Tema 3. Análisis de costes

`http://aulavirtual.uji.es`

José M. Badía, Begoña Martínez, Antonio Morales y José M. Sanchiz

{badia, bmartine, morales, sanchiz}@icc.uji.es

Estructuras de datos y de la información

Universitat Jaume I

Índice

Bibliografía

- (Nyhoff'06), Apartados 10.4 y 13.1
- G. Brassard y P. Bratley, *Algorithmique: Conception et Analyse*, Ed. Mason, (1987), Capítulos 1 y 2
- G. Martínez, Apuntes de la asignatura IS04, (2006), Tema 10
- A. Marzal, Apuntes de las asignaturas II04 e IG04, (2005), Tema 14
- L. Joyanes y I. Zahonero, *Estructuras de Datos. Algoritmos, abstracción y objetos*, Ed. McGrawHill, (1998), Capítulos 15 y 16

Objetivos

- Comprender la importancia del desarrollo de algoritmos eficientes.
- Conocer los conceptos relacionados con el análisis de costes.
- Saber utilizar la notación asintótica para el análisis de costes.
- Conocer los principales algoritmos cuadráticos de ordenación y saber analizar su coste.

1 Motivación

¿Por qué se buscan algoritmos eficientes?

Problema: Desarrollar un algoritmo que calcule el valor de x^{2^n}

```
1  float intento1 ( float x, int n) {
2     int aux;
3     float res;
4     res = 1;
5     aux = power(2,n);
6     for (int i=0; i<aux; i++)
7        res = res * x;
8     return res;
9  }
```

1 Motivación (II)

Duración de la primera versión

	Potencia de cálculo (prod/seg)		
n	$\mathbf{10^6}$	$\mathbf{10^9}$	$\mathbf{10^{12}}$
10	$0{,}001s.$	10^{-6}	10^{-9}
20	$1s.$	$0{,}001s.$	10^{-6}
50	$36\ años$	$13\ dias$	$19\ minutos$
100	$4 * 10^{16}\ años$	$4 * 10^{13}\ años$	$4 * 10^{10}\ años$

Edad del universo $\approx 1{,}5 * 10^{10}$ años

1 Motivación (III)

Solución eficiente

```
1  float intento2 ( float x , int n) {
2    float res;
3    res = x;
4    for (int i=0; i<n; i++)
5      res = res * res;
6    return res;
7  }
```

Coste: n productos

Duración de la segunda versión

Potencia de cálculo: 10^6 prod/seg

n	versión 1	versión 2
10	$0{,}001s.$	$10^{-5}s.$
20	$1s.$	$2*10^{-5}s.$
100	$4*10^{16}$ *años*	$10^{-4}s.$
3.600.000.000	*siempre*	1 *hora*

Coste exponencial vs. Coste lineal

2 Conceptos y definiciones

La ejecución de un algoritmo en una máquina supone el uso de una serie de recursos: espacio y tiempo.

- **Coste espacial:** Cantidad de memoria consumida. Datos.
- **Coste temporal:** Tiempo necesario para ejecutar el programa. Operaciones.

Ambos factores suelen entrar en conflicto:

$\Rightarrow$ Necesidad de establecer una prioridad o un compromiso.

2 Conceptos y definiciones (II)

Métodos de análisis

- **A posteriori** (o experimental): Se estudia el comportamiento de una implementación en un ordenador.

  Ejemplos:

  - Coste espacial: $320 Kbytes$
  - Coste temporal: $12,3 seg.$

- **A priori** (o teórico): Se determina matemáticamente el coste antes de implementar o ejecutar el algoritmo.

  Ejemplos:

  - Coste espacial: $3n - 2$
  - Coste temporal: $2 \log n$

2 Conceptos y definiciones (III)

Factores de que depende el coste

- Habilidad del programador.
- Lenguaje de programación y compilador utilizado.
- Ordenador sobre el que se implementa. Tipo y carga.

¿Son realmente importantes?

¿Es necesario implementar el algoritmo?

Es mejor realizar un análisis teórico independiente de la implementación.

2 Conceptos y definiciones (IV)

Coste de un algoritmo genérico

$$\sum_{i \in \{instrucciones\}} (n_i * t_i)$$

n_i: número de veces que se ejecuta i

t_i: tiempo de ejecución de i

- El factor n_i puede depender de:
  - **Talla:** Tamaño del problema
  - **Instancia:** Conjunto posible de datos de entrada del problema
- El factor t_i depende de:
  - Las características del ordenador

Ejemplos de talla e instancia

- **Problema:** Sumar dos vectores de reales.
  - **Talla:** Tamaño de los vectores.
  - **Instancia:** Cualquier par posible de vectores de reales.
- **Problema:** Buscar un elemento en una lista.
  - **Talla:** Longitud de la lista.
  - **Instancia:** Cualquier posible conjunto de valores en la lista y el valor buscado.
- **Problema:** Calcular el sumatorio de 1 a n.
  - **Talla:** Cantidad de elementos a sumar.
  - **Instancia:** El valor de n (coincide con la talla).

Coste temporal

Número de pasos de un programa expresado en función de la talla del problema.

¿Qué es un paso de programa?

Cualquier operación o conjunto de operaciones cuyo tiempo de ejecución no depende de la instancia ni de la talla del problema.

- Operaciones aritméticas.
- Comparaciones.
- ...

Al comparar hay que mantener siempre el mismo criterio.

3 Análisis del coste (II)

Suma de vectores

```
1  void sumavec ( float v[], float w[], int n) {
2     for (int i=0; i<n; i++)
3        v[i] = v[i] + w[i];
4  }
```

Paso: Suma en coma flotante.

Coste = n sumas (igual para todas las instancias posibles).

Búsqueda de un elemento en un vector ordenado

Primera aproximación

```
1  int buscalin ( float v[], int n, float buscado) {
2     int i = 0;
3     while ((i<n) && (v[i]<buscado))
4        i++;
5     if ( (i == n) || (v[i] > buscado) ) return -1;
6     else return i;
7  }
```

Paso: elementos comprobados $\equiv$ comparaciones $\equiv$ incrementos.

3 Análisis del coste (IV)

El coste depende de la instancia.

3 casos básicos:

- **Caso mejor**: Aquellas instancias que, para cada talla, hacen que se ejecuten el menor número de pasos.

  ¡Cuidado! No es la instancia de talla más pequeña posible.

- **Caso peor**: Idem con el mayor número de pasos.

- **Caso promedio**: Aquellas instancias que para cada talla hacen que se ejecuten el número de pasos promedio del algoritmo.

$$\sum_{i \in \{instancias\}} numop_i * prob_i$$

Coste de la búsqueda lineal

- **Caso mejor:** El elemento buscado es el primero.

$$Coste = 1$$

- **Caso peor:** el elemento buscado es mayor que el último.

$$Coste = talla = n$$

- **Caso promedio:** depende de la probabilidad de cada instancia.

$$Coste = \sum_{i=0}^{n-1} (i+1)\frac{1}{n} = \frac{n+1}{2}$$

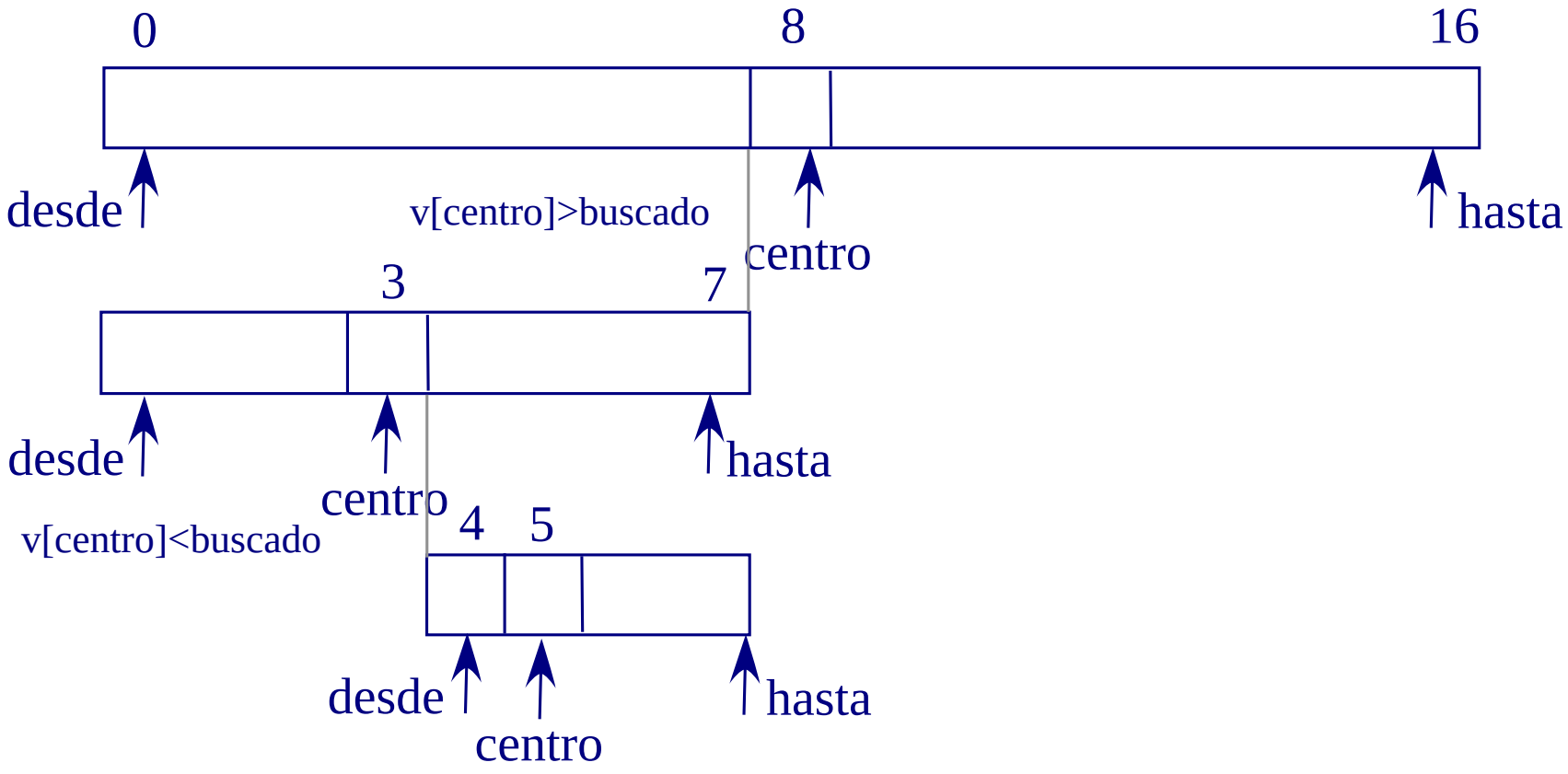
3 Análisis del coste (VI)

Búsqueda dicotómica

```
1  int buscadic ( float v[], int n, float buscado) {
2    int desde = 0, hasta = n-1, centro;
3    bool encontrado = false;
4    while ((desde <= hasta) && (!encontrado)) {
5      centro = (desde + hasta)/2;
6      if (v[centro] == buscado) encontrado = true;
7      else if (v[centro] > buscado) hasta = centro - 1;
8      else desde = centro + 1;
9    }
10   if encontrado return centro;
11   else return -1;
12 }
```

3 Análisis del coste (VII)

Búsqueda dicotómica



El tamaño del espacio de búsqueda se divide aproximadamente por dos en cada iteración.

3 Análisis del coste (VIII)

Paso: intentos (uno por iteración del bucle while).

- **Caso mejor:** el elemento está en la mitad del vector: $Coste = 1$.
- **Caso peor:** el elemento no está o es el último buscado: $Coste = \log_2 n$.

iteración		**tamaño**	
1	...	$n/2$	
2	...	$n/2^2$	Última iteración
...	...	...	$\equiv$
k	...	$n/2^k$	un solo elemento en el vector

$$\frac{n}{2^k} = 1 \Rightarrow \log_2 \frac{n}{2^k} = log_2 1 = 0 \Rightarrow k = \log_2 n$$

3 Análisis del coste (IX)

Búsqueda dicotómica vs. lineal

n (talla)	lineal (n)	binario ($\log_2 n$)	lineal/binario
10	10	4	2,5
10^2	10^2	7	14,3
10^3	10^3	10	100
10^4	10^4	14	714,3
10^5	10^5	17	5882,4
10^6	10^6	20	50,000

4 Notación asintótica

Problema: Multiplicar una matriz triangular superior por un vector.

```
1  void matvec(float a[][MAX], float b[], float c[], int n) {
2    for (int i=0; i<n; i++) {
3      c[i] = 0.0;
4      for (int j=i; j<n; j++)
5        c[i] = c[i] + a[i][j] * b[j];
6    }
7  }
```

Paso: operación en coma flotante (suma y producto).

Coste del producto matriz vector

i		pasos bucle j
0	...	$2n$
1	...	$2(n-1)$
2	...	$2(n-2)$
...	...	...
n-1	...	2

$$Coste = 2n + 2(n-1) + \ldots + 2 = \sum_{i=1}^{n} 2i = n(n+1) = n^2 + n$$

4 Notación asintótica (III)

Influencia de los distintos factores

n	n^2	n^2+n	$2n^2$
1	1	2	2
10	100	110	200
100	10.000	10.100	20.000
1.000	1.000.000	1.001.000	2.000.000
10.000	100.000.000	100.010.000	200.000.000

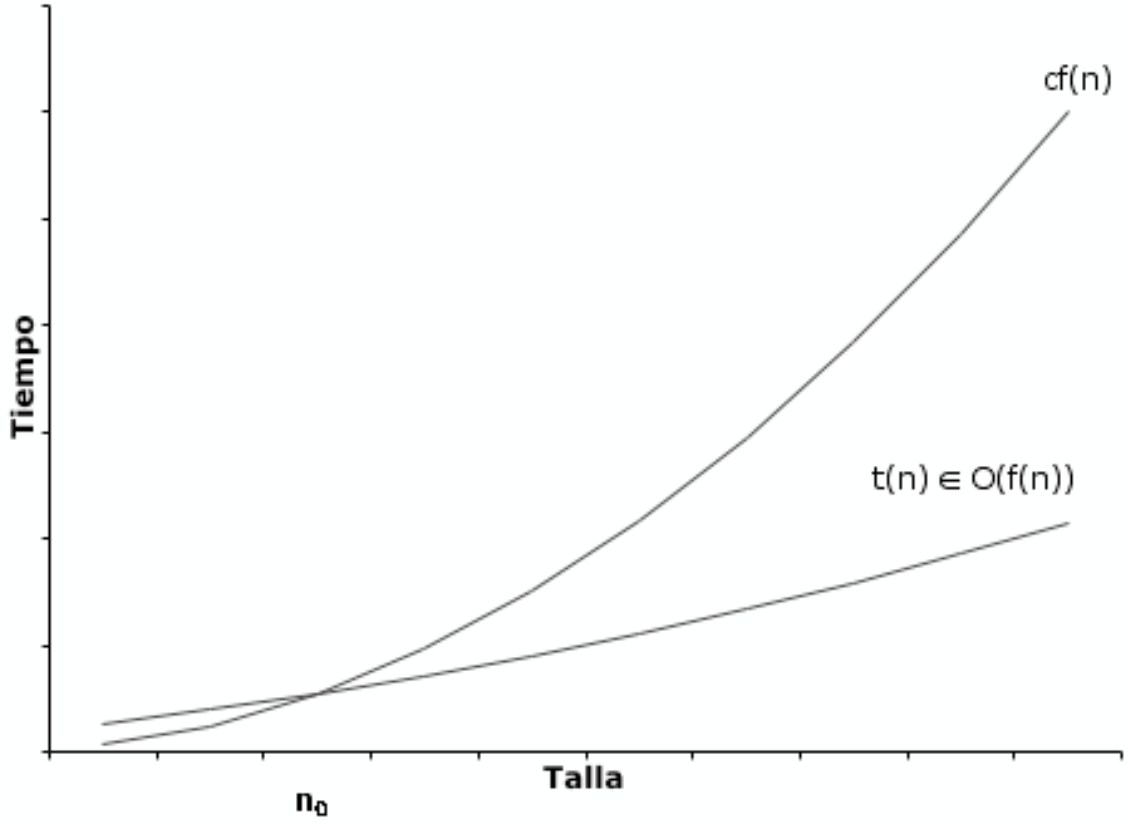
n^2+n se aproxima **asintóticamente** a n^2

4 Notación asintótica (IV)

Notación O

Sea $f : \mathbb{N} \to \mathbb{R}^+$, definimos la siguiente familia de funciones

$$O(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 \;\; t(n) \leq cf(n)\}$$



4 Notación asintótica (V)

El coste de un algoritmo es $O(f(n))$ si puede expresarse como una función que pertenece a esa familia.

Ejemplos:

- $3n+2$ es $O(n)$

$$\exists c = 4, \exists n_0 = 2 : \forall n > 2 \quad 3n+2 \leq 4n$$

- $2n^2+10n$ es $O(n^2)$

$$\exists c = 3, \exists n_0 = 10 : \forall n > 10 \quad 2n^2+10n \leq 3n^2$$

- $6*2^n+n^2$ es $O(2^n)$

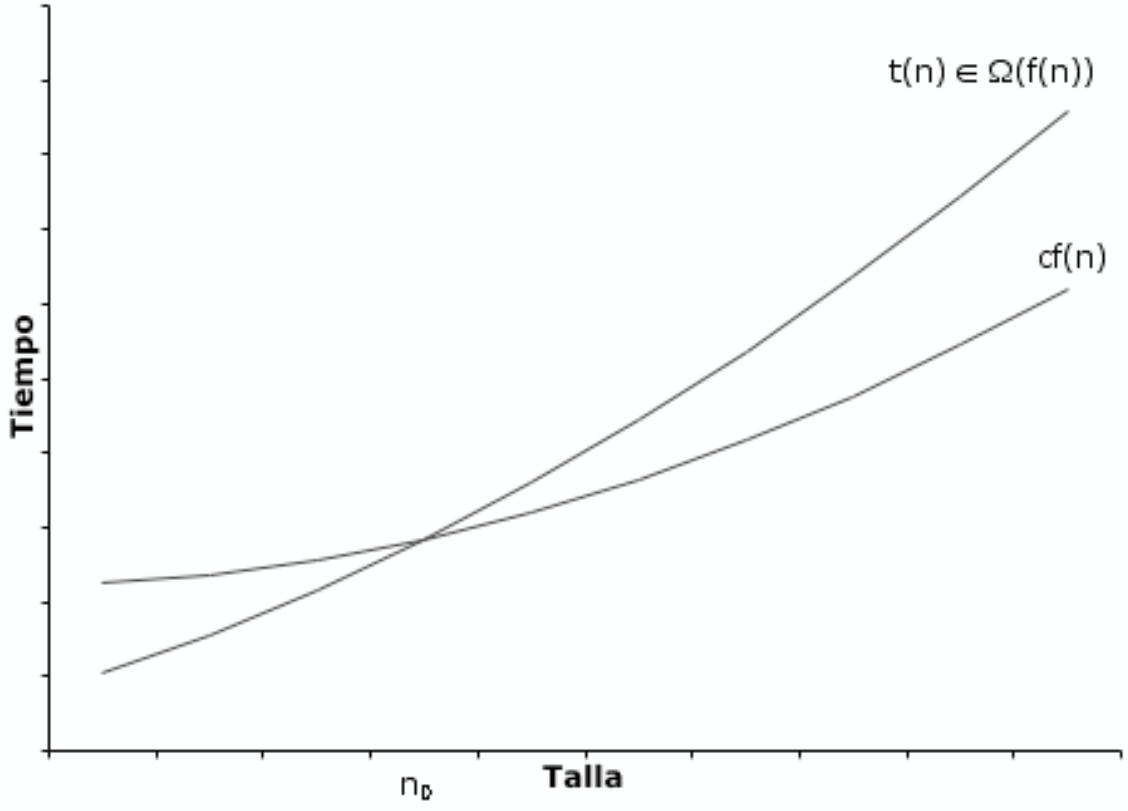
$$\exists c = 7, \exists n_0 = 4 : \forall n > 4 \quad 6*2^n+n^2 \leq 7*2^n$$

4 Notación asintótica (VI)

Notación Ω

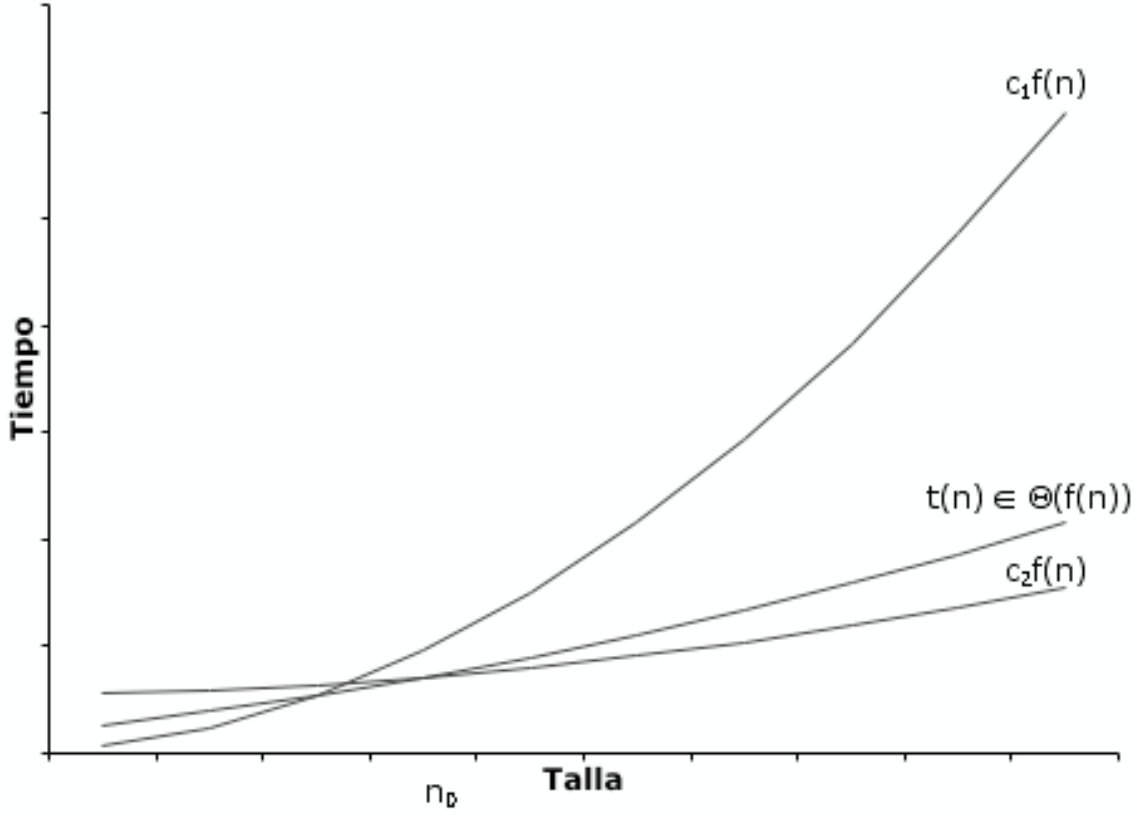
Sea $f : \mathbb{N} \to \mathbb{R}^+$, definimos la siguiente familia de funciones

$$\Omega(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n > n_0 \;\; t(n) \geq c f(n)\}$$

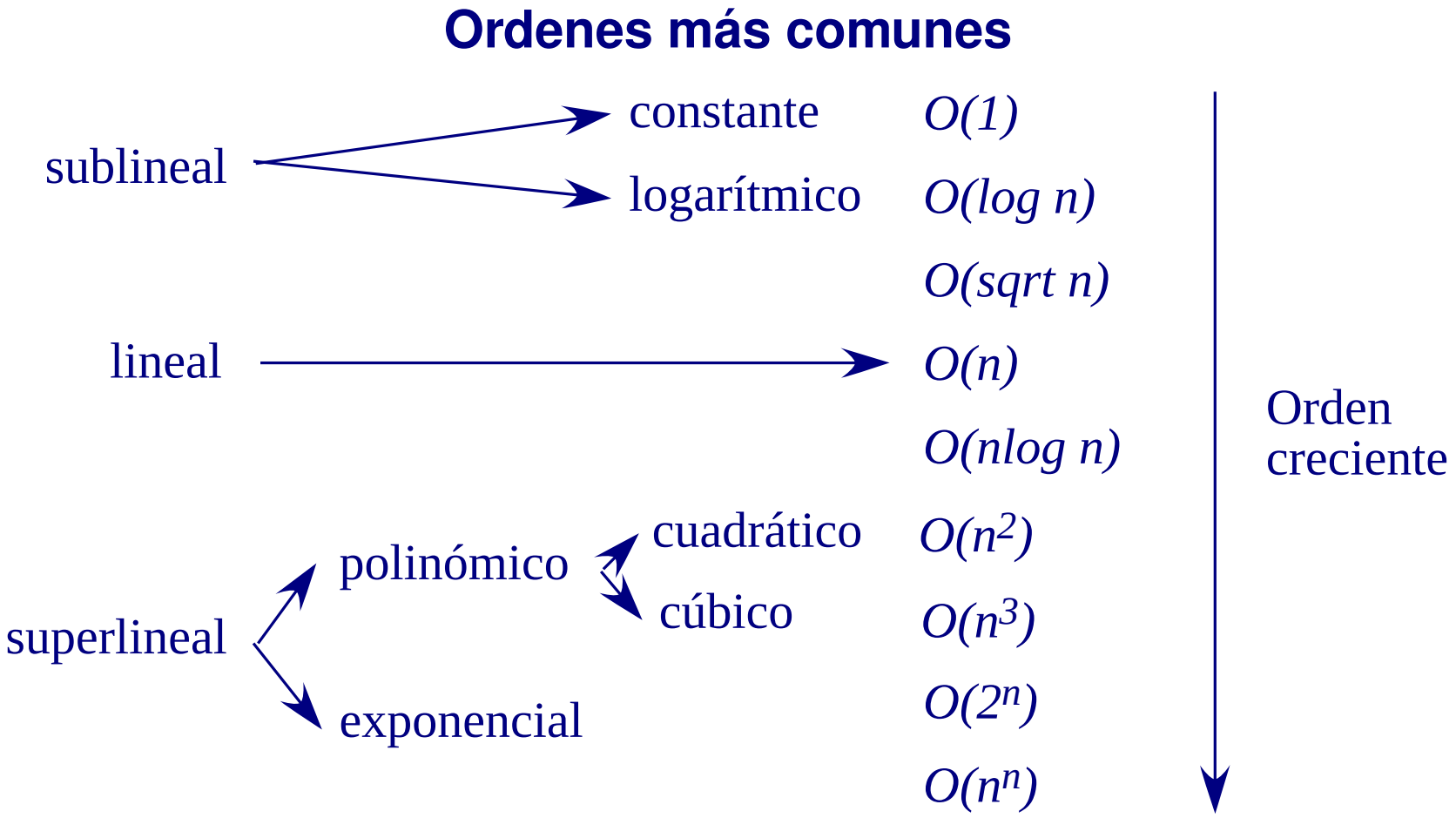


Notación Θ

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$



4 Notación asintótica (VIII)



Algoritmos cuadráticos

- Selección
- Burbuja
- Inserción

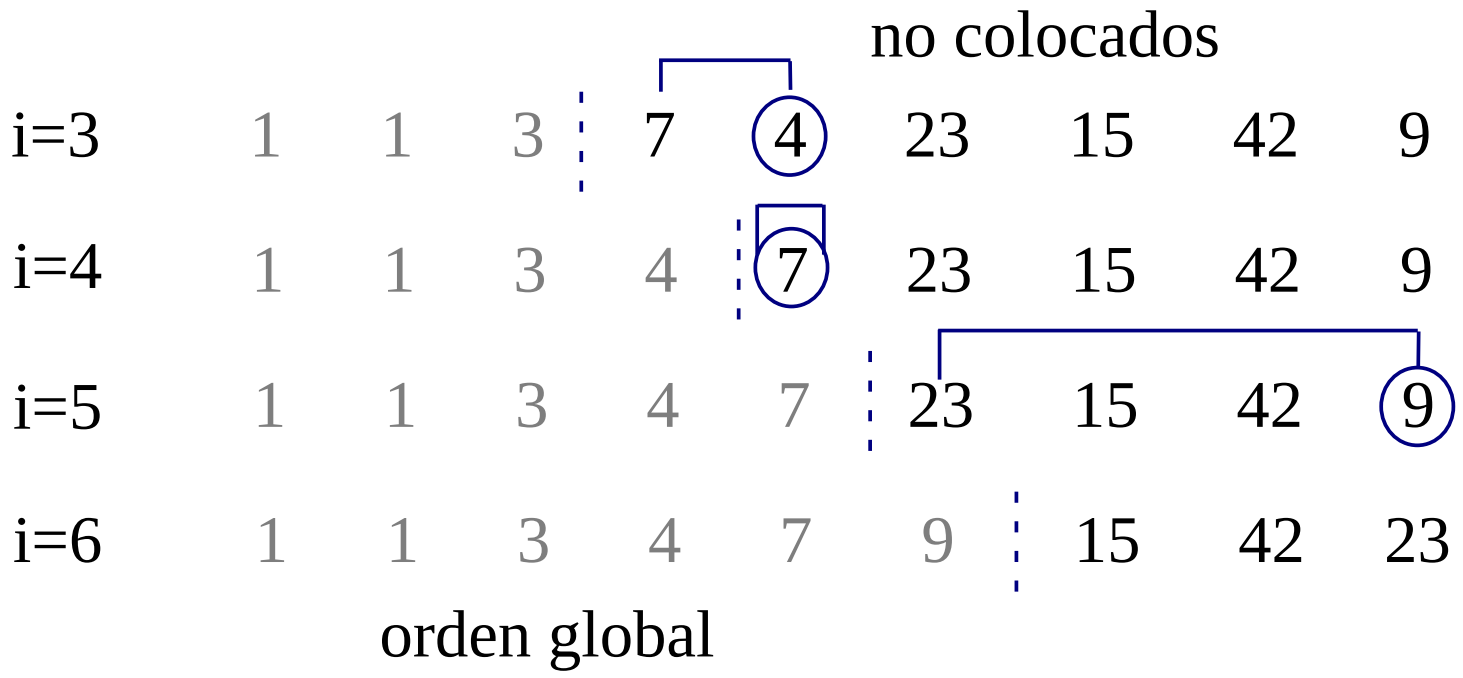
No cuadráticos: mergesort, quicksort, ...

Pasos:

- Comparación de dos elementos del vector.
- Intercambio de dos elementos del vector.

5.1 Ordenación por Selección

```
Para i=0 hasta n-2 hacer
  *  Buscar el mínimo entre i y n.
  *  Intercambiarlo con el elemento que ocupa la posición i.
```



5.1 Ordenación por Selección (II)

```
1   void Seleccion ( int v[], int n) {
2     int posmin, aux;
3     for (int i=0; i<n-1; i++) {
4       posmin = i;
5       for (int j=i+1; j<n; j++)
6         if (v[j] < v[posmin]) posmin=j;
7       aux = v[i];
8       v[i] = v[posmin];
9       v[posmin] = aux;
10    }
11  }
```

5.1 Ordenación por Selección (III)

Caso peor = Caso mejor

- **Paso:** comparaciones

$$Coste = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \frac{n^2-n}{2}$$

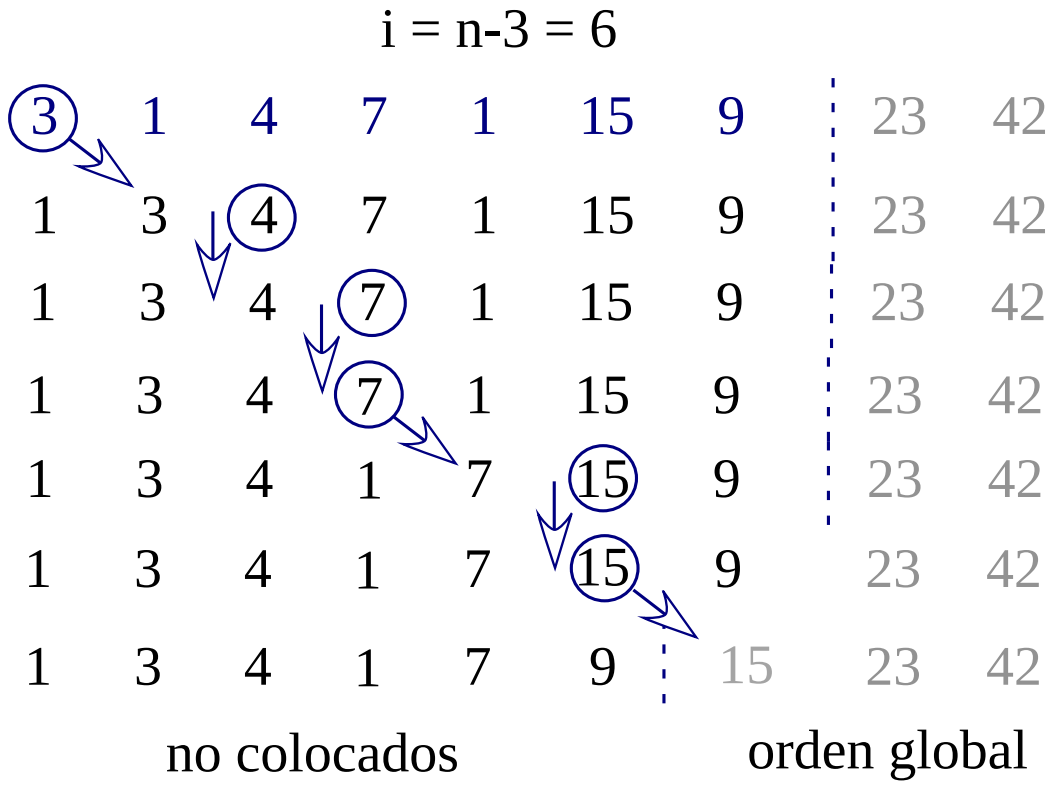
$$Coste \in O(n^2) : cuadrático$$

- **Paso:** intercambios

$$Coste = \sum_{i=0}^{n-2} 1 = n-1 \in O(n) : lineal$$

5.2 Método de la burbuja

```
Para i=n-1 hasta 1 hacer
  Mover el elemento mayor entre 0 e i a la posición i.
  intercambiando elementos sucesivos desordenados.
```



5.2 Método de la burbuja (II)

```
1  void Burbuja ( int v[], int n) {
2    int aux;
3    for (int i=n-1; i>0; i--)
4      for (int j=0; j<i; j++)
5        if (v[j] > v[j+1]) {
6          aux = v[j];
7          v[j] = v[j+1];
8          v[j+1] = aux;
9        }
10 }
```

5.2 Método de la burbuja (III)

- **Paso:** comparaciones (Caso mejor = Caso peor).

$$Coste = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{n^2 - n}{2}$$

$$Coste \in O(n^2) : cuadrático$$

- **Paso:** intercambios
  - **Caso mejor:** vector en orden creciente.

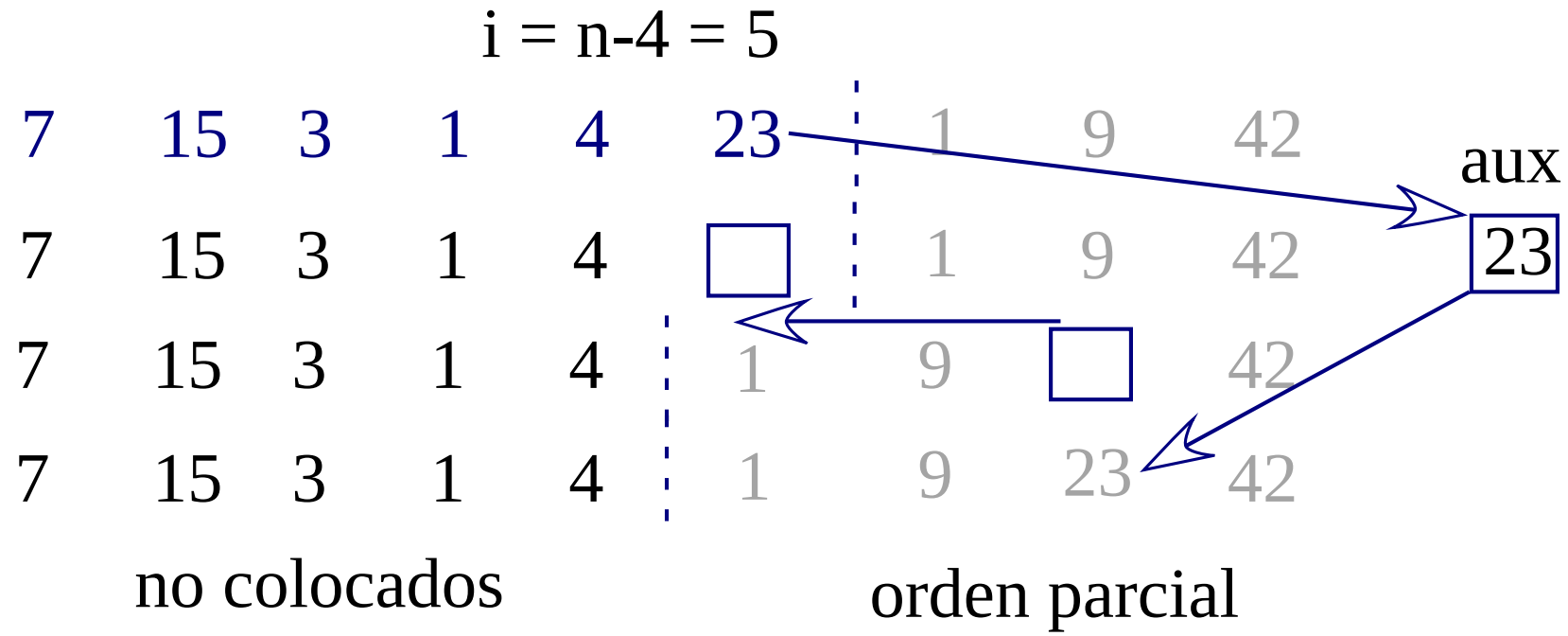
$$Coste = 0 \in O(1) : constante$$

  - **Caso peor:** vector en orden decreciente.

$$Coste = \frac{n^2 - n}{2} \in O(n^2) : cuadrático$$

5.3 Ordenación por Inserción

```
Para i=n-2 hasta 0 hacer
  * Guardar el elemento i en un auxiliar.
  * Desplazar una posición a la izquierda los elementos
    menores que el i-ésimo situados a su derecha.
  * Colocar el auxiliar en el hueco habilitado.
```



5.3 Ordenación por Inserción (II)

```
1  void CasiInsercion ( int v[], int n) {
2    int j, aux;
3    for (int i=n-2; i>=0; i--){
4      aux = v[i];
5      j=i;
6      while (v[j+1] < aux) {
7        v[j] = v[j+1];
8        j++;
9      }
10     v[j] = aux;
11 }
```

5.3 Ordenación por Inserción (III)

v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]
7	15	3	1	4	23	1	42	9

```
i=7
  aux=v[7]      (aux=42)
  j=7
  v[8]<aux?     (9<42?)  SI
    v[7]=v[8]      (v[7]=9)
  j=8
  v[9]<aux?     ¿Cuánto vale v[9]?
```

Solución: Colocar un *centinela* en la posición adicional $v[n]$.

5.3 Ordenación por Inserción (IV)

```
1  void Insercion ( int v[], int n) {
2    int i, j;
3    for (i=n-2; i>=0; i--){
4      v[n] = v[i];      // v[n] es el centinela
5      j=i;
6      while (v[j+1] < v[n]) {
7          v[j] = v[j+1];
8          j++;
9      }
10     v[j] = v[n];
11   }
12 }
```

5.3 Ordenación por Inserción (V)

- **Paso:** comparaciones
  - **Caso mejor:** vector en orden creciente.

$$Coste = \sum_{i=0}^{n-2} 1 = n-1 \in O(n) : lineal$$

  - **Caso peor:** vector en orden decreciente.

$$Coste = \sum_{i=0}^{n-2} \sum_{j=i}^{n-1} 1 = \frac{n^2+n-2}{2} \in O(n^2) : cuadrático$$

5.3 Ordenación por Inserción (VI)

- **Paso:** intercambios (en este caso contamos desplazamientos).
  - **Caso mejor:** vector en orden creciente.

$$Coste = \sum_{i=0}^{n-2} 2 = 2(n-1) \in O(n) : lineal$$

  - **Caso peor:** vector en orden decreciente.

$$Coste = \sum_{i=0}^{n-2} (2 + \sum_{j=i}^{n-2} 1) = \frac{n^2+3n-4}{2} \in O(n^2) : cuadrático$$

6 Coste espacial

Coste espacial: ocupación de memoria en función de la talla.

Paso $\equiv$ elementos de tipo base (no tienen porque ser bytes).

Ejemplo: Producto de una matriz cuadrada de tamaño $n \times n$ por un vector.

Coste: número de elementos de la matriz + elementos del vector.

No importa si son enteros, reales (float o double).

Puede usarse notación asintótica: O, Ω y Θ.

$$Coste = n^2 + n + k \in O(n^2)$$

6 Coste espacial (II)

Ejemplos

- Máximo de tres enteros: $Coste \in O(1)$.
- Suma de dos vectores: $Coste \in O(n)$.
- Ordenación de un vector por inserción: $Coste \in O(n)$.
- Producto de dos matrices cuadradas: $Coste \in O(n^2)$.

6 Coste espacial (III)

- **Caso sencillo:** algoritmos estáticos no recursivos (zona de datos en memoria).
- **Otros casos:**
  - Algoritmos con memoria dinámica: Uso del montículo (*heap*).
  - Algoritmos recursivos: Uso de la pila (*stack*).

Ejemplo: Cálculo del factorial.

- Versión iterativa: $Coste \in O(1)$.
- Versión recursiva: $Coste \in O(n)$.