COMPLEJIDAD COMPUTACIONAL

Índice

Índice

Introducción

Generalmente, un problema dado puede resolverse mediante muy diversos algoritmos o programas. No todas las soluciones son igualmente buenas.

¿De qué depende la calidad de un programa que resuelve un problema?

- ¿Elegancia de la idea en la que se basa el algoritmo?
- ¿Claridad en la organización del código del programa?
- ¿Vistosidad de la presentación de los resultados?
- ¿Eficiencia en la forma de obtener la solución?

Desde un punto de vista ***computacional***, el factor esencial es la ***eficiencia***.

Introducción

El ***análisis de algoritmos*** es una actividad muy importante en el proceso de desarrollo, especialmente en entornos con recursos restringidos (en tiempo o en memoria).

El análisis de algoritmos es necesario para:

- Comparar dos algoritmos distintos.
- Predecir el comportamiento de un algoritmo en circunstancias extremas.
- Ajustar los parámetros de un algoritmo para obtener los mejores resultados.

Introducción

El análisis se puede realizar de dos formas distintas:

- *empírica* (experimental), o
- *teóricamente*.

Tipo	***Ventajas***	***Inconvenientes***
Experimental	"Sencilla", resultados "reales"	Hace falta tener dos implementaciones igual de cuidadas, hacen falta datos reales, más costosa, dependiente de la máquina
Teórica	Flexible, "barata", independiente de la máquina	Resultados no "exactos"

Índice

Consumo de recursos: costes espaciales y temporales

Eficiencia: Capacidad de resolver el problema propuesto empleando un *bajo consumo de recursos computacionales*

Dos factores fundamentales de la *eficiencia:*

- ***Coste espacial*** o cantidad de memoria requerida
- ***Coste temporal*** o tiempo necesario para resolver el problema.

Para resolver un problema dado, un algoritmo o programa A será mejor que otro B si A resuelve el problema en menos tiempo y/o emplea menos cantidad de memoria que B.

En ocasiones *tiempo* y *memoria* son recursos competitivos y un buen algoritmo es aquel que resuelve el problema con un buen *compromiso* tiempo/memoria.

En lo que sigue nos ocuparemos principalmente del *coste temporal*.

Un ejemplo simple

El tiempo o la memoria por sí solos *no* son realmente adecuados para valorar la *calidad* de un programa: Consideremos tres programas simples para calcular $\mathbf{10^2}$:

```
int main(){ /*A1*/
  int m;
  m = 10 * 10; /*producto*/
  printf("%d\n", m);
}
```

```
int main(){ /*A2*/
  int i,m; m=0;
  for (i=1; i<=10; i++)
    m = m + 10; /*suma*/
  printf("%d\n", m);
}
```

```
int main(){ /*A3*/
  int i,j,m; m=0;
  for (i=1; i<=10; i++)
    for (j=1; j<=10; j++)
      m++;    /*sucesor*/
  printf("%d\n", m);
}
```

Sean t_*, t_+, t_s los tiempos que se requieren para un *producto*, una *suma* y un *sucesor*:

$$T_{A1} = t_* \qquad T_{A2} = 10\,t_+ \qquad T_{A3} = 100\,t_s$$

Un ejemplo simple (cont.)

Supongamos que $A1, A2, A3$ se ejecutan en cuatro computadores con diferentes características (diferentes tiempos de ejecución de *sucesor*, *suma* y *producto*):

t_*	$100\ \mu s$	$50\ \mu s$	$100\ \mu s$	$200\ \mu s$
t_+	$10\ \mu s$	$10\ \mu s$	$5\ \mu s$	$10\ \mu s$
t_s	$1\ \mu s$	$2\ \mu s$	$1\ \mu s$	$0{,}5\ \mu s$
$A1$	$100\ \mu s$	$50\ \mu s$	$100\ \mu s$	$200\ \mu s$
$A2$	$100\ \mu s$	$100\ \mu s$	$50\ \mu s$	$100\ \mu s$
$A3$	$100\ \mu s$	$200\ \mu s$	$100\ \mu s$	$50\ \mu s$

¿qué programa es mejor, $A1, A2, A3$?

Para cada computador hay un mejor programa

Índice

Problemas, instancias y *talla* de una instancia

Nuestro "problema simple" era demasiado restrictivo. En la práctica querremos calcular no sólo $\mathbf{10^2}$ sino, en general, $\boldsymbol{n^2}$, donde $\boldsymbol{n}$ es un *dato* del problema.

Según este nuevo planteamiento, el problema concreto de calcular $\mathbf{10^2}$ se considera una ***instancia*** del problema general de calcular $\boldsymbol{n^2}$.

Dado un problema, en general no todas sus instancias son igual de *"grandes"*. A cada *instancia* se le puede asignar un número entero que mida la "*envergadura*" de esa instancia. A este entero se le denomina ***talla***.

En nuestro problema de cálculo de $\boldsymbol{n^2}$, una medida natural de la *talla* es justamente $\boldsymbol{n}$: Así, la talla de la instancia $\mathbf{10^2}$ es $\mathbf{10}$ y la talla de la instancia $\mathbf{2\,345\,678^2}$ es $\mathbf{2\,345\,678}$.

Coste en función de la *talla* del problema

Podemos reescribir fácilmente $A1,\ A2,\ A3$ para calcular n^2 en vez de 10^2:

```
int main(){ /*A1*/
  int n, m;
  scanf("%d", &n);
  m = n * n; /*producto*/
  printf("%d\n", m);
}
```

```
int main(){ /*A2*/
  int i, n, m;  m=0;
  scanf("%d", &n);
  for (i=1; i<=n; i++)
    m = m + n; /*suma*/
  printf("%d\n", m);
}
```

```
int main(){ /*A3*/
  int i, j, n, m;  m=0;
  scanf("%d", &n);
  for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      m++;  /*sucesor*/
  printf("%d\n", m);
}
```

Talla=n; costes temporales (μs): $T_{A1} = t_*$ $T_{A2} = t_+ \cdot n$ $T_{A3} = t_s \cdot n^2$

¡El coste sigue dependiendo de t_*, t_+, t_s!

El coste como función de la talla sigue sin ser adecuado

Intuitivamente parece claro que el mejor programa es $A1$ y el peor $A3$. Pero, incluso fijando las características del computador, veremos que esta intuición no se mantiene claramente.

Por ejemplo si fijamos $t_* = 100\ \mu s,\ t_+ = 5\ \mu s,\ t_s = 1\ \mu s$:

	$T_{Ai}(n)$	$n = 4$	$n = 10$	$n = 1\,000$
$A1$	100	$100\ \mu s$	$100\ \mu s$	$100\ \mu s$
$A2$	$5n$	$20\ \mu s$	$50\ \mu s$	$5\,000\ \mu s$
$A3$	n^2	$16\ \mu s$	$100\ \mu s$	$1\,000\,000\ \mu s$

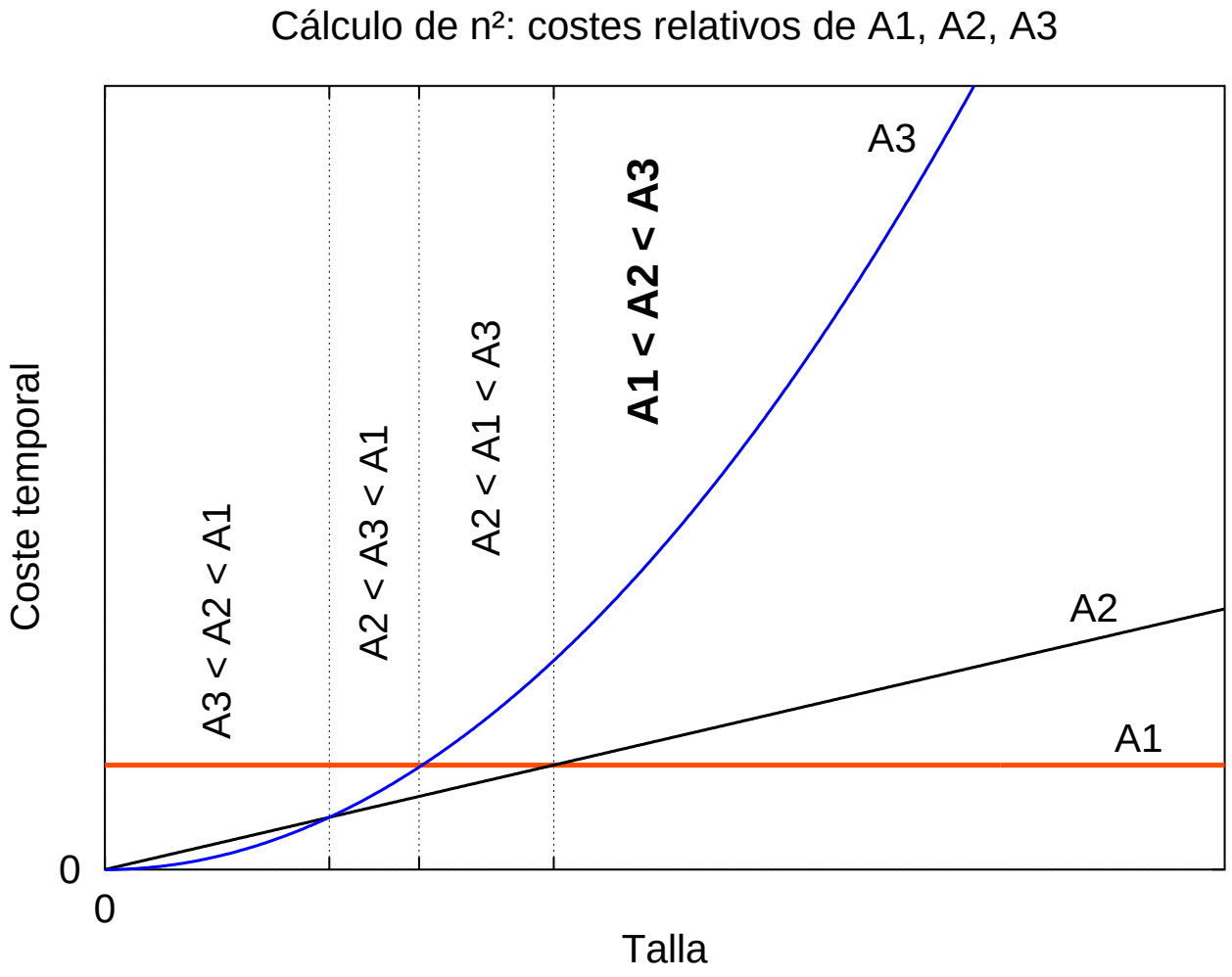
¿qué programa es mejor, $A1$, $A2$, $A3$?

Para cada talla hay un mejor programa

Una buena caracterización del coste debería permitir establecer la "calidad" de un programa con independencia tanto del computador como de la talla de las instancias concretas a procesar.

Coste asintótico

En general, tendríamos un comportamiento relativo de $A1,\ A2,\ A3$ tal como:



Una buena caracterización computacional de un programa:

Dependencia funcional del coste con la talla – ¡para tallas grandes!

Ventajas de la caracterización asintótica del coste computacional en función de la talla

- Generalmente los programas sólo son útiles para resolver problemas de gran talla (si es pequeña podríamos resolverlos manualmente sin dificultad)

- Al considerar sólo tallas grandes, se pueden hacer aproximaciones sencillas que simplifican considerablemente el análisis del coste.

- ***La bondad relativa de distintos programas ya no depende de los valores concretos de los tiempos de ejecución de las distintas operaciones elementales empleadas (siempre que éstos no dependan de la talla), ni de la talla concreta de las instancias del problema a resolver***

Simplificación del análisis del coste: concepto de "paso"

Un PASO es la ejecución de un segmento de código cuyo tiempo de proceso no depende de la talla del problema considerado, o bien está acotado por alguna constante.

***Coste computacional de un programa: Número de* PASOS *en función de la talla del problema*.**

Construcciones a las que se les puede asignar *1 PASO*:

- Asignación, operaciones aritméticas o lógicas, comparación, acceso a un elemento de vector o matriz, etc.
- Cualquier secuencia finita de estas operaciones cuya longitud no dependa de la talla.

Construcciones a las que no se le puede asignar 1 PASO, sino un número de PASOS en *función de la talla*:

- Asignación de variables estructuradas (ej. vectores) cuyo número de elementos dependa de la talla
- Bucles cuyo número de iteraciones dependa de la talla.

Análisis de costes (PASOS) de los programas de cálculo de n^2

```c
int main(){ /*A1*/
  int n, m;
  scanf("%d", &n);
  m = n * n; /*producto*/
  printf("%d\n", m);
}
```

```c
int main(){ /*A2*/
  int i, n, m;  m=0;
  scanf("%d", &n);
  for (i=1; i<=n; i++)
    m = m + n; /*suma*/
  printf("%d\n", m);
}
```

```c
int main(){ /*A3*/
  int i, j, n, m;  m=0;
  scanf("%d", &n);
  for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      m++;  /*sucesor*/
  printf("%d\n", m);
}
```

$$T_{A1}(n) = 1, \quad T_{A2}(n) = n, \quad T_{A3}(n) = n^2$$

Otro ejemplo de análisis del coste en PASOS:
impresión de los elementos de un vector en orden inverso

```c
#include <stdio.h>
#define MAXN 1000000
int main()                  /* invOrd.c */
{
    int i, n, x, v[MAXN];

    n = 0;     /* Inicializa contador de datos (talla) */
    printf("Teclear datos (fin: ^D)\n");
    while ((n < MAXN) && (scanf("%d", &x) == 1)) { /* Lee datos */
        v[n] = x;
        n++;                              /* Actualiza talla */
    }
    /* Imprime resultados en orden inverso */
    for (i = n-1; i >= 0; i--)
        printf("%d ", v[i]);
    printf("\n");
    return 0;
}
```

PASO: Accesos a v; $\quad$ ***Talla*** $= n;\ T_{invOrd}(n) = n + n = 2n$ ***PASOS***

Más ejemplos de análisis del coste en PASOS:
búsqueda del elemento más cercano a la media

```
#include <stdio.h>
#include <math.h>
#define MAX 100000
int main()
{
    int cercano, i, n = 0, aux, A[MAX];
    double media, suma;

    while (n < MAX && scanf("%d", &aux) == 1) {
        A[n] = aux; n++;
    }
    suma = 0;
    for (i = 0; i < n; i++) suma += A[i];
    media = suma/n; cercano = 0;                    /* Cálculo de la media */
    for (i = 1; i < n; i++)                      /* Búsqueda del más cercano */
        if (fabs(A[i]-media) < fabs(A[cercano]-media))
            cercano = i;
    printf("El más cercano es A[%d]=%d\n", cercano, A[cercano]);
}
```

PASO: Accesos a A; $Talla = n;$ $T_{mascerca}(n) =$ ***??? PASOS***

Más ejemplos de análisis del coste en PASOS:
cálculo de la moda de una serie de edades: lectura de datos

```
#include <stdio.h>
#define MAXDATOS 100000
#define MAXEDAD 150
int main()                                  /* modax.c: cálculo de la moda */
{
    int i, j, n, edad, frec, maxFrec, moda, edades[MAXDATOS];

    n = 0;                              /* Inicializa contador de datos (talla) */
    printf("Teclear edades, finalizando con ^D\n");
    while ((n < MAXDATOS) && (scanf("%d", &edad) != EOF)) /* Lee edades */
        if ((edad >= 0) && (edad < MAXEDAD)) {              /* hasta EOF, */
            edades[n] = edad;                             /* las memoriza */
            n++;                                         /* y actualiza n */
        }
```

Más ejemplos de análisis del coste en PASOS:
cálculo (ineficiente) de la moda de una serie de edades

```
/* moda0.c: cálculo (ineficiente) de la moda */

    maxFrec=0;                              /* Explora n veces edades[] para */
    for (i = 0; i < n; i++) {               /* determinar cuál es la edad */
        frec = 0;                           /* que mas se repite (moda) */
        for (j = 0; j < n; j++)
            if (edades[i] == edades[j])
                frec++;
        if (frec > maxFrec) {
            maxFrec = frec;
            moda = edades[i];
        }
    }
    printf("Leidos %d datos; Moda=%d (frecuencia=%d)\n", n, moda, maxFrec);
    return 0;
}
```

PASO: Comparaciones en if; $Talla = n;\ T_{mod}(n) =$*??? PASOS*

Más ejemplos de análisis del coste en PASOS:
cálculo *más eficiente* de la moda de una serie de edades

```
/* moda1.c: cálculo (poco eficiente) de la moda */

  maxFrec = 0;                          /* Explora edades[] maxEdad veces para */
  for (i = 0; i < MAXEDAD; i++) {   /* determinar cuál es la edad */
    frec = 0;                           /* que más se repite (moda) */
    for (j = 0; j < n; j++)
      if (edades[j] == i) frec++;
    if (frec > maxFrec) {
      maxFrec = frec;
      moda = i;
    }
  }
  printf("Leidos %d datos; Moda=%d (frecuencia=%d)\n", n, moda, maxFrec);
  return 0;
}
```

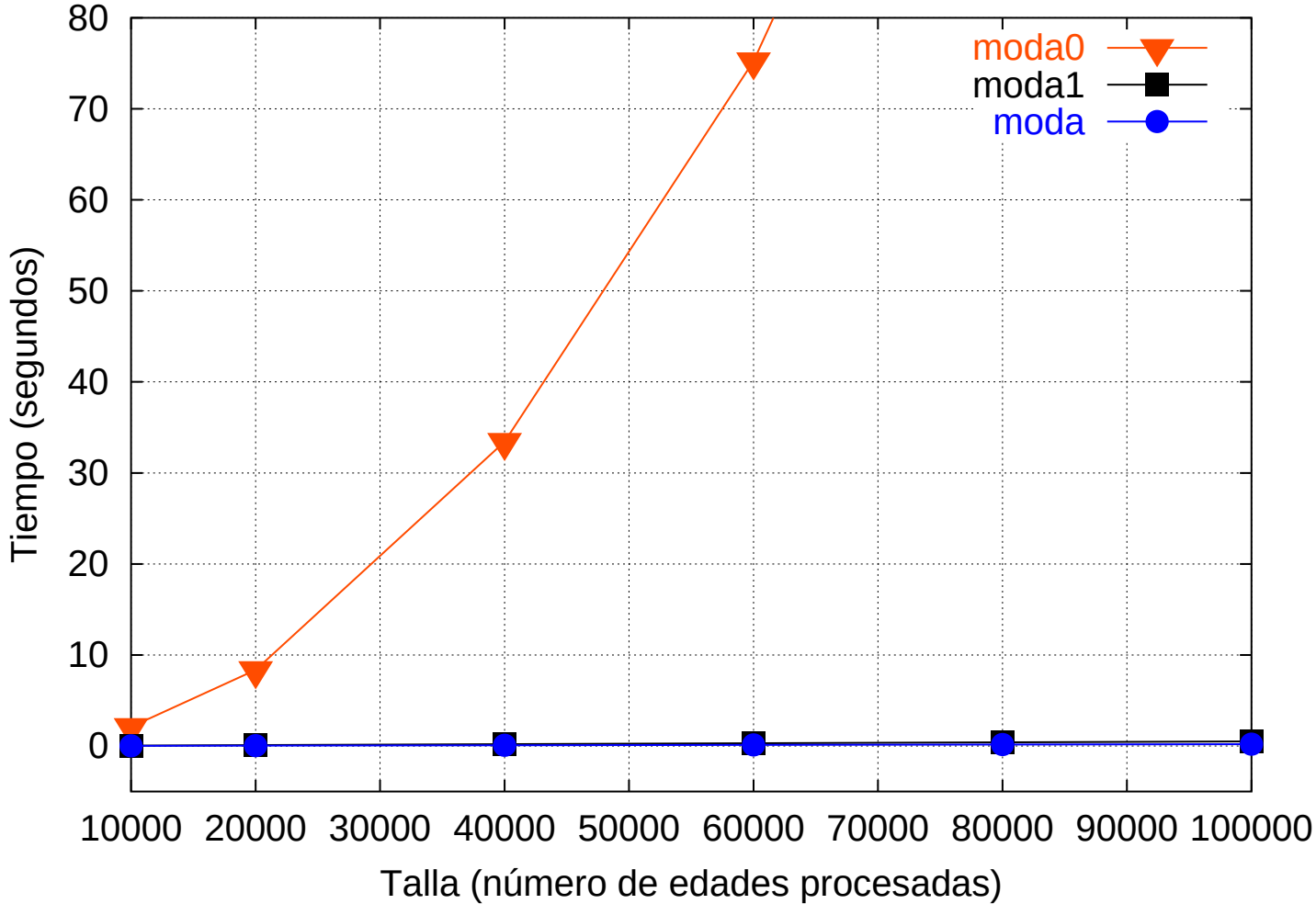
PASO: Comparaciones en if; $\quad$ ***Talla*** $= n$***;*** $\quad$ $T_{moda}(n) =$***??? PASOS***

Más ejemplos de análisis del coste en PASOS:
cálculo *eficiente* de la moda de una serie de edades

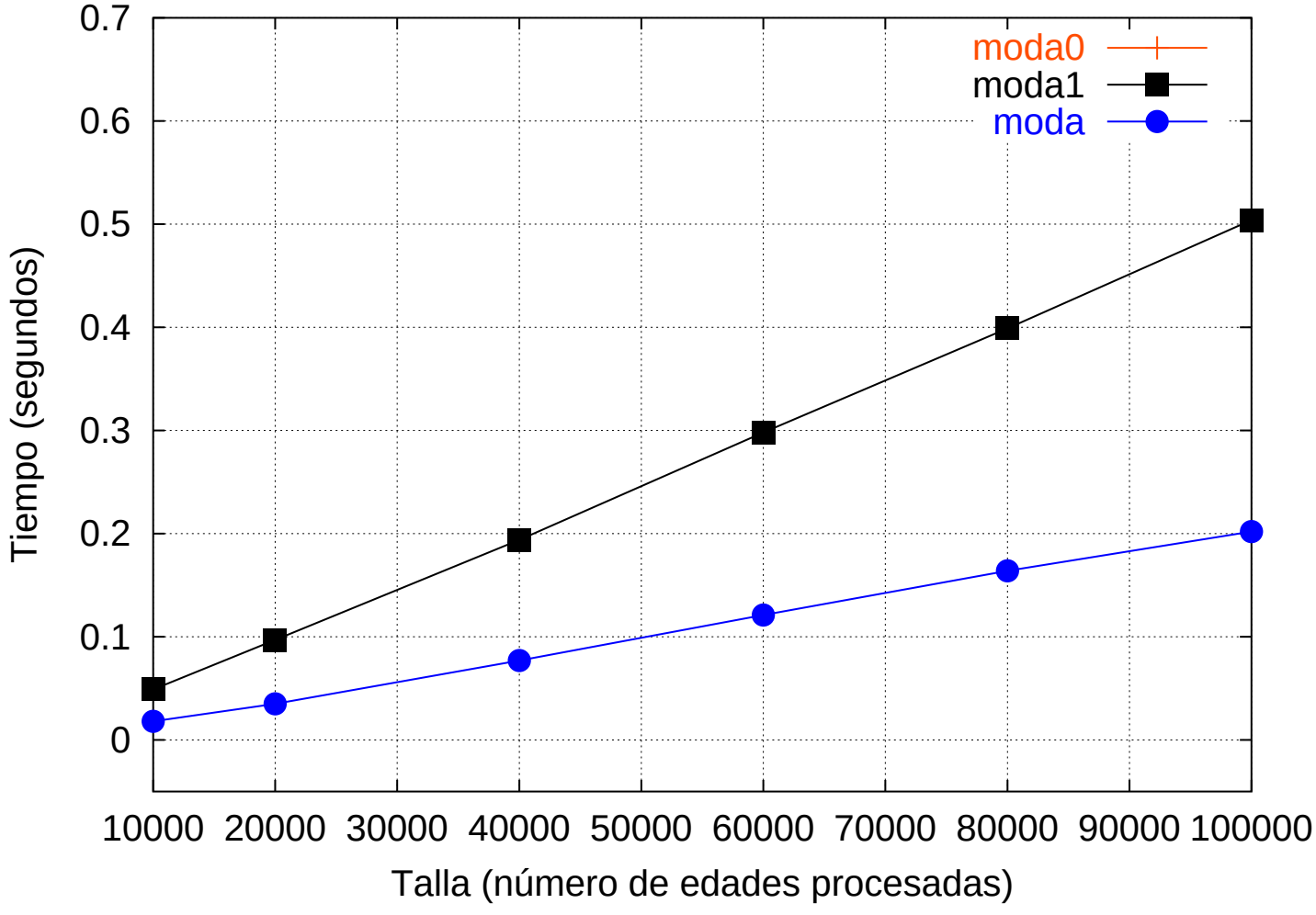
```
#include <stdio.h>
#define MAXEDAD 150
int main()                          /* moda.c: cálculo eficiente de la moda */
{
    int n = 0, edad, maxFrec = 0, moda, frecs[MAXEDAD];
    /* Inicializa vector de frecuencias */
    for (edad = 0; edad < MAXEDAD; edad++)
        frecs[edad] = 0;
    printf("Teclear edades, fin con ^D\n");  /*  Lee edades hasta EOF  */
    while (scanf("%d", &edad) != EOF)         /* y actualiza frecuencias */
        if ((edad >= 0) && (edad < MAXEDAD)) {
           n++; frecs[edad]++;
        }
    for (edad = 0; edad < MAXEDAD; edad++)    /* máx. frecuencia (moda) */
        if (frecs[edad] > maxFrec) {
            maxFrec = frecs[edad]; moda = edad;
        }
    printf("Leidos %d datos; Moda=%d (frecuencia=%d)\n", n,moda,maxFrec);
}
```

PASO: Comparaciones en if; $\quad$ ***Talla*** $= n$; $\quad T_{moda}(n) =$??? ***PASOS***

Costes de las variantes del cálculo de la moda:
representación gráfica



Costes de las variantes del cálculo de la moda:
comparación de las versiones más eficientes



Más ejemplos de análisis del coste en PASOS:
funciones extrañas

```c
#include <stdio.h>

int f1(int x, int n)
{
    int i;
    for (i = 1; i <= n; i++)
        x += i;
    return x;
}

int f2(int x, int n)
{
    int i;
    for (i = 1; i <= n; i++)
        x += f1(i, n);
    return x;
}
```

```c
int main()
{
    int a, n;

    scanf("%d", &n);
    a = f2(0, n);
    printf("%d\n", f1(a, n));
    return 0;
}
```

PASO: $+=$; ***Talla*** $= n$; $T_{main}(n) =$??? ***PASOS***

Índice

A menudo el coste NO es (sólo) función de la talla

En los ejemplos vistos hasta ahora, todas las "instancias" de una talla dada tenían el mismo coste computacional. Pero esto *no* es siempre así. En el siguiente ejemplo, ¿cuál es el coste de la función `busca(n)`?:

```
#include <stdio.h>
#define MAXN 1000000

int busca(int *v, int n, int x) /* busca x en v[0...n-1] */
{
    int i;
    for (i=0; i<n; i++)
        if (v[i] == x)
            return i;
    return -1;
}
```

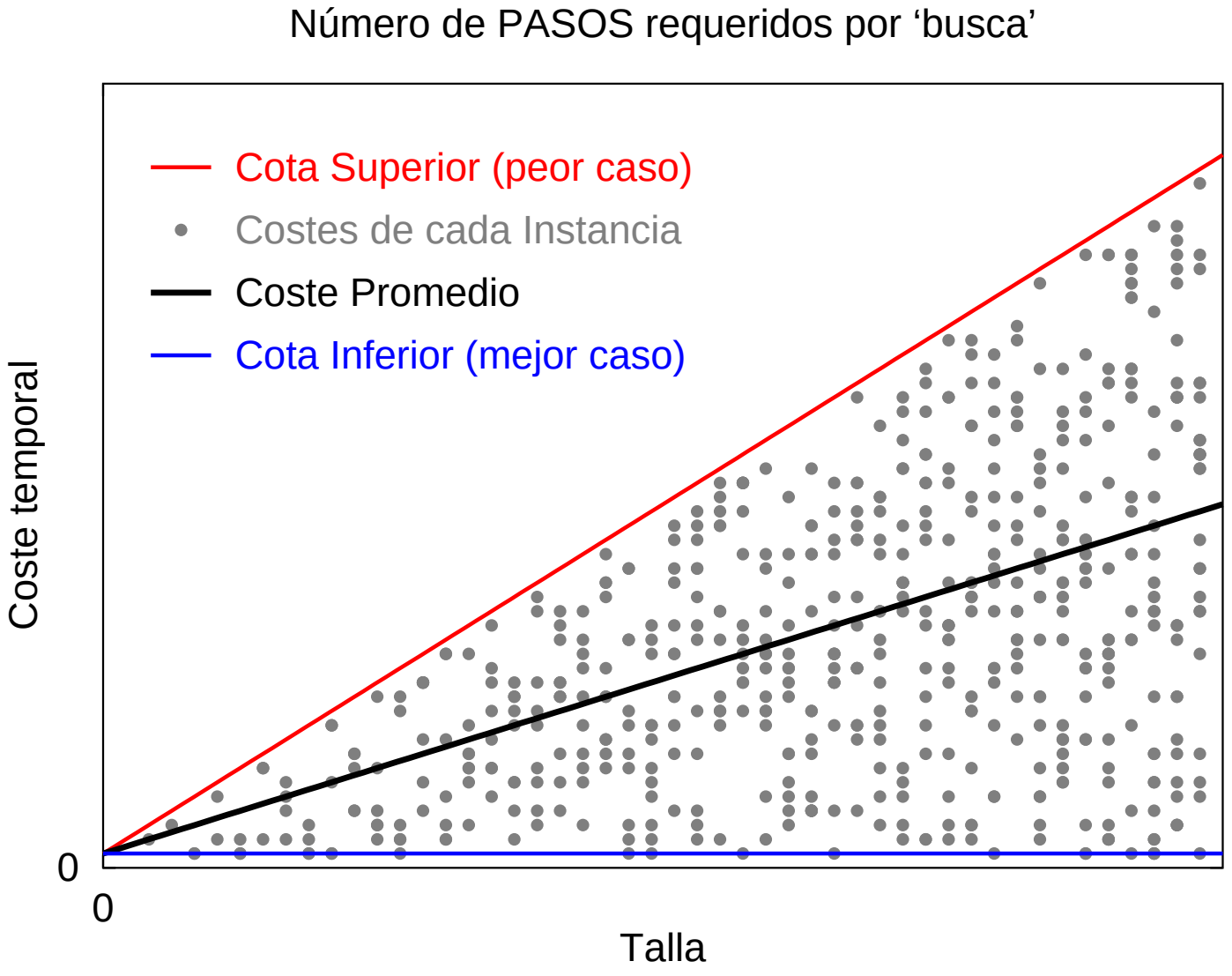
A menudo el coste NO es (sólo) función de la talla

```
int main() /* busca.c */
{
    int i, x, aux;
    int v[MAXN], n = 0;     /* Vector donde buscar y su talla */

    printf("Teclear datos, fin con ^D)\n");
    while ((n < MAXN) && (scanf("%d", &aux) != EOF)) { /* lee v[] */
        v[n] = aux;
        n++;
    }
    printf("Dato a buscar: ");
    while (scanf("%d", &x) == 1) {
        i = busca(v, n, x);
        printf("Posición de %d: %d\n", x, i);
        printf("Dato a buscar: ");
    }
    return 0;
}
```

PASO: comparación en if; $\quad$ ***Talla*** $= n;$ $\quad$ $T_{busca}(n) = ???$

¡Depende del contenido del vector y del valor concreto del elemento a buscar!

Extremos del coste: casos mejor, peor y promedio



$$T^b_{busca}(n) = 1 \quad T^w_{busca}(n) = n \quad T^m_{busca}(n) = n/2$$

A veces es difícil calcular costes "exactos" para cada talla. Ejemplo:

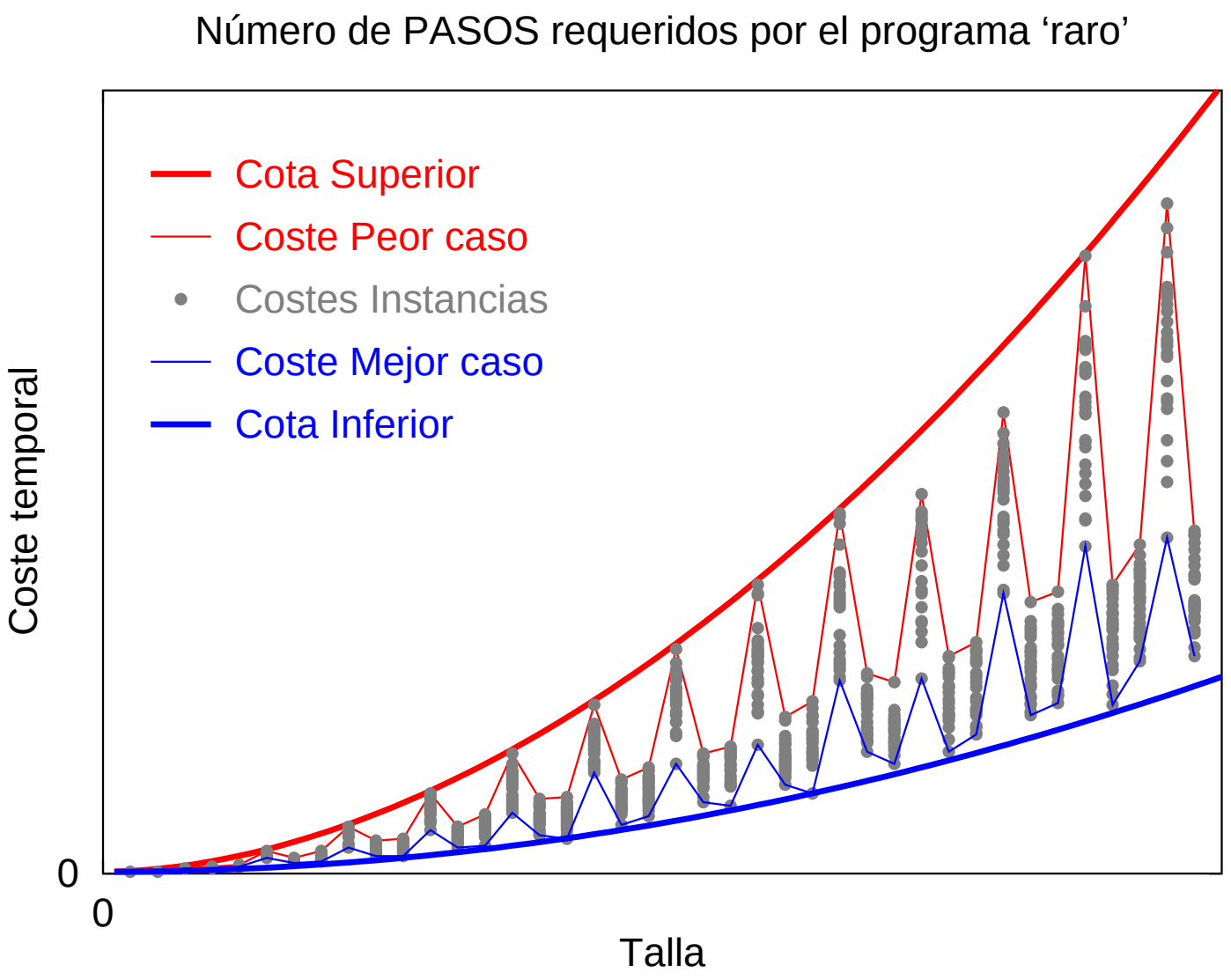
```
#include <stdio.h>
#define K 64
#define maxN 1000000

int main()   /* raro.c: Realiza un extraño proceso sobre un vector */
{
    int i, j, k, n, x, X[maxN], pasos;

    n = 0;
    pasos = 1;
    printf("Teclear datos\n");
    while ((n < maxN) && (scanf("%d", &x) != EOF)) {   /* lee datos */
        X[n] = x;
        n++;
    }
```

Ejemplo (cont.)

```
    for (i = 2; i < n; i++) {                    /* procesa datos */
        for (k = 1; k <= K; k++)
            X[i] = X[i] + k;
        if (n % 3 == 0) {
            for (k = 1; k <= K; k++)
                X[i] = X[i] - k;
            if ((i % 2 == 0) && (X[i] != X[i-1]))
                for (j = i; j <= n - 2; j++)
                    for (k = 1; k <= K; k++)
                        X[i] = k * X[i];
        }
        else
            for (j = i; j <= n - i; j = j + 2)
                if (X[i] % 2 == 0)
                    for (k = 1; k <= K; k++)
                        X[i] = X[i] + 2 * k;
    }
    for (i = 0; i < n; i++) printf("%d\n", X[i]);
}
```

PASO: accesos a X. El número total de pasos varía de manera diferente con las instancias de cada talla según ésta sea o no múltiplo de 3.

Cotas superior e inferior de los costes



Generalmente es suficiente determinar las $\mathbf{cotas\ asintóticas}$*; es decir para* $\mathbf{talla}$ $>>>$

Índice

Aspectos esenciales en la determinación de costes computacionales

- El coste no se puede expresar por un único valor (ej. el "tiempo de ejecución", sino que debe ser una ***función de la talla*** del problema a resolver
- La comparación de funciones de coste debe ser ***insensible a "constantes de implementación"*** tales como los tiempos concretos de ejecución de operaciones individuales (cuyo coste sea independiente de la talla)
- La independencia de las constantes se consigue considerando sólo el comportamiento ***asintótico*** de la función de coste (es decir, para ***tallas grandes***)
- A menudo ocurre que, para una talla dada, hay diversas instancias con costes diferentes, por lo que ***el coste no puede expresarse propiamente como función de la talla***
- En estas ocasiones conviene determinar ***cotas superiores e inferiores*** de la función coste; es decir, en el ***peor caso*** y en el ***mejor caso***
- Hay situaciones en las que incluso las cotas para mejores y peores casos son funciones complicadas. Generalmente bastará determinar ***funciones simples*** que ***acoten superior e inferiormente los costes*** de todas las instancias ***para tallas grandes***

Notación asintótica

Abstracción de las constantes asociadas al concepto de "PASO", de la noción de "tallas grandes" y de la de "cotas asintóticas":

Sea $f : \mathbb{N} \to \mathbb{R}^+$ una función de los naturales en los reales positivos. Se define:

- $O(f(n)) \doteq \{t : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}),\ \forall n \geq n_0 \quad t(n) \leq cf(n)\}$
- $\Omega(f(n)) \doteq \{t : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}),\ \forall n \geq n_0 \quad t(n) \geq cf(n)\}$
- $\Theta(f(n)) \doteq O(f(n)) \cap \Omega(f(n))$

A partir de la definición de $\Theta(f(n))$ se tiene:

$\Theta(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c_1, c_2 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}),\ \forall n \geq n_0 \quad c_1 f(n) \leq t(n) \leq c_2 f(n)\}$

Ejemplos

$$t(n) = 3n + 2$$

$$t(n) = 100n + 6$$

$$t(n) = 10n^2 + 4n + 2$$

$$t(n) = n^2 + 1000n - 100$$

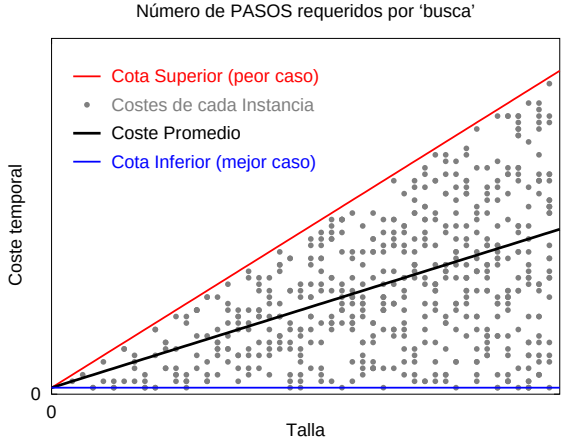
Ejemplos

$$
\begin{aligned}
t(n) = 3n + 2 \quad & \leq 4n \ \forall n \geq 2 \quad \Rightarrow \ t(n) \in O(n) \\
& \geq 3n \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n) \ ; \quad t(n) \in \Theta(n) \\
t(n) = 100n + 6 \quad & \leq 101n \ \forall n \geq 6 \quad \Rightarrow \ t(n) \in O(n) \\
& \geq 100n \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n) \ ; \quad t(n) \in \Theta(n) \\
t(n) = 10n^2 + 4n + 2 \quad & \leq 11n^2 \ \forall n \geq 5 \quad \Rightarrow \ t(n) \in O(n^2) \\
& \geq 10n^2 \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n^2) \ ; \quad t(n) \in \Theta(n^2) \\
t(n) = n^2 + 1000n - 100 \quad & \leq 2n^2 \ \forall n \geq 1000 \ \Rightarrow \ t(n) \in O(n^2) \\
& \geq n^2 \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n^2) \ ; \quad t(n) \in \Theta(n^2)
\end{aligned}
$$

Ejemplos de costes en notación asintótica

$\boldsymbol{A1},\ \boldsymbol{A2},\ \boldsymbol{A3}$: $T_{A1}(n) \in \Theta(1)$ $T_{A2}(n) \in \Theta(n)$ $T_{A3}(n) \in \Theta(n^2)$

Moda: $T_{moda}(n) \in \Theta(n^2)$ (ineficiente) $T_{moda}(n) \in \Theta(n)$ (eficiente)

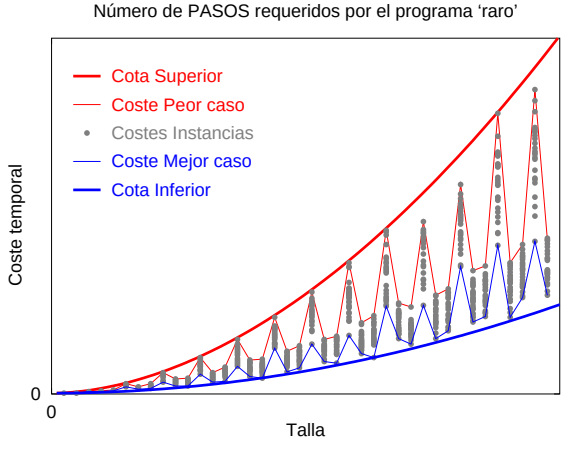


busca:

Peor caso: $O(n)$

Mejor caso: $\Omega(1)$

Promedio: $O(n)$



raro:

Peor caso: $O(n^2)$

Mejor caso: $\Omega(n)$

Promedio: $O(n^2)$

Notación asintótica: algunas propiedades útiles

Relación de orden entre órdenes de funciones:

- $f(n) \in \Theta(f(n))$
- $O(f(n)) \subset O(g(n)) \;\Leftrightarrow\; f(n) \in O(g(n)) \wedge g(n) \notin O(f(n))$
- $f(n) \in O(g(n)) \;\Leftrightarrow\; g(n) \in \Omega(f(n))$

Orden de suma de funciones: función dominante

- $(f_i(n) \in \Theta(g_i(n))\ 1 \leq i \leq k) \quad \Rightarrow \quad \sum_{i=1}^{k} f_i(n) \in \Theta\left(\max_{1 \leq i \leq k} g_i(n)\right)$

Propiedades útiles para análisis asintóticos

Regla del límite

Para dos funciones arbitrarias f y g: $\mathbb{N} \to \mathbb{R}^+$:

- *si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}^+$ entonces $f(n) \in \Theta(g(n))$
- *si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ entonces $f(n) \in O(g(n))$ pero $f(n) \notin \Theta(g(n))$, y
- *si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$ entonces $f(n) \in \Omega(g(n))$ pero $f(n) \notin \Theta(g(n))$

Por ejemplo, $f(n) = \log n$ y $g(n) = \sqrt{(n)}$. ¿Cuál es su orden relativo?

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} \overset{\text{L'Hopital}}{=} \lim_{n\to\infty} \frac{1/n}{1/(2\sqrt{n})} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$

Notación asintótica: más propiedades útiles

Algunas sumas de series

- Serie aritmética: $a_i = a_{i-1} + r \;\Rightarrow$

$$\sum_{i=1}^{n} a_i \;=\; n\,\frac{a_1 + a_n}{2} \;=\; a_1\, n + \frac{r}{2}\, n\,(n-1)$$

- $$\sum_{i=1}^{n} i \;=\; n\,\frac{n+1}{2}$$

- $$\sum_{i=1}^{n} i^2 \;=\; \frac{1}{3}\, n\,(n+1)\,(n+\frac{1}{2})$$

Notación asintótica: más propiedades útiles

Órdenes de funciones polinómicas

- $c \in \Theta(1)$, $\quad \forall c \in \mathbb{R}^+$
- $\sum_{i=0}^{k} c_i n^i \in \Theta(n^k)$, $\quad \forall c_k \in \mathbb{R}^+, \;\; \forall c_i \in \mathbb{R}, \;\; 1 \leq i < k$
- $\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$, $\quad \forall k \in \mathbb{N}^+$
- $\sum_{i=1}^{n} (n-i)^k \in \Theta(n^{k+1})$, $\quad \forall k \in \mathbb{N}^+$

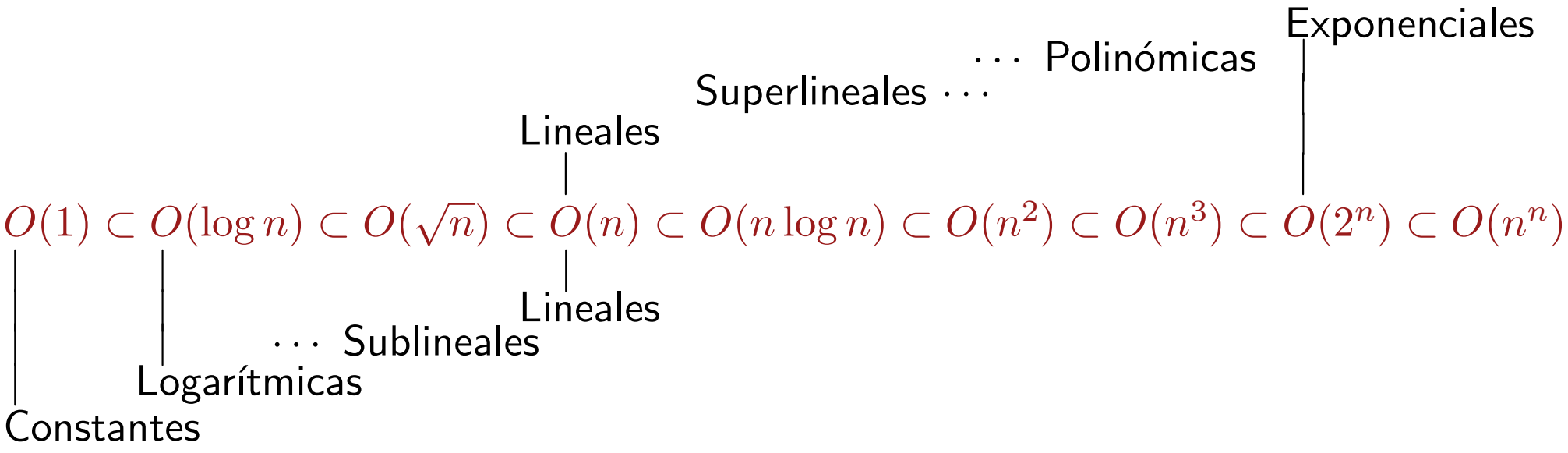
Notación asintótica: otras propiedades

Órdenes de funciones exponenciales, logarítmicas, etc.

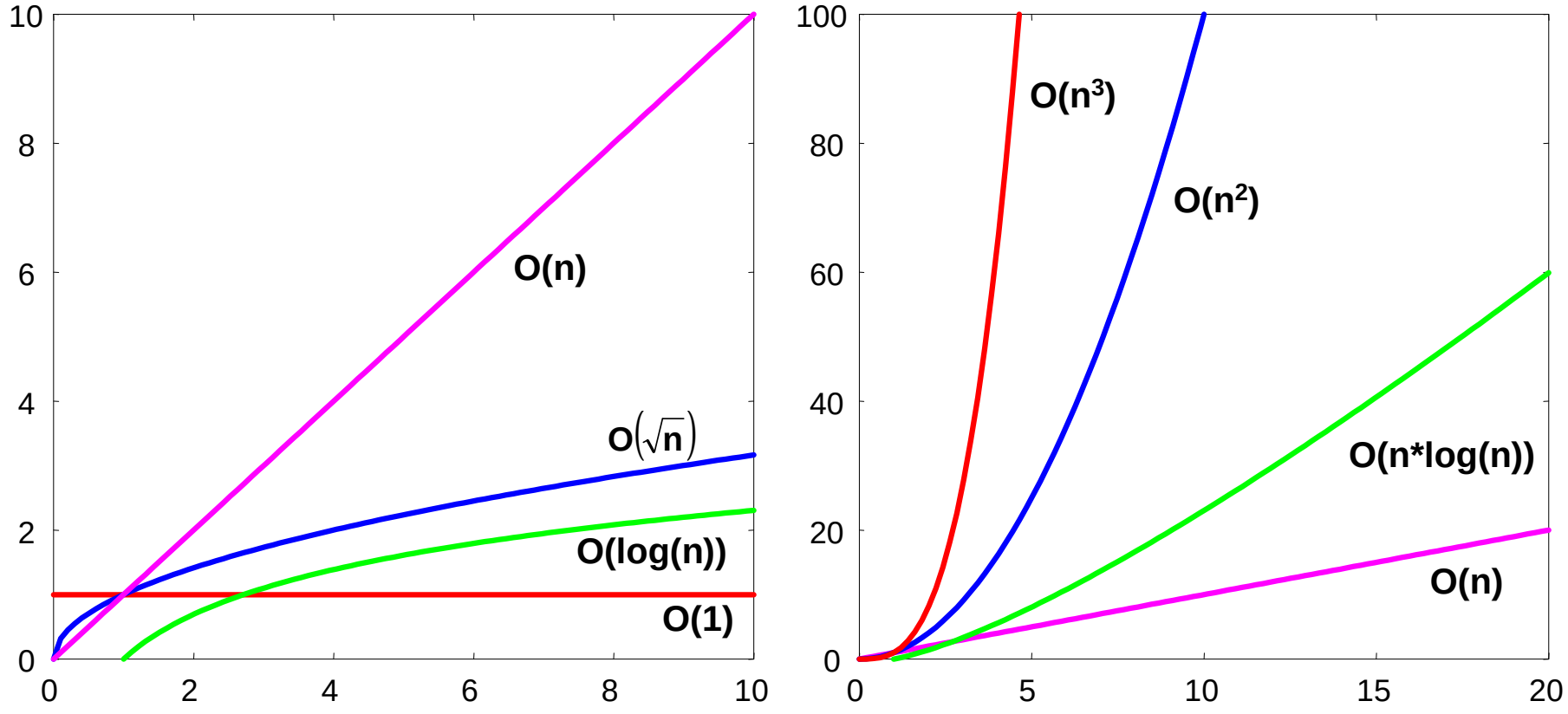
- $n! \in \Omega\left(2^n\right), \quad n! \in O\left(n^n\right)$
- $\log(n!) \in \Theta\left(n \log n\right)$
- $\sum_{i=1}^{n} r^i \in \Theta\left(r^n\right), \quad \forall r \in \mathbb{R}^{>1}$
- $\sum_{i=1}^{n} \frac{1}{i} \in \Theta\left(\log n\right)$
- $\sum_{i=1}^{n} \frac{i}{r^i} \in \Theta\left(1\right), \quad \forall r \in \mathbb{R}^{>1}$

Notación asintótica: jerarquía de costes computacionales

Algunas relaciones entre órdenes usuales:

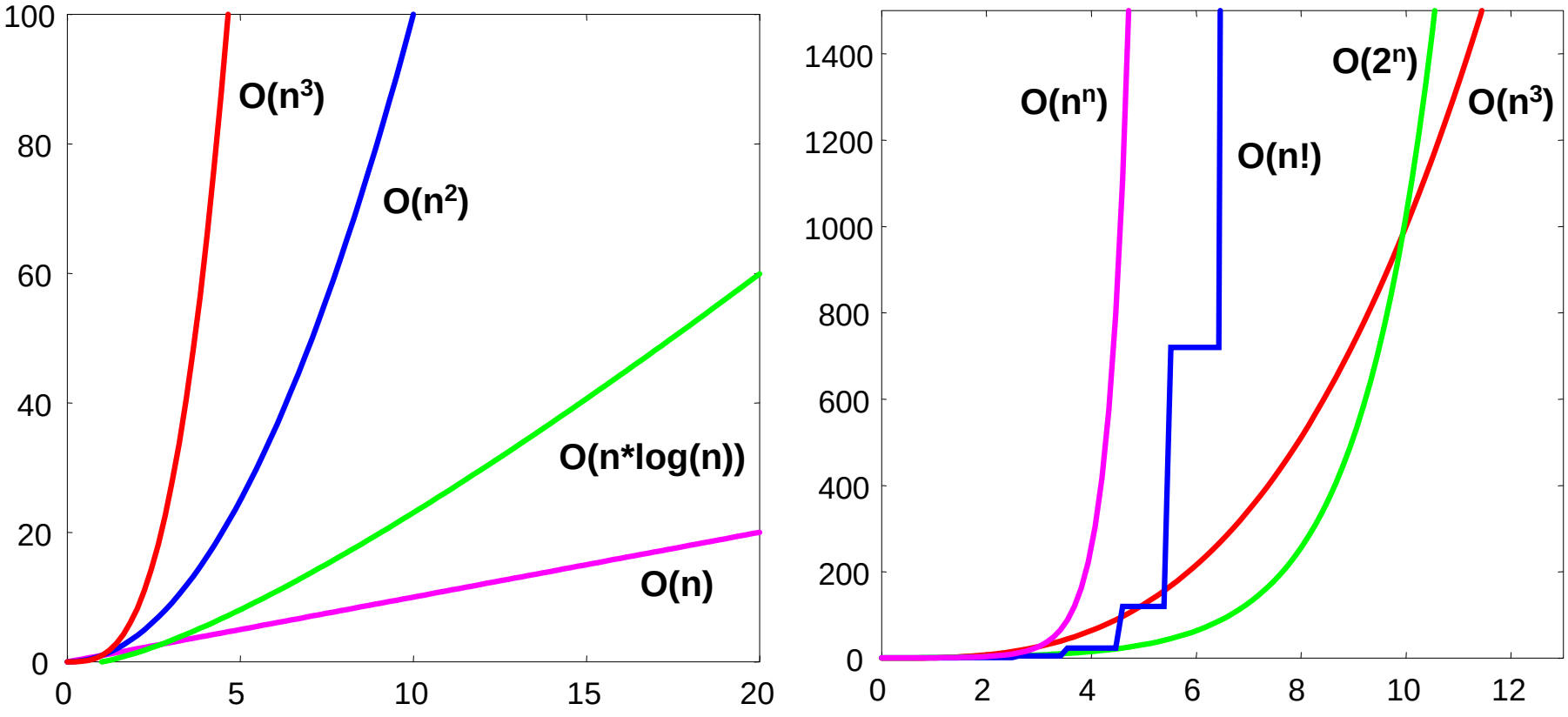


Notación asintótica: jerarquía de costes computacionales



sublineales, lineales y superlineales

Notación asintótica: jerarquía de costes computacionales



superlineales, polinómicas y exponenciales

Jerarquía de costes computacionales: consecuencias prácticas

Máximo tamaño aproximado de un problema, n, que puede ser resuelto por diversos algoritmos y computadores con distintas velocidades de proceso:

Coste	1 paso = 1 ms			1 paso = 0.1 ms (10 veces mas rápido)			
temporal	1 seg	1 min	1 hora	1 seg	1 min	1 hora	$n'=f(n)$
$\log_2 n$	$\approx 10^{330}$	$\approx 10^{2\cdot 10^4}$	$\approx 10^{10^{16}}$	$\approx 10^{3\cdot 10^3}$	$\approx 10^{2\cdot 10^5}$	$\approx 10^{10^{17}}$	n^{10}
n	$1\,000$	$6\cdot 10^4$	$3{,}6\cdot 10^6$	10^4	$6\cdot 10^5$	$3{,}6\cdot 10^7$	$10\,n$
$n\log_2 n$	141	$4\,896$	$2\cdot 10^5$	$1\,003$	$4\cdot 10^4$	$1{,}7\cdot 10^6$	$\approx 9\,n$
n^2	32	245	$1\,897$	100	775	$6\,000$	$3{,}16\,n$
2^n	10	16	22	13	19	25	$n+3$

Al aumentar el tiempo de cálculo y/o la velocidad del computador, un algoritmo $\cdots$

- ***logarítmico*** incrementa ***enormemente*** la talla de los problemas abordables
- ***lineal*** (o $n\log n$) consigue incrementos ***lineales*** (o casi) de las tallas manejables
- ***cuadrático*** (polinómico) consigue ***mejoras proporcionales moderadas***
- ***exponencial*** solo logra ***mejoras aditivas despreciables***

Jerarquía de costes computacionales: consecuencias prácticas (II)

Tiempos de ejecución en una máquina que ejecuta 10^9 *pasos por segundo* ($\sim$ *1 GHz), en función del coste del algoritmo y del tamaño del problema* n:

Talla	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
10	3.322 ns	10 ns	33 ns	100 ns	1 μs	1 μs
20	4.322 ns	20 ns	86 ns	400 ns	8 μs	1 ms
30	4.907 ns	30 ns	147 ns	900 ns	27 μs	1 s
40	5.322 ns	40 ns	213 ns	2 μs	64 μs	18.3 min
50	5.644 ns	50 ns	282 ns	3 μs	125 μs	13 días
100	6.644 ns	100 ns	664 ns	10 μs	1 ms	$40 \cdot 10^{12}$ años
1000	10 ns	1 μs	10 μs	1 ms	1 s	
10000	13 ns	10 μs	133 μs	100 ms	16.7 min	
100000	17 ns	100 μs	2 ms	10 s	11.6 días	
1000000	20 ns	1 ms	20 ms	16.7 min	31.7 años	

Índice

Otros conceptos de interés en complejidad computacional

Complejidad espacial

- Concepto de ***posición de memoria***: espacio de almacenamiento ocupado por uno o más datos, cuya extensión es *independiente de la talla* de las instancias del problema considerado

- ***Coste espacial*** de un programa o algoritmo: número de *posiciones de memoria* requeridas para su ejecución

- Todos los conceptos estudiados en el análisis de *coste temporal* son directamente aplicables para el *coste espacial*

Talla de una instancia definida por más de un parámetro

(ej. talla de una matriz de $m \times n$)