \in \mathcal{C}_n}} |\mathcal{E}_n^{(c)}| = |\mathcal{C}_n| \cdot u$.
- Aplicando esta relación en la ecuación (3) obtenemos:

$$\widehat{c}(n) = \frac{1}{|\mathcal{C}_n|} \sum_{\forall c \in \mathcal{C}_n} c \tag{4}$$

- En nuestro problema $\mathcal{C}_n = \{2, \ldots, n+1\}$ y el coste promedio sería:

$$\widehat{c}(n) = \frac{1}{n} \sum_{c=2}^{n+1} c = \frac{n+3}{2}$$

Análisis en tiempo amortizado

- **Análisis en tiempo amortizado**: coste de una *secuencia* de ejecuciones. *Promedio temporal*.
- Hay ejecuciones en las que el coste es mucho mayor de lo normal, pero estas ejecuciones *garantizan* que un cierto número de las siguientes ejecuciones tendrán un coste normal.
- Calcularemos el coste medio de una secuencia de k ejecuciones del algoritmo, cada una de ellas con un coste $c_i(n)$, y extrapolaremos para valores de $k \to \infty$:

$$c_{amort}(n,k) = \frac{\sum_{i=1}^{k} c_i(n)}{k} \tag{5}$$

$$c_{amort}(n) = \lim_{k\to\infty} c_{amort}(n,k) \tag{6}$$

- Si la ecuación anterior da como resultado ∞, indica que no se ha tenido en cuenta la dependencia respecto a n.

Análisis en tiempo amortizado - Ejemplo

Inserción al final

InsFinal($\mathbf{v}, m, n, x\, ;\, \mathbf{w}$)

if $n = m$ **then**
    $\mathbf{w} \leftarrow$ *Crear vector de tamaño* $m' > m$
    for $i = 1, \ldots, n$ **do**
        $w_i \leftarrow v_i$
    end for
else
    $\mathbf{w} \leftarrow \mathbf{v}$
end if
$w_{n+1} \leftarrow x$

- Inserta el valor x en la posición $n + 1$ del vector **v**.
- El vector **v** tiene la *capacidad* de almacenar m elementos.
- El vector contiene n elementos en las posiciones $v_{1\ldots n}$.
- Si $n = m$ se crea un nuevo vector de mayor tamaño.
- En **w** se devuelve la referencia al vector resultante.

Análisis en tiempo amortizado - Ejemplo

- Contar las asignaciones que se realizan a elementos del vector.
- **Mejor caso** se da si $n < m$ y el coste es 1 asignación.
- **Peor caso** se da cuando $n = m$ y el coste será $n + 1$ asignaciones.
- Para conocer el comportamiento del algoritmo hay que hacer un Análisis en tiempo amortizado:
  - Coste promedio tras realizar un gran número de secuencias de ejecuciones del algoritmo.
- Examinaremos dos casos para saber cuál es el mejor:
  - Caso 1: $m' = m + 10$
  - Caso 2: $m' = 2m$

Análisis en tiempo amortizado - Caso 1

- **Caso 1**: $m' = m + 10$
- Una secuencia de inserciones consistirá en una serie de ciclos.
- Cada ciclo:
  - 1 inserción que amplia el vector de un tamaño n a un tamaño $n + 10$, con un coste de $n + 1$ asignaciones.
  - 9 inserciones sin ampliación, con un coste de 1 asignación.
- Supondremos que la secuencia de operaciones comienza con el vector vacío ($n = 0$) y un tamaño inicial de 9 ($m = 9$).
- Si realizamos k inserciones, el número de ciclos completos será igual a $q = k \operatorname{div} 10$ y restaran $r = k \bmod 10$ inserciones normales.
- El número de elementos del vector será igual al de operaciones: $k = n = 10 \cdot q + r$.

Análisis en tiempo amortizado - Caso 1

Ciclo	k y n	Secuencia	Coste	m
1	$1 \dots 9$	9 ejec. $\times$ coste 1	9	9
1	10	1 ejec. $\times$ coste 10	10	19
2	$11 \dots 19$	9 ejec. $\times$ coste 1	9	19
2	20	1 ejec. $\times$ coste 20	20	29
3	$21 \dots 29$	9 ejec. $\times$ coste 1	9	29
3	30	1 ejec. $\times$ coste 30	30	39
$\vdots$	$\dots$	$\dots$	$\dots$	$\dots$
j	$10(j-1)+1 \dots 10j-1$	9 ejec. $\times$ coste 1	9	$10j-1$
j	$10j$	1 ejec. $\times$ coste $10j$	$10j$	$10j+9$
$\vdots$	$\dots$	$\dots$	$\dots$	$\dots$
q	$10(q-1)+1 \dots 10q-1$	9 ejec. $\times$ coste 1	9	$10q-1$
q	$10q$	1 ejec. $\times$ coste $10q$	$10q$	$10q+9$
	$10q+1 \dots 10q+r$	r ejec. $\times$ coste 1	r	$10q+9$

Análisis en tiempo amortizado - Caso 1

- La suma de los costes es:

$$c_{amort}(n,k) = \frac{\sum_{i=1}^{k} c_i(n)}{k} = \frac{r + \sum_{j=1}^{q}(10j+9)}{k} = \frac{5q^2 + 14q + r}{k}$$

- Si sustituimos $q = \frac{k-r}{10}$

$$c_{amort}(n,k) = \frac{\sum_{i=1}^{k} c_i(n)}{k} = \frac{k}{20} + \frac{14-r}{10} + \frac{r^2-8r}{20k}$$

- Si hacemos tender k a infinito nos encontramos que el límite tiende a infinito también:
    - El coste depende del tamaño de la entrada.

Análisis en tiempo amortizado - Caso 1

- Si sustituimos en los términos lineales en k el hecho de que $n = k$:

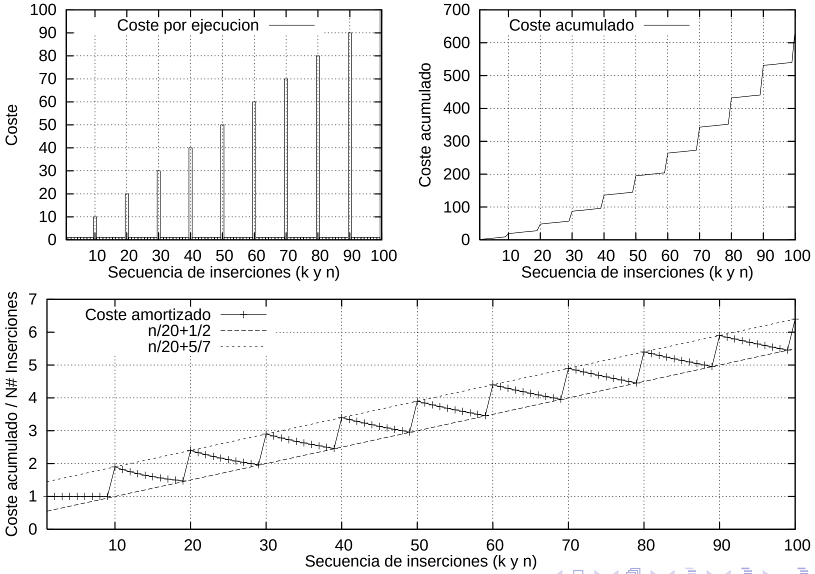
$$c_{amort}(n,k) = \frac{n}{20} + \frac{14-r}{10} + \frac{r^2-8r}{20k}$$
$$c_{amort}(n) = \lim_{k\to\infty} c_{amort}(n,k) = \frac{n}{20} + \frac{14-r}{10}$$

- Como $0 \leq r \leq 9$ obtenemos las cotas:

$$\frac{n}{20} + \frac{1}{2} \leq c_{amort}(n) \leq \frac{n}{20} + \frac{7}{5}$$

- El punto importante es que **el coste amortizado depende del tamaño de la entrada,** n.

Análisis en tiempo amortizado - Caso 1



Análisis en tiempo amortizado - Caso 2

- **Caso 2**: $m' = 2m$.
- Cada ciclo:
  - 1 una inserción que amplia el tamaño del vector de n a $2n$, con un coste de $n + 1$.
  - $n - 1$ inserciones de coste 1.
- Supondremos un vector vacío ($n = 0$) de capacidad inicial $m = m_0$.
- Realizamos q ciclos de ampliación, faltando al último ciclo r inserciones para llenar el vector.

Análisis en tiempo amortizado - Caso 2

Ciclo	k y n	Secuencia	Coste	m
—	$1 \ldots m_0$	m_0 ejec. $\times$ coste 1	m_0	m_0
1	$m_0 + 1$	1 ejec. $\times$ coste $m_0 + 1$	$m_0 + 1$	$2m_0$
1	$m_0 + 2 \ldots 2m_0$	$m_0 - 1$ ejec. $\times$ coste 1	$m_0 - 1$	$2m_0$
2	$2m_0 + 1$	1 ejec. $\times$ coste $2m_0 + 1$	$2m_0 + 1$	$4m_0$
2	$2m_0 + 2 \ldots 4m_0$	$2m_0 - 1$ ejec. $\times$ coste 1	$2m_0 - 1$	$4m_0$
3	$4m_0 + 1$	1 ejec. $\times$ coste $4m_0 + 1$	$4m_0 + 1$	$8m_0$
3	$4m_0 + 2 \ldots 8m_0$	$4m_0 - 1$ ejec. $\times$ coste 1	$4m_0 - 1$	$8m_0$
⋮	...	...	...	...
j	$2^{j-1} m_0 + 1$	1 ejec. $\times$ coste $2^{j-1} m_0 + 1$	$2^{j-1} m_0 + 1$	$2^j m_0$
j	$2^{j-1} m_0 + 2 \ldots 2^j m_0$	$2^{j-1} m_0 - 1$ ejec. $\times$ coste 1	$2^{j-1} m_0 - 1$	$2^j m_0$
⋮	...	...	...	...
q	$2^{q-1} m_0 + 1$	1 ejec. $\times$ coste $2^{q-1} m_0 + 1$	$2^{q-1} m_0 + 1$	$2^q m_0$
q	$2^{q-1} m_0 + 2 \ldots 2^q m_0 - r$	$2^{q-1} m_0 - r - 1$ ejec. $\times$ coste 1	$2^{q-1} m_0 - r - 1$	$2^q m_0$

Análisis en tiempo amortizado - Caso 2

- El número total de inserciones será de $k = 2^q m_0 - r$.
- El valor de r está acotado por $0 \leq r \leq 2^{q-1} m_0 - 1$.
- Podemos ver que el ciclo j tiene un coste de $2^j m_0$ asignaciones.
- La suma de los costes dividida por el número de operaciones será:

$$c_{amort}(n,k) = \frac{\sum_{i=1}^{k} c_i(n)}{k} = \frac{m_0 - r + \sum_{j=1}^{q} (2^j m_0)}{k}$$

$$c_{amort}(n,k) = \frac{2^{q+1} m_0 - m_0 - r}{k}$$

- Como $k = 2^q m_0 - r$ podemos despejar $2^q = \frac{k+r}{m_0}$ y obtenemos:

$$c_{amort}(n,k) = 2 + \frac{r}{k} - \frac{m_0}{k}$$

Análisis en tiempo amortizado - Caso 2

- El valor de r puede variar desde $r = 0$ hasta $r = 2^{q-1}m_0 - 1$.
- Si $r = 2^{q-1}m_0 - 1$, entonces $r = k - 2$, ya que $k = 2^q m_0 - r = 2^q m_0 - 2^{q-1}m_0 + 1 = 2^{q-1}m_0 + 1$.
- Podemos acotar $c_{amort}(n, k)$ por los valores:

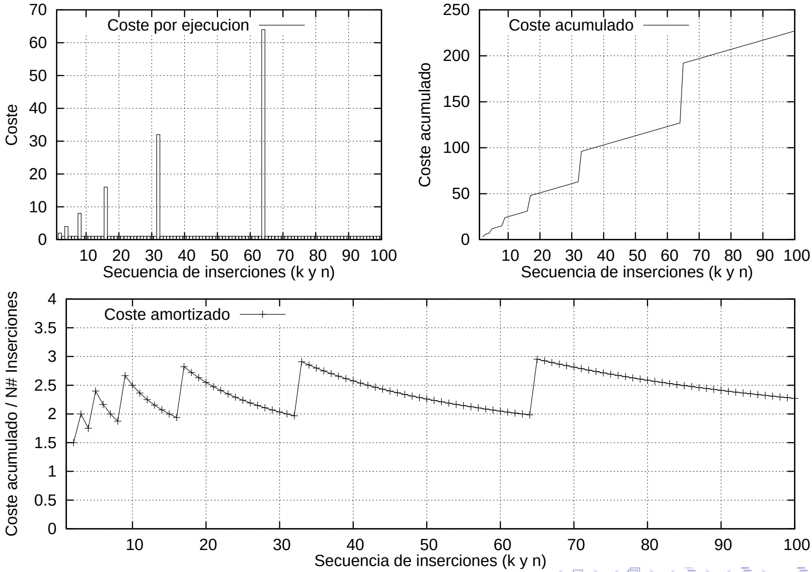
$$2 - \frac{m_0}{k} \leq c_{amort}(n, k) \leq 3 - \frac{m_0 + 2}{k}$$

- Y haciendo que k tienda a infinito:

$$2 \leq c_{amort}(n) \leq 3$$

- El promedio de asignaciones es un valor constante tras una secuencia lo bastante grande de inserciones.
- **El coste de las inserciones normales *compensa* al coste de las inserciones que amplían el vector**.

Análisis en tiempo amortizado - Caso 2

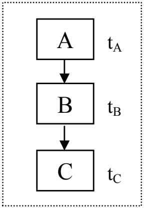


Análisis en tiempo amortizado

- Un ejemplo de la utilidad del análisis en tiempo amortizado es la clase `ArrayList<E>` de Java:
  - http://docs.oracle.com/javase/10/docs/api/java/util/ArrayList.html

Reglas para medir algoritmos - Tiempo

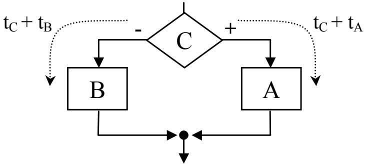
- Para medir *el tiempo* hay que *contar* las operaciones de un determinado tipo que realiza el algoritmos para un determinado tamaño de la entrada.
- **Operaciones elementales**:
  - Aquellas que el procesador puede realizar en un tiempo constante.
  - Las operaciones básicas son la asignación y la evaluación de expresiones, ya sean de tipo aritmético o lógico.
  - Estas son las operaciones que se cuentan en el algoritmo.
- **Estructura secuencial**:
  - El tiempo empleado en ejecutar una secuencia de sentencias es igual a la *suma* de los tiempos empleados en ejecutar cada sentencia por separado.



$t_A + t_B + t_C$

Reglas para medir algoritmos - Tiempo

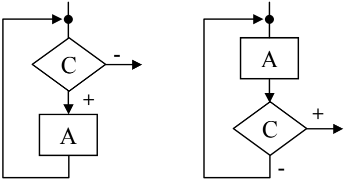
- **Estructura alternativa**:
  - El tiempo empleado es igual al tiempo necesario para evaluar la condición más el tiempo necesario en ejecutar la rama indicada por el valor de la condición.
  - Para alternativas dobles podemos obtener dos tiempos distintos.
  - En el análisis de mejor caso se escoge la alternativa menos costosa.
  - En el análisis de peor caso la alternativa más costosa (siempre teniendo en cuenta la *totalidad* del algoritmo).



  - Para los análisis de tiempo promedio y tiempo amortizado se debe calcular la probabilidad de que una entrada cualquiera haga cierta o falsa la condición.
  - Si denominamos p la probabilidad de que una entrada haga que la condición sea cierta, el coste promedio será $t_c + p \cdot t_a + (1 - p) \cdot t_b$.

Reglas para medir algoritmos - Tiempo

- **Estructura iterativa**:
  - Se debe sumar los tiempos de cada una de las iteraciones ejecutadas (incluyendo el coste de las comparaciones de control del bucle).
  - Los bucles con salida al principio realizan una comparación extra.



$t_C + \Sigma(t_A + t_C)$ $\Sigma(t_A + t_C)$

  - Si t_a es constante para todas las iteraciones, en lugar de calcular un sumatorio se multiplica t_a por el número total de iteraciones.
  - Generalmente el número total de iteraciones depende de la entrada del algoritmo.
  - En esos casos, al igual que con la estructura alternativa, se divide el análisis en casos.

Reglas para medir algoritmos - Tiempo

- **Llamadas a subprogramas**:
  - Cualquier subprograma que vaya a ser utilizado por el algoritmo debe medirse previamente.
  - Esto puede plantear problemas en el caso de algoritmos recursivos, donde es necesario aplicar relaciones de recurrencia.

Reglas para medir algoritmos - Tiempo - Uso de un barómetro

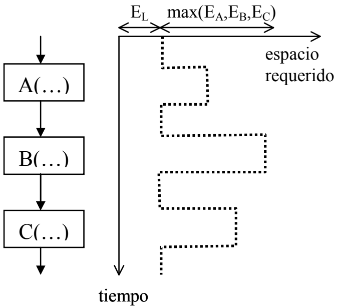
- El análisis de muchos algoritmos se **simplifica** de forma significativa cuando se emplea una instrucción como barómetro.
- Una **“instrucción barómetro”** es aquella que se ejecuta por lo menos con tanta frecuencia como cualquier otra instrucción del algoritmo.
  - Si alguna instrucción se ejecuta como mucho un número constante de veces más que el barómetro su contribución quedará absorbida en la notación asintótica
  - El tiempo requerido por el algoritmo completo será del orden exacto del número de veces que se ejecuta la instrucción barómetro.
- Se desprecian los tiempos exactos que requieren ciertas instrucciones y se evita tener que introducir constantes que serán descartadas al emplear la notación asintótica.
- Por ejemplo, en los algoritmos de ordenación, se consideran solamente:
  - Comparaciones y asignaciones en las que intervengan elementos del vector

Reglas para medir algoritmos - Memoria

- El *espacio necesario* para la ejecución de un algoritmo vendrá dado por el punto de **máxima ocupación de memoria**.
- **Variables locales**:
  - Espacio de almacenamiento constante a lo largo de la ejecución del algoritmo.
  - Se incluyen los parámetros de entrada y salida, excepto los parámetros pasados por variable.
- **Variables dinámicas y objetos**:
  - La creación de variables dinámicas (sentencia new de Pascal o malloc en C) y objetos reservan memoria.
  - Su destrucción (sentencia dispose en Pascal, free en C, pérdida de referencias en Eiffel) libera la memoria reservada.

Reglas para medir algoritmos - Memoria

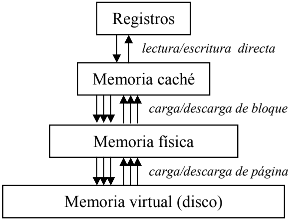
- **Llamadas a subprogramas**:
  - Reservan cierta cantidad fija de memoria (punto y contexto de retorno).
  - Hay que sumar el espacio utilizado por el subprograma.
  - Cuando termina la llamada, esta memoria *se libera*.
  - Ejemplo: llamadas a los subprogramas A,B y C.



  - El espacio ocupado por las variables locales se denomina E_l.
  - Para calcular el espacio utilizado por las llamadas a los subprogramas se halla el valor máximo.

Otros factores que influyen en la medida

- **Localización**:
  - *Uso localizado de memoria*: si tras cada acceso a memoria existe una gran probabilidad de que el siguiente acceso sea a una posición cercana.



  - Jerarquía de memoria:
    - Los datos se tratan en el nivel de los registros del procesador.
    - Si el dato no está en un registro, se busca en la caché.
    - Si no está en la caché, se intercambia el bloque de direcciones de la memoria física donde reside la posición con uno de los bloques de la caché.
    - De este modo se carga no solo el dato que necesitamos sino varios de los datos que residen en las direcciones anterior y posterior a él.
    - **Si el siguiente acceso es a una dirección cercana, existen muchas probabilidades de que el dato esté ya cargado en la caché**.
    - Si los datos manejados son muy grandes, parte de ellos residirán en disco, y se realizarán operaciones de intercambio de páginas con la memoria principal.

Otros factores que influyen en la medida

- **Paralelización**:
  - Si un algoritmo se puede dividir en tareas que se pueden resolver independientemente, entonces es posible dedicar varios procesadores.
- **Gestión de memoria**:
  - Ciertas estructuras de datos *dinámicas* puedan cambiar de tamaño en ejecución (el ejemplo típico son las cadenas de texto).
  - Es preciso realizar *compactación de memoria* o *recolección de basura*.
  - Esto puede generar problemas de rendimiento en ciertas aplicaciones.
  - La gestión de memoria suele depender del compilador, del entorno de programación y del lenguaje de programación utilizado.

Contenidos

Notación asintótica

- La notación asintótica sirve para expresar la eficiencia de los algoritmos:
  - De una manera más compacta.
  - Ocultando detalles que no son relevantes.
  - Mostrando la forma en que crece el coste en función del tamaño de la entrada.
- **Cota Superior** Dada una función $f(n)$ donde $n \in \mathbb{Z}^+$ la notación $O(f(n))$ representa al conjunto de funciones que cumplen la siguiente propiedad:

$$g(n) \in O(f(n)) \iff \exists\, n_0 \in \mathbb{Z}^+, c \in \mathbb{R}^+ : \forall\, n > n_0 : g(n) \leq c \cdot f(n) \tag{7}$$

Esto significa que $f(n)$ es una cota superior asintótica (cuando n es grande) de las funciones que pertenecen al conjunto $O(f(n))$, sin tener en cuenta constantes de proporcionalidad. La propiedad anterior se puede enunciar también de la siguiente manera:

$$g(n) \in O(f(n)) \iff \lim_{n \to \infty} \frac{g(n)}{f(n)} = \begin{cases} 0 \\ \text{cte} \end{cases} \tag{8}$$

Notación asintótica

- **Cota Inferior** Dada una función $f(n)$ donde $n \in \mathbb{Z}^+$ la notación $\Omega(f(n))$ representa al conjunto de funciones que cumplen la siguiente propiedad:

$$g(n) \in \Omega(f(n)) \iff \exists\, n_0 \in \mathbb{Z}^+, c \in \mathbb{R}^+ : \forall\, n > n_0 : g(n) \geq c \cdot f(n) \tag{9}$$

La propiedad anterior se puede enunciar también de la siguiente manera:

$$g(n) \in \Omega(f(n)) \iff \lim_{n\to\infty} \frac{g(n)}{f(n)} = \begin{cases} \infty \\ \text{cte} \end{cases} \tag{10}$$

- **Cota Estricta** Dada una función $f(n)$ donde $n \in \mathbb{Z}^+$ la notación $\Theta(f(n))$ representa al conjunto de funciones que cumplen la siguiente propiedad:

$$g(n) \in \Theta(f(n)) \iff g(n) \in O(f(n)) \wedge g(n) \in \Omega(f(n)) \tag{11}$$

La propiedad anterior se puede enunciar también de la siguiente manera:

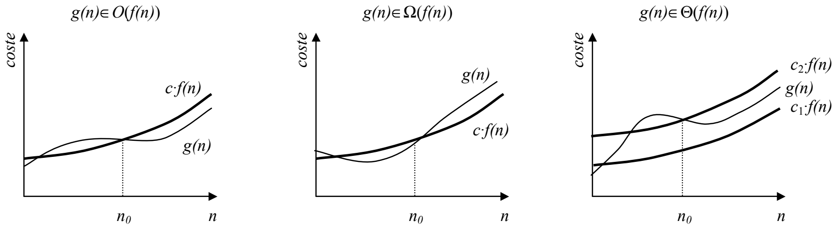
$$g(n) \in \Theta(f(n)) \iff \lim_{n\to\infty} \frac{g(n)}{f(n)} = \text{cte} \tag{12}$$

También se puede expresar como:

$$\Theta(f(n)) = O(f(n)) \bigcap \Omega(f(n)) \tag{13}$$

Notación asintótica

- En el siguiente gráfico se muestra de forma esquemática el comportamiento de las funciones que pertenecen a cada conjunto de cota:



- No importa el comportamiento de las funciones cuando el tamaño de la entrada es pequeño.
- Tampoco se tiene en cuenta los factores constantes que multiplican a las funciones.

Notación asintótica

- Si no hablamos de un conjunto de cotas concreto se sobreentiende que nos referimos a las cotas superiores.
- De manera informal se puede decir que las funciones de cota representan:
  - $O(f(n))$ al conjunto de las funciones que *no crecen más rápido* que $f(n)$.
  - $\Omega(f(n))$ al conjunto de las funciones que *crecen igual o más rápido* que $f(n)$.
  - $\Theta(f(n))$ al conjunto de las funciones que *crecen al mismo ritmo* que $f(n)$.
- El trabajar con conjuntos de cotas en lugar de funciones simplifica el análisis de algoritmos porque:
  - Lo que importa es el comportamiento en el límite, es decir cuando $n \to \infty$.
  - No se tiene en cuenta las constantes de proporcionalidad.
  - Sólo se tiene en cuenta el término con mayor crecimiento de una expresión.

Notación asintótica

- Propiedades básicas de los conjuntos de cotas:

$$
\begin{aligned}
f(n) &\in O(f(n)) \\
O\Big(O(f(n))\Big) &= O(f(n)) \\
O(c \cdot f(n)) &= O(f(n)), \quad c > 0 \\
O(f(n) + g(n)) &= f(n) + O(g(n)) = g(n) + O(f(n)) = O(f(n)) + O(g(n)) \\
O(f(n) + g(n)) &= O\Big(\max\,(f(n), g(n))\Big) \\
O(f(n) \cdot g(n)) &= f(n) \cdot O(g(n)) = g(n) \cdot O(f(n)) = O(f(n)) \cdot O(g(n))
\end{aligned}
$$

- Sólo indicamos las fórmulas para las cotas superiores, pero estas expresiones son válidas también para cotas inferiores y estrictas.

Notación asintótica

- No es necesario indicar la base del logaritmo:

$$\log_b n = \frac{\log_c n}{\log_c b} \tag{14}$$

$$\Theta(\log n) \equiv \Theta(\ln n) = \Theta(\lg n) = \Theta(\log_b n) \tag{15}$$

- Ecuaciones útiles para el análisis de bucles:

$$O\left(\sum_{i=1}^{n} i^a\right) = \sum_{i=1}^{n} O(i^a) = O(n^{a+1}) \tag{16}$$

$$\sum_{i=1}^{n} i^a = \frac{n^{a+1}}{a+1} + O(n^a) \tag{17}$$

$$\sum_{i=1}^{n} \frac{1}{i} = \ln(n) + O(1) \tag{18}$$

Notación asintótica

- Se debe tener cierto cuidado cuando tenemos una resta de términos.
- Si los términos que se restan tienen un crecimiento menor que los términos que se suman, se pueden despreciar sin problemas. Por ejemplo: $O(n^2 - n) = O(n^2)$ y $\Omega(n^2 - n) = \Omega(n^2)$.
- En la resta de términos con igual crecimiento, como en el caso $O(n^2) - O(n^2)$, sólo podemos despreciar el término que se resta si sabemos que su constante de proporcionalidad es menor que la del término que se suma.
- Si no tenemos cuidado podemos llegar a paradojas como la siguiente:

$$O(n^2) = O(n^2 + n^3 - n^3) = O(n^2 + n^3) - O(n^3) = O(n^3) - O(n^3) = O(n^3)$$

Notación asintótica

- Utilizaremos la notación asintótica para indicar la eficiencia de un algoritmo con el nivel de detalle que deseemos.
- Por ejemplo, podemos indicar la eficiencia del algoritmo de ordenación por inserción de la siguiente forma:
  - Indicando que el algoritmo tiene un orden $O(n^2)$.
  - Indicando que el algoritmo tiene órdenes $O(n^2)$ y $\Omega(n)$.
  - Indicando que el peor caso es $\Theta(n^2)$, el mejor caso es $\Theta(n)$ y el caso promedio es $\Theta(n^2)$.
  - Indicando que el número de comparaciones es $\frac{n^2}{2} + O(n)$ en el peor caso, $n + O(1)$ en el mejor y $\frac{n^2}{4} + O(n)$ en promedio.

Notación asintótica - Comparar algoritmos

- Para comparar algoritmos hay que fijarse en el orden asintótico.
- Si los **órdenes son distintos**:
  - El algoritmo con menor orden será el más eficiente.
  - Aunque desconozcamos las constantes de proporcionalidad sabemos que para un tamaño de n lo bastante grande la función de mayor crecimiento dominará a la otra,
  - Por ejemplo $\Theta(n^2)$ y $\Theta(n \lg n)$. En este caso $\Theta(n \lg n)$ es más eficiente.
- Si los **ordenes son iguales**:
  - Es necesario conocer las constantes de proporcionalidad para establecer que algoritmo es más adecuado.
  - Por ejemplo tanto la ordenación por inserción como la ordenación burbuja tienen casos promedios $\Theta(n^2)$.
  - Sabiendo que la ordenación por inserción tiene un caso promedio $\frac{1}{4}n^2 + O(n)$ y la ordenación burbuja un caso promedio $\frac{3}{4}n^2 + O(n \lg n)$ ya podemos apreciar que el primero es un algoritmo más eficiente.

Notación asintótica - Cotas ajustadas

- El algoritmo de ordenación por inserción sabemos que es $O(n^2)$ y $\Omega(n)$.
- Sin embargo, también podríamos decir que es un algoritmo $O(n^3)$ y $\Omega(1)$ y sería cierto.
- La segunda forma proporcionamos un información *menos útil* sobre la eficiencia del algoritmo.
- Cuando se utiliza la notación asintótica se presupone que proporcionamos las **cotas más ajustadas posibles** sobre la eficiencia del algoritmo.
- Si decimos que un algoritmo es $O(n^2)$, se sobreentiende que existen entradas que tienen un coste proporcional a n^2.

Notación asintótica - Órdenes de magnitud

- Según su crecimiento, se puede clasificar a las funciones en las siguientes categorías:
  - **Coste constante**: coste independiente del tamaño de la entrada se denota por el conjunto de cotas $O(1)$.
  - **Coste sub-polinómico**: Una función tiene coste sub-polinómico cuando pertenece a $O(n^{\alpha})$ donde α es una constante positiva de valor arbitrariamente pequeño. Un ejemplo son las funciones logarítmicas.
  - **Coste polinómico**: Una función tiene coste polinómico si pertenece a $O(n^{\alpha})$ donde α es una constante positiva de valor arbitrariamente grande.
  - **Coste no polinómico**: Cualquier función que no tenga coste polinómico pertenece a ésta categoría. Un ejemplo son las funciones exponenciales y el factorial.
- Funciones de cota más comunes ordenadas de menor a mayor crecimiento (a es un valor mayor que 1):
  $O(1) \subset O(\lg n) \subset O(n^{1/a}) \subset O(n) \subset O(n \lg n) \subset O(n\sqrt{n}) \subset O(n^a) \subset O(2^n) \subset O(3^n) \subset O(n!) \subset O(n^n)$

Notación asintótica - Órdenes de magnitud

- Tiempo necesario para resolver un problema de un determinado tamaño suponiendo que si $n = 1$ el tiempo que se tarda es de 1 ns.

n	$O(1)$	$O(\lg n)$	$O(\sqrt{n})$	$O(n)$	$O(n \lg n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
1	1 ns.	1,0 ns.	1,0 ns.	1,0 ns.	1,0 ns.	1,0 ns.	1,0 ns.	1,0 ns.
5	1 ns.	2,6 ns.	2,2 ns.	5,0 ns.	12,9 ns.	25,0 ns.	16,0 ns.	0,1 μs.
10	1 ns.	3,5 ns.	3,2 ns.	10,0 ns.	34,6 ns.	0,1 μs.	0,5 μs.	3,6 ms.
20	1 ns.	4,4 ns.	4,5 ns.	20,0 ns.	87,8 ns.	0,4 μs.	0,5 ms.	77,1 años
30	1 ns.	5,0 ns.	5,5 ns.	30,0 ns.	0,1 μs.	0,9 μs.	0,5 s.	$8 \cdot 10^{15}$ años
40	1 ns.	5,4 ns.	6,3 ns.	40,0 ns.	0,2 μs.	1,6 μs.	9,2 min.	—
50	1 ns.	5,7 ns.	7,0 ns.	50,0 ns.	0,3 μs.	2,5 μs.	6,5 días	—
100	1 ns.	6,7 ns.	10,0 ns.	0,1 μs.	0,7 μs.	10,0 μs.	$2 \cdot 10^{13}$ años	—
1.000	1 ns.	9,9 ns.	31,6 ns.	1,0 μs.	10,0 μs.	1,0 ms.	—	—
10.000	1 ns.	13,3 ns.	0,1 μs.	10,0 μs.	0,1 ms.	0,1 s.	—	—
100.000	1 ns.	16,6 ns.	0,3 μs.	0,1 ms.	1,7 ms.	10,0 s.	—	—
1.000.000	1 ns.	19,9 ns.	1,0 μs.	1,0 ms.	19,9 ms.	16,7 min.	—	—
10.000.000	1 ns.	23,3 ns.	3,2 μs.	10,0 ms.	0,2 s.	1,2 días	—	—
100.000.000	1 ns.	26,6 ns.	10,0 μs.	0,1 s.	2,7 s.	115,7 días	—	—
1.000.000.000	1 ns.	29,9 ns.	31,6 μs.	1,0 s.	29,9 s.	31,7 años	—	—

Notación asintótica - Tamaño de la entrada

- Es preciso indicar claramente el significado de los parámetros de las funciones de cota, que representan el tamaño de la entrada.
- *Estructuras de datos multidimensionales*. Por ejemplo, la multiplicación de dos matrices cuadradas puede ser:
  - $\Theta(n^{3/2})$ si n representa el número de elementos de la matriz.
  - $\Theta(n^3)$ si n representa el número de filas de la matriz.
- *Números de rango ilimitado*:
  - Si los valores numéricos de entrada están limitados se pueden representar mediante tipos de datos predefinidos.
  - En caso contrario, serían en realidad *vectores de dígitos* y el tamaño de la entrada estaría representado por el número de dígitos necesarios para representar el valor.
  - Por ejemplo, el problema de elevar al cuadrado un entero n:
    - Orden $O(1)$ si el valor n se puede almacenar en una variable de tipo entero.
    - Orden $O(\lg^2 n)$ si el valor de n no está limitado.
- No siempre se tiene un único parámetro para representar el tamaño de la entrada:
  - Por ejemplo, el problema de buscar una subcadena de longitud m caracteres en un texto de n caracteres tiene un orden $O(n \cdot m)$.

Contenidos

Relaciones de recurrencia

- Problemas a la hora de medir algoritmos recursivos.
- Ejemplo del cálculo del factorial, $n!$
  - $0! = 1! = 1$.
  - $n! = n \cdot (n-1)!$

Cálculo recursivo del factorial

función Fact(n: integer): integer

1: **if** $n < 2$ **then**
2:     $Fact \leftarrow 1$
3: **else**
4:     $Fact \leftarrow n \cdot \text{Fact}(n-1)$
5: **end if**

Relaciones de recurrencia- Resolución de ecuaciones homogéneas

- Denominaremos ecuación homgénea a una ecuación en recurrencia con el siguiente aspecto:

$$a_0 F_n + a_1 F_{n-1} + \ldots + a_k F_{n-k} = 0$$

- Buscaremos soluciones de la forma $F_n = r^n$ donde r es una constante:

$$a_0 r^n + a_1 r^{n-1} + \ldots + a_k r^{n-k} = 0$$
$$r^{n-k}(a_0 r^k + a_1 r^{k-1} + \ldots + a_k) = 0$$

- Dejamos aparte la solución trivial de $r = 0$.
- Denominaremos ecuación característica a:

$$a_0 r^k + a_1 r^{k-1} + \ldots + a_k = 0$$

- Si ese polinomio tiene k raíces distintas, entonces una solución de la ecuación homogénea es:

$$F_n = c_1 r_1^n + c_2 r_2^n + \ldots + c_k r_k^n$$

- Los c_i se encontrarán a partir de las condiciones iniciales.

Relaciones de recurrencia- Resolución de ecuaciones homogéneas

- **Ejemplo**: los números de Fibonacci:

$$F(n) := \begin{cases} 0 & si\ n = 0 \\ 1 & si\ n = 1 \\ F(n-1) + F(n-2) & si\ n > 1 \end{cases}$$

- Es posible encontrar un fórmula cerrada para los números de Fibonacci.
- Hay que resolver la ecuación de recurrencia: $F_n = F_{n-1} + F_{n-2}$.
- La ecuación característica es $r^2 - r - 1 = 0$
- Por tanto las raíces son $r_1 = \frac{1+\sqrt{5}}{2}$ y $r_2 = \frac{1-\sqrt{5}}{2}$.
- Las soluciones de la ecuación homogénea serán de la forma:

$$F_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

- Sabiendo que $F_0 = 0$ y $F_1 = 1$ obtenemos que $c_1 = \frac{1}{\sqrt{5}}$ y que $c_2 = -\frac{1}{\sqrt{5}}$.

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right)$$

Relaciones de recurrencia- Resolución de ecuaciones homogéneas

- Si no todas las raíces son distintas, y r_i es una raíz con multiplicidad m, entonces $r_i^n, nr_i^n, n^2 r_i^n, \ldots, n^{m-1} r_i^n$ formarán parte de la base.
- **Ejemplo**:

$$F_0 = 0, F_1 = 1, F_2 = 2$$

$$F_n = 5F_{n-1} - 8F_{n-2} + 4F_{n-3} \; si \; n \geq 2$$

- La ecuación característica es $r^3 - 5r^2 + 8r - 4 = 0$.
- Entonces $(r-1)(r-2)^2 = 0$.
- Por tanto:

$$F_n = c_1 1^n + c_2 2^n + c_3 n 2^n$$

- Utilizando las condiciones iniciales:

$$F_n = -2 + 2^{n+1} - n2^{n-1}$$

Relaciones de recurrencia- Resolución de ecuaciones no homogéneas

- Ejemplo

$$F_n - 2F_{n-1} = 3^n$$

- Sustituyendo n por $n + 1$.

$$F_{n+1} - 2F_n = 3^{n+1}$$

- Multiplicando la original por 3:

$$3F_n - 6F_{n-1} = 3^{n+1}$$

- Restando las dos anteriores:

$$F_{n+1} - 5F_n + 6F_{n-1} = 0$$

- Que es una ecuación homogénea con ecuación característica:

$$0 = r^2 - 5r + 6 = (r - 2)(r - 3)$$

Teorema maestro de las recurrencias

- Si la relación es del tipo:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \tag{19}$$

  - a y b son valores enteros que cumplen $a \geq 1$, $b \geq 2$
  - $f(n) \in \Theta(n^k)$
  - k es un valor entero que cumple $k \geq 0$:

- Entonces la solución pertenece al siguiente orden asintótico:

$$T(n) \in \Theta(n^k) \qquad \text{si } a < b^k \tag{20}$$

$$T(n) \in \Theta(n^k \log n) \qquad \text{si } a = b^k \tag{21}$$

$$T(n) \in \Theta(n^{\log_b a}) \qquad \text{si } a > b^k \tag{22}$$

Relaciones de recurrencia - Cotas inferiores y superiores

```
function f(n: integer) : integer;
begin
  if n < 1 then
    f := 1
  else
    f := f(n/3)+f(n/6)+f(n/9)
end;
```

- Coste en tiempo (operaciones suma):
  $F(n) = F(\frac{n}{3}) + F(\frac{n}{6}) + F(\frac{n}{9}) + O(1),\ si\ n > 1$
  - $G(n) = 3G(\frac{n}{3}) + O(1)$; $G(n) \in \Theta(n)$
  - $H(n) = 3H(\frac{n}{9}) + O(1)$; $H(n) \in \Theta(\sqrt{n})$
  - $H(n) < F(n) < G(n)$
  - $F(n) \in \Omega(\sqrt{n})$ y $F(n) \in O(n)$
- Coste en memoria: $E(n) = E(\frac{n}{3}) + O(1)$.
  - $E(n) \in \Theta(\log n)$.

Relaciones de recurrencia - Cambio de variables

- A veces puede ser conveniente realizar un cambio de variable para convertir una relación de recurrencia en otra que se pueda resolver.
- Por ejemplo, si en la relación

$$T(n) = 2T(\sqrt{n}) + \lg n$$

- Definimos la variable $m = \lg n \rightarrow n = 2^m$ y hacemos el cambio

$$T(2^m) = 2T(2^{m/2}) + m$$

- Si definimos la función $S(m) = T(2^m)$ obtenemos una recurrencia que se puede resolver mediante el teorema maestro:

$$S(m) = 2S(\frac{m}{2}) + m \implies S(m) \in \Theta(m \lg m)$$

- Deshaciendo el cambio de variable y de función obtenemos:

$$T(n) \in \Theta(\lg n \lg \lg n)$$