Programación

Facultad de Informática

Universidad Politécnica de Valencia

Grupo 1B

Tema 3:
Complejidad computacional

Enrique Vidal Ruiz

E. Vidal – UPV: Septiembre, 2001

Índice

Índice

Introducción

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

¿De que 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***.

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 $\mathbf{10^2}$:

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

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

```
void 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 μs	50 μs	100 μs	200 μs
t_+	10 μs	10 μs	5 μs	10 μs
t_s	1 μs	2 μs	1 μs	0.5 μs
$A1$	100 μs	50 μs	100 μs	200 μs
$A2$	100 μs	100 μs	50 μs	100 μs
$A3$	100 μs	200 μs	100 μs	50 μs

¿que 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:

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

```
void 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);
}
```

```
void 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): $T_{A1} = t_* \quad T_{A2} = t_+ \cdot n \quad T_{A3} = t_s \cdot n^2$

¡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	$16\ \mu s$	$100\ \mu s$	$1\,000\,000\ \mu s$

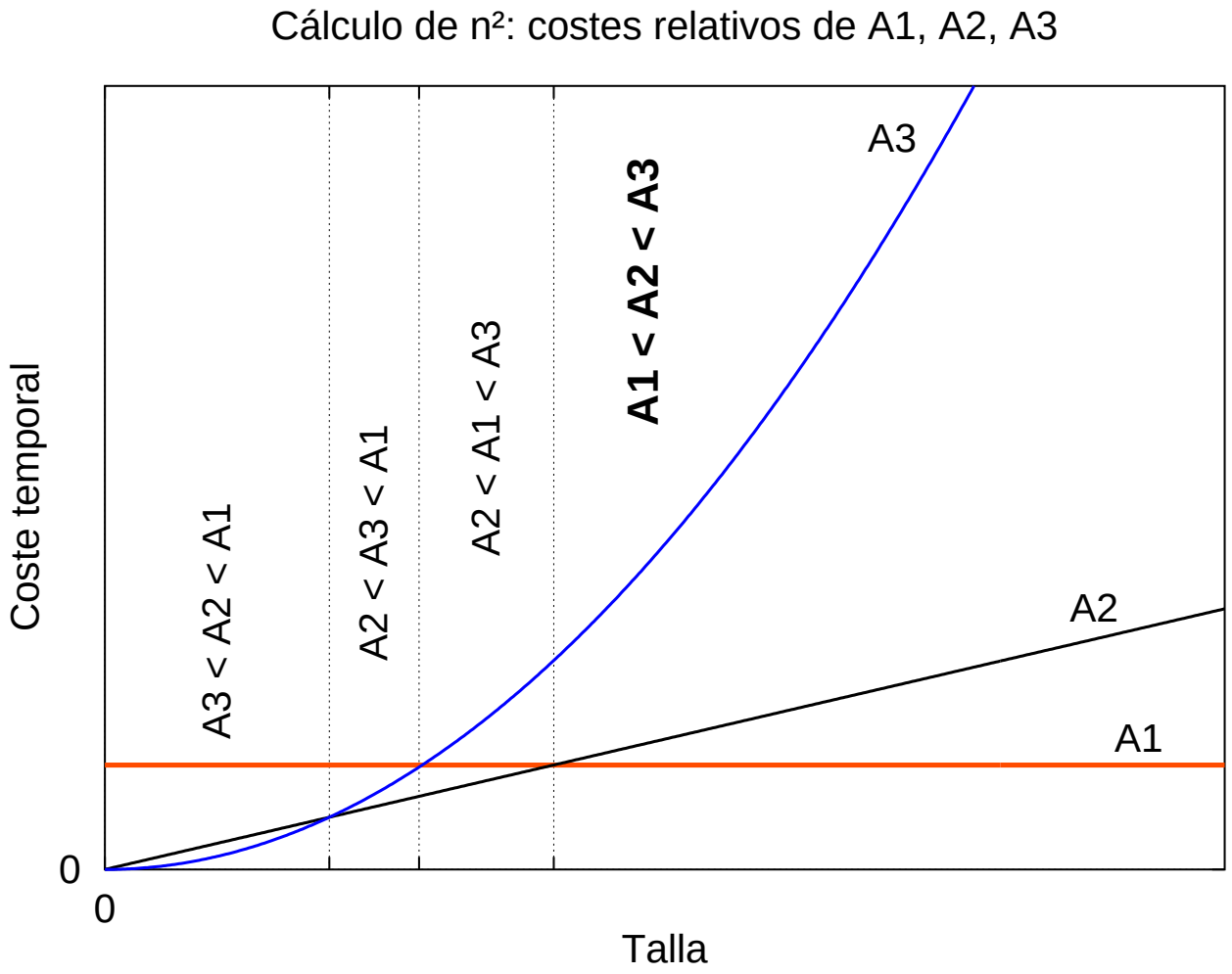
¿que programa es mejor, $A1$, $A2$, $A3$?

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 grandes tallas!

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 asigna *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

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

```
void 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);
}
```

```
void 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);
}
```

$$T_{A1}(n) = c_0, \quad T_{A2}(n) = c_1 + n, \quad T_{A3}(n) = c_1 + n^2 \qquad [c_0 \approx 1,\ c_1 \approx 2]$$

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

```
#include <stdio.h>
#define maxN 1000000
void main(){                    /* invOrd.c */
int  i, n, x, X[maxN];
  n=0; /* Inicializa contador de datos (talla) */

  printf("Teclear datos (fin: ^D)\n");
  /* Lee datos hasta fin de fichero, */
  /* los memoriza y actuliza la talla */
  while (scanf("%d", &x) != EOF) {X[n]=x; n++;}

  /* imprime resultados en orden inverso */
  for (i=n-1; i>=0; i--) printf("%d ", X[i]);
  printf("\n");
}
```

$$\textbf{Talla} = n; \quad T_{invOrd}(n) = c_0 + c_1 \cdot n \ \textbf{PASOS} \qquad [c_0 \approx 2,\ c_1 \approx 2]$$

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

```
#include <stdio.h>
#define maxDatos 1000000
void main() {              /* moda0.c: cálculo (ineficiente) de la moda */
int i,j,n,edad,frec,maxFrec,moda, edades[maxDatos];
  n=0;                          /* Inicaliza contador de datos (talla) */

  printf("Teclear edades, finalizando con ^D\n");
  while (scanf("%d", &edad) != EOF)     /* Lee edades hasta EOF, las */
    if ((edad>=0)) {edades[n] = edad; n++;} /* memoriza y actuliza n */

  maxFrec=0;                          /* Explora n veces edades[] para */
  for (i=0; i<n; i++) { frec=0;          /* determinar cuál es la edad */
    for (j=0; j<n; j++)                   /* que mas se repite (moda) */
      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);
}
```

$$\textbf{Talla} = n; \quad T_{mod}(n) = c_0 + c_1 \cdot n + c_2 \cdot n^2 \ \textbf{PASOS} \quad [c_0 \approx 3,\ c_1 \approx 2,\ c_2 \approx 1]$$

Mas ejemplos de análisis del coste en PASOS:
Cálculo *eficiente* de la moda de una serie de edades

```
#include <stdio.h>
#define maxEdad 150
void main() {                          /* moda.c: cálculo eficiente de la moda */
int  n,edad,maxFrec,moda, frecs[maxEdad];
       /* Inicaliza contador de datos (talla) y vector de frecuencias */
  n=0; 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]++;}

  maxFrec=0;                                /*  Determina  la edad de  */
  for (edad=0; edad<maxEdad; edad++)       /* máxima frecuencia (moda) */
    if (frecs[edad] > maxFrec) {maxFrec=frecs[edad]; moda=edad;}

  printf("Leidos %d datos; Moda=%d (frecuencia=%d)\n", n,moda,maxFrec);
}
```

$$\textbf{Talla} = n; \qquad T_{moda}(n) = c_0 + c_1 \cdot n \ \textbf{PASOS} \qquad [c_0 \approx 2,\ c_1 \approx 1]$$

Índice

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

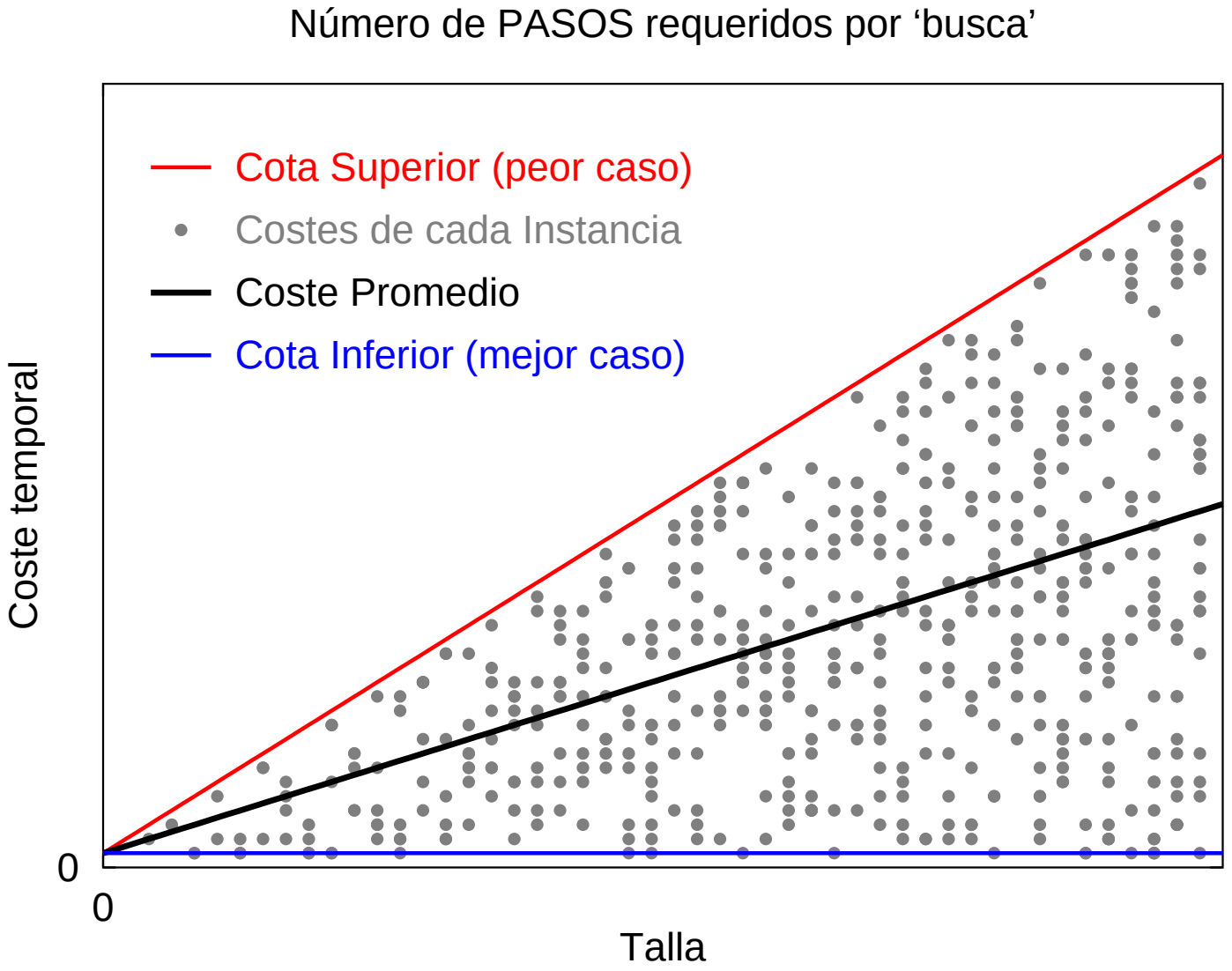
En los ejemplos vistos hasta ahora, todas "instancias" de una talla dada tenían el mismo coste computacional. Pero esto *no* es siempre así. En el siguiente ejemplo, ¿cual es el coste de la función `busca(n)`?:

```
#include <stdio.h>
#define maxN 1000000
int   V[maxN], n=0;     /* Vector donde buscar y su talla */

int busca(int x) { int i;          /* busca x en V[0...n-1] */
  for (i=0; i<n; i++) if (V[i] == x) return i;
  return -1;
}
void main(){ int i, x;                              /* busca.c */
  printf("Teclear datos, fin con ^D)\n");
  while (scanf("%d", &V[n]) == 1) n++; /* lee V[0...n-1] */

  while (printf("Dato a buscar: ") && scanf("%d", &x) == 1)
    {i = busca(x);  printf("Posición de %d: %d\n", x, i);}
}
```

$$Talla = n; \qquad T_{busca}(n) = ???$$

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

Extremos del coste: Casos mejor, peor y promedio



$$T^{b}_{busca}(n) = c_0 \qquad T^{w}_{busca}(n) = c_0 + c_1 \cdot n \qquad T^{m}_{busca}(n) = c_0 + c_1 \cdot n/2$$

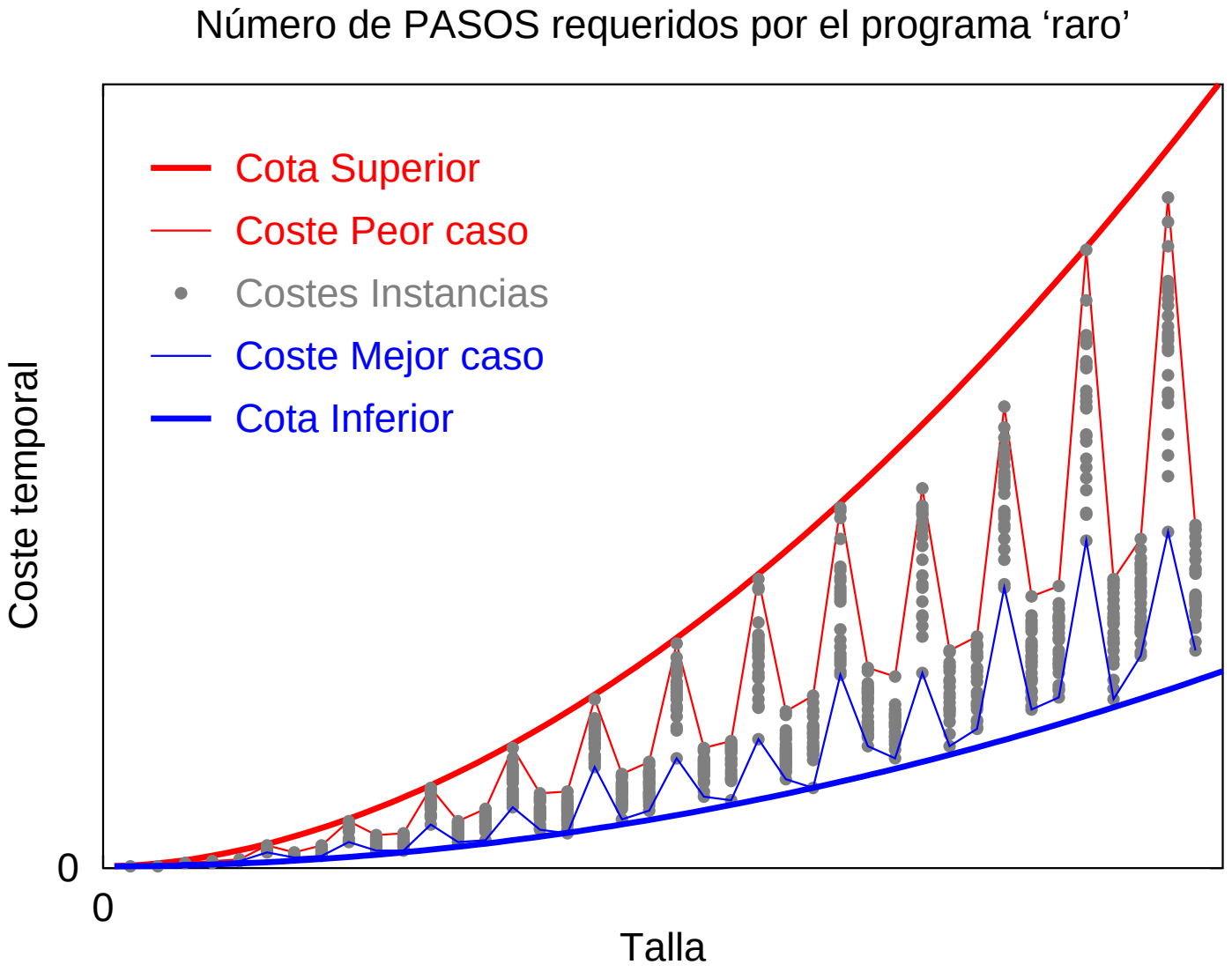
$$[c_0 \approx 2,\ c_1 \approx 1]$$

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

```
#include <stdio.h>
void main(){   /* raro.c: Realiza un extraño proceso sobre un vector */
#define K 64
#define maxN 1000000
int i, j, k, n, x, X[maxN], pasos; n=0; pasos=1;          /* lee datos */
  printf("Teclear datos\n"); while (scanf("%d",&x)!=EOF) {X[n]=x; n++;}

  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]);          /* imprime resultados */
}
```

PASO: bucle de K iteraciones sobre X[i]. 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 $\boldsymbol{cotas\ asintóticas}$*; es decir para* $\boldsymbol{talla}$ $>>>$

Índice

Aspectos esenciales en la determinación de Costes Computacionales

- El coste no se puede expresar por una 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 solo 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 ***mejor caso*** y en el ***peor 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 $f : \mathbb{N} \rightarrow \mathbb{R}^+$ una función de los naturales en los reales positivos. Se define:

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

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

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

Ejemplos

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

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

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

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

$$t(n) = 6 \cdot 2^n + n^2$$

$$t(n) = 6 \cdot 2^n + n^2 \quad \text{¿ Es } \Theta(n^{100}) \text{ ?}$$

Ejemplos

$$
\begin{aligned}
t(n) = 3n+2 \quad & \leq 4n \ \forall n \geq 2 \quad \Rightarrow \ t(n) \in O(n) \\
& \geq 3n \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n) \ ; \quad t(n) \in \Theta(n)
\end{aligned}
$$

$$
\begin{aligned}
t(n) = 100n+6 \quad & \leq 101n \ \forall n \geq 6 \quad \Rightarrow \ t(n) \in O(n) \\
& \geq 100n \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n) \ ; \quad t(n) \in \Theta(n)
\end{aligned}
$$

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

$$
\begin{aligned}
t(n) = n^2+1000n-100 \quad & \leq 2n^2 \ \forall n \geq 520 \ \Rightarrow \ t(n) \in O(n^2) \\
& \geq n^2 \ \forall n \geq 1 \quad \Rightarrow \ t(n) \in \Omega(n^2) \ ; \quad t(n) \in \Theta(n^2)
\end{aligned}
$$

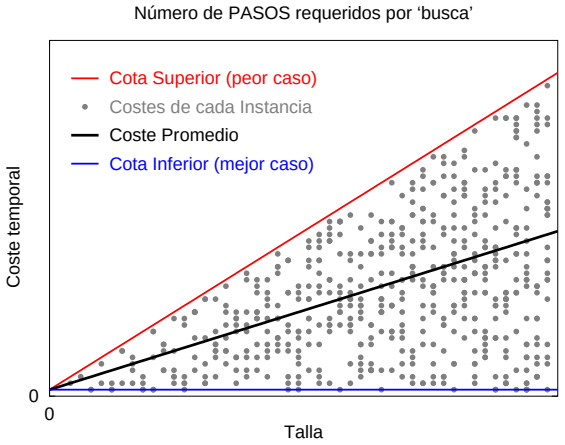
$$
\begin{aligned}
t(n) = 6 \cdot 2^n + n^2 \quad & \leq 7 \cdot 2^n \ \forall n \geq 4 \ \Rightarrow \ t(n) \in O(2^n) \\
& \geq 6 \cdot 2^n \ \forall n \geq 1 \ \Rightarrow \ t(n) \in \Omega(2^n) \ ; \quad t(n) \in \Theta(2^n)
\end{aligned}
$$

$$
\begin{aligned}
t(n) = 6 \cdot 2^n + n^2 \quad & \nexists \, c, n_0 : t(n) \leq c\, n^{100} \ \forall n \geq n_0 \quad \Rightarrow \quad t(n) \notin O(n^{100}) \\
& \geq n^{100} \ \forall n \geq 10^3 \Rightarrow t(n) \in \Omega(n^{100}) ; \ t(n) \notin \Theta(n^{100})
\end{aligned}
$$

Ejemplos de costes en notación asintótica

A1, A2, 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)

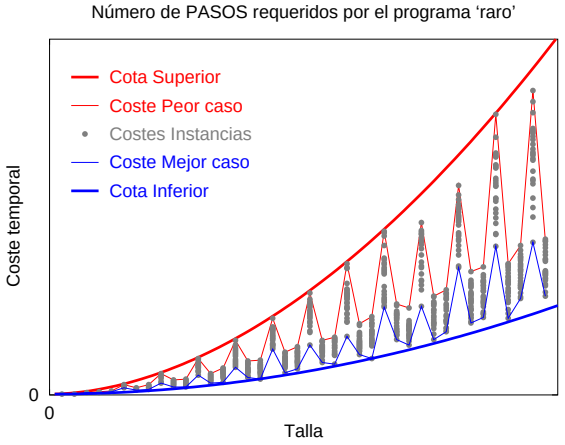


busca:

Peor caso: $O(n)$

Mejor caso: $\Omega(1)$

Promedio: $\Theta(n)$



raro:

Peor caso: $O(n^2)$

Mejor caso: $\Omega(n^2)$

Promedio: $\Theta(n^2)$

Notación asintótica: algunas propiedades útiles

Relación de orden entre órdenes de funciones:

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

Órden de suma de funciones: función dominante

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

Mas propiedades útiles: *Ferri, Albert i Martín: Introducció a l'anàlisi i diseny d'algorismes*, Universitat de València, 1998, pag. 55.

Propiedades útiles para análisis asintóticos

Algunas sumas de series

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

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

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

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

Mas sumas útiles: *Ferri, Albert i Martín: Introducció a l'anàlisi i diseny d'algorismes*, Universitat de València, 1998, pag. 60.

Notación asintótica: mas propiedades útiles

Órdenes de funciones polinómicas

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

Notación asintótica: otras propiedades

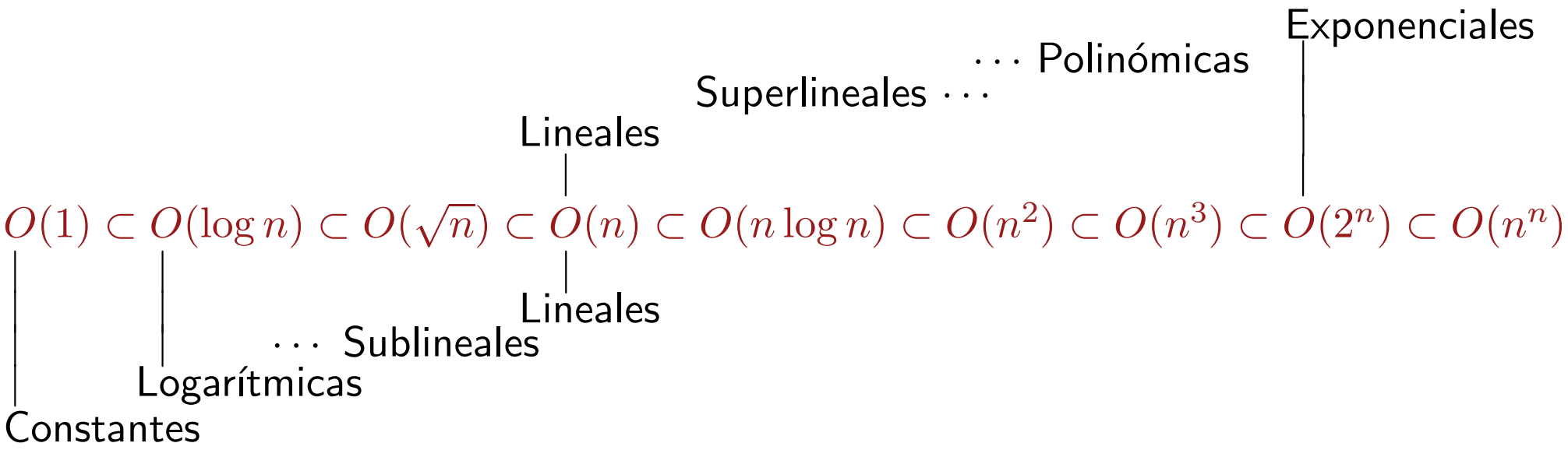
Órdenes de funciones exponenciales, logarítmicas, etc.

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

Ver también: *Ferri, Albert i Martín: Introducció a l'anàlisi i diseny d'algorismes*, Universitat de València, 1998, Apèndix A.

Notación asintótica: Jerarquía de costes computaciones

Algunas relaciones entre órdenes usuales:



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 temporal	1 paso = 1 ms			1 paso = 0.1 ms (10 veces mas rápido)			
	1 seg	1 min	1 hora	1 seg	1 min	1 hora	$n'=f(n)$
$\log n$	$\approx 10^{330}$	$\approx 10^{2\cdot 10^4}$	$\approx 10^{10^{16}}$	$\approx 10^{3\cdot 10^3}$	$\approx 10^{2\cdot 10^5}$	$\approx 10^{10^{17}}$	n^{10}
n	$1\,000$	$6\cdot 10^4$	$3.6\cdot 10^6$	10^4	$6\cdot 10^5$	$3.6\cdot 10^7$	$10\,n$
$n\log n$	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 ···

- ***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***

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 $m \times n$)

Aspectos formales de la Complejidad Computacional (para futuro estudio)

- Modelos Computacionales
  - Máquina RAM
  - Modelos de coste *Uniforme* y *Logarítmico*
  - Otros modelos computacionales: *Recursión* y *Memoria Dinámica*

- Teoría de la Complejidad Computacional
  - Problemas probablemente intratables
  - Equivalencia de problemas: Familias de Complejidad

Índice

Ejercicios propuestos

1. Analizar el coste asintótico de los siguientes programas (vistos en el tema 2): `fechas1.c`, `fechas2.c`, `switch3.c`, `while1.c` (en función de N), `while12.c` (en función del número de datos leídos), `for3.c` (en función de M), `for7.c` (en función de `MAX`), `vect1.c` (en función de `NUM`), `matriz1.c` (en función de $I \cdot J$).

2. Considera los siguientes programas (vistos en el tema 2): `matriz2.c`, `matriz3.c`, `funciones1.c`, `prototipos1.c`, `cadenas3.c`, `cadenas4.c`. ¿Que parámetro o parámetros es (son) adecuado(s) para medir la *talla del problema* en cada uno de estos programas? Analizar el coste asintótico de estos programas en función de las tallas de los problemas correspondientes.

3. En programa `moda0.c` admite una simple mejora que consiste en modificar la inicialización del bucle mas interno para evitar visitar valores que ya han sido tenidos en cuenta antes por el bucle mas externo. Implementar esta variante, analizar su coste asintótico y discutir hasta qué punto logra mejorar la eficiencia.

4. Completar la tabla de *"Máximo tamaño de un problema que puede ser resuelto por diversos algoritmos"* con las siguientes funciones: $O(n\sqrt{n}), O(log\, n^2), O(\sqrt{n}), O(2^{n/10}), O(log^2 n)$.

5. Completar la lista de funciones de la *"Jerarquía de costes computacionales"* incluyendo donde proceda las siguientes funciones: $O(n\sqrt{n}), O(log\, n^2), O(\sqrt{n}), O(2^{n/10}), O(log^2 n)$.