

Técnicas de Programación Avanzadas (TPA)



Universidad
Europea
LAUREATE INTERNATIONAL UNIVERSITIES

Tema 1. Eficiencia de los algoritmos



Tema 1. Eficiencia de los algoritmos

1. Criterios para medir la eficiencia de un algoritmo
2. Tiempo de ejecución
3. Orden de magnitud
4. Reglas para calcular la complejidad del código
5. Complejidad en código recursivo

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA)

Criterios para medir la eficiencia de un algoritmo

- Para un mismo problema se pueden diseñar distintos algoritmos (soluciones) que lo resuelvan correctamente.
- Debe poderse determinar qué algoritmo utilizar.

▪ Ejemplo: ¿Cuál es la mejor ruta?

- La más corta.
- La más rápida.
- La que no tenga peajes.



Universidad Europea

Técnicas de Programación Avanzadas (TPA)

Criterios para medir la eficiencia de un algoritmo

- Para ello se compararán según un criterio determinado. Los más usuales:
 - Legibilidad y facilidad de comprensión, codificación y depuración.
 - Uso eficiente de los recursos: tiempo de ejecución y espacio de memoria.
 - (Otros: reutilización, documentación, portabilidad, etc.)
- Mediremos la eficiencia según el segundo de los criterios (los primeros se presuponen).

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA)



Criterios para medir la eficiencia de un algoritmo

- La **memoria** ya no es un recurso crítico → mediremos la eficiencia de un algoritmo basándonos casi siempre en su tiempo de ejecución.
- El tiempo de ejecución no dependerá de la **potencia** de la máquina donde se ejecuta. Habrá que estudiarlo en función del **algoritmo** utilizado.

Universidad Europea



Tema 1. Eficiencia de los algoritmos

1. *Criterios para medir la eficiencia de un algoritmo*
2. **Tiempo de ejecución**
3. **Orden de magnitud**
4. **Reglas para calcular la complejidad del código**
5. **Complejidad en código recursivo**

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Tiempo de ejecución

- En general, el tiempo de ejecución de un programa depende de varios factores:
 1. Nº de veces que se ejecuta cada instrucción.
 2. Compilador utilizado (una instrucción de alto nivel se puede traducir a lenguaje máquina de diferentes formas).
 3. El ordenador utilizado (puede variar el tiempo de ejecución de cada instrucción máquina).

- Nos basaremos sólo en el primer factor. Los demás no se pueden presuponer.

Tiempo de ejecución

- Realizaremos una **estimación** a priori de la frecuencia de ejecución.

- En el tiempo de ejecución de un caso concreto también influye:
 - Nº de datos de entrada,
 - Estructura de la entrada,
 - Distribución de la entrada.
 - Calidad del código fuente y calidad del código

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Tiempo de ejecución

- Dependiendo de la entrada del problema, pueden darse 3 casos:
 - **Caso Peor:** el tiempo de ejecución es máximo.
 - **Caso Mejor:** el tiempo de ejecución es mínimo.
 - **Caso Medio:** caso más probable. Tiempo de ejecución = media estadística de todos los casos posibles.
- Para probar la eficiencia de un algoritmo usaremos como medida el tiempo de ejecución en el peor caso: $T(n)$.

Tiempo de ejecución

- **Ejemplo:**
 - ¿Cuántas veces se ejecuta cada instrucción si el tamaño N de la lista es 4? ¿Y si es 6?

```

boolean buscar (TElemento item, Lista L) {
    {Pre: N>0}
    {Post: Buscar = ∃i:1<=i<=N: L[i]=item}
    1 boolean encontrado = false;
    2 int cont = 0;
    3 while ((!encontrado) && (cont < L.longitud())){
        cont++;
    }
}

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA) 

Tiempo de ejecución

- En el peor de los casos:

Instrucción	Código	Ejecuciones
1	<code>boolean encontrado = false;</code>	1
2	<code>int cont = 0;</code>	1
3	<code>while ((!encontrado) && (cont < L.longitud()))</code>	n+1
4	<code>cont++;</code>	n
5	<code>encontrado = L[cont] == item;</code>	n
6	<code>return encontrado;</code>	1
Total		3n+4

- A medida que se incrementa el tamaño n de la lista, el valor de $T(n)$ crece de manera proporcional a $n \rightarrow T(n)$ tiene un orden de magnitud de n .

Universidad Europea

Técnicas de Programación Avanzadas (TPA) 

Tiempo de ejecución

- $T(n) \in O(f(n))$, si \exists ctes c y n_0 tal que $T(n) \leq c \cdot f(n), \forall n \geq n_0$
- $T(n) \in \Omega(g(n))$, si \exists ctes c y n_0 tal que $T(n) \geq c \cdot g(n), \forall n \geq n_0$
- $T(n) \in \Theta(h(n))$, si y sólo $T(n) \in O(h(n))$ y $T(n) \in \Omega(h(n))$
- Si $T(n) \in O(f(n)) \rightarrow T(n)$ nunca crece más rápido que $f(n)$.
- Si $T(n) \in \Omega(g(n)) \rightarrow T(n)$ crece, al menos, al mismo ritmo que $g(n)$
- Si $T(n) \in \Theta(h(n)) \rightarrow$ representa exactamente la tasa de



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**



Tema 1. Eficiencia de los algoritmos

1. *Criterios para medir la eficiencia de un algoritmo*
2. *Tiempo de ejecución*
3. **Orden de magnitud**
4. **Reglas para calcular la complejidad del código**
5. **Complejidad en código recursivo**

Universidad Europea



Técnicas de Programación Avanzadas (TPA)

Tiempo de ejecución

- Reglas para el cálculo de complejidad de un algoritmo
 1. Si $T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$, entonces:
 - a) $T_1(n) + T_2(n) \in \text{Max} (O(f(n)), O(g(n)))$
 - b) $T_1(n) * T_2(n) \in O (f(n) * g(n))$
 2. Si $T(x)$ es un polinomio de grado $g \rightarrow T(x) \in O(x^g)$

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA)

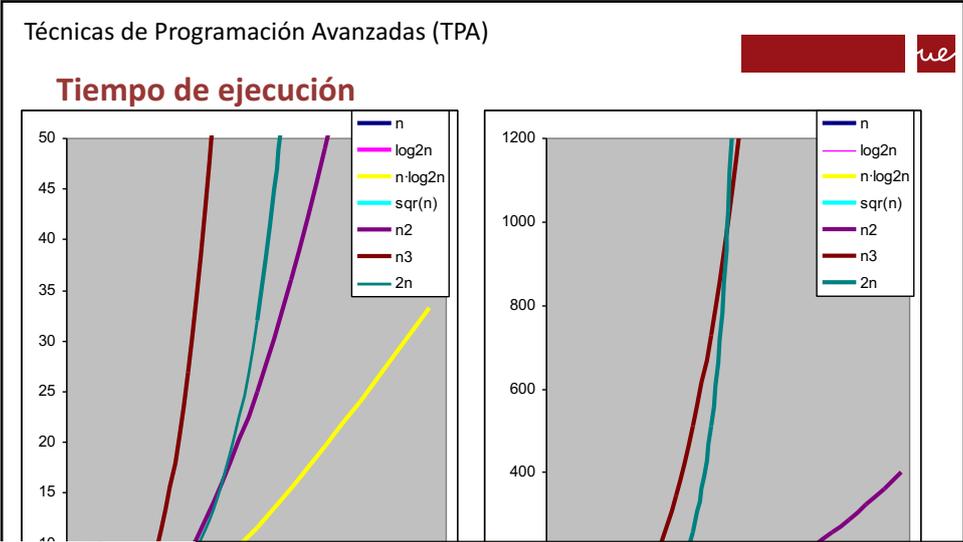
Tiempo de ejecución

- Definiremos una relación entre los distintos órdenes de magnitud de las funciones $f(n)$ (siempre válida a partir de un n_0 determinado):
 $O(1) < O(\log n) < O(n) < O(n \cdot \log n) < O(n^2) < O(n^3) < \dots < O(2^n) < O(n!) < O(n^n)$
 $\forall n \geq n_0$
- Esta notación (O) indica un límite superior de $f(n)$.
Ejemplo: si $f(n)$ es $O(n^2) \rightarrow$ también es $O(n^3), O(2^n) \dots$
- Al hallar la complejidad se desprecian las ctes de proporcionalidad.
 - Inconveniente: no muy bueno para valores pequeños de la entrada.

Ejemplo:

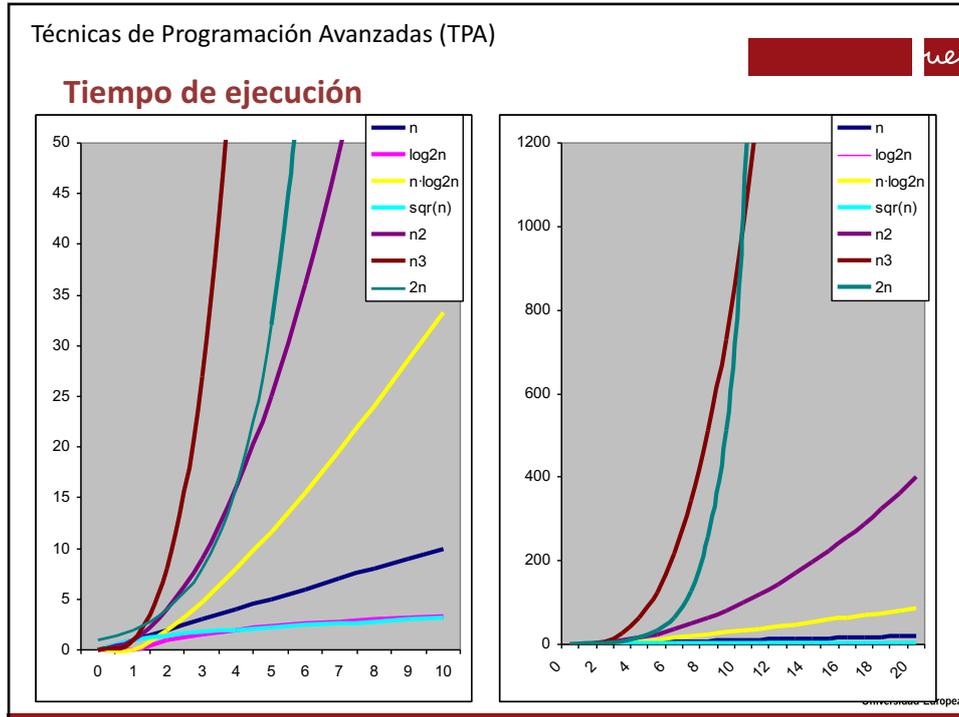
$T_1(n) = 5n^3 = O(n^3)$
 $T_2(n) = 100n^2 = O(n^2)$
 $5n^3 / 100n^2 = n/20 \rightarrow$ para $n < 20, T_1$ es mejor.

Universidad Europea



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70



Tema 1. Eficiencia de los algoritmos

1. *Criterios para medir la eficiencia de un algoritmo*
2. *Tiempo de ejecución*
3. *Orden de magnitud*
4. **Reglas para calcular la complejidad del código**
5. **Complejidad en código recursivo**

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA)

Reglas para calcular la complejidad del código

- ¿Cómo se traduce todo esto a la hora de calcular la complejidad de nuestro código?
 - Aplicaremos diferentes herramientas
 - Dependerán de los elementos o estrategias de programación utilizadas
- **Sentencias simples**
 - Cada instrucción máquina se ejecuta en un tiempo constante K_i .
 - Escogemos como unidad el mayor tiempo K_i .
 - Cada sentencia se considera de $O(1)$.
 - Válido para manipulaciones (asignaciones, comparaciones, etc.) sobre datos simples.
- **Composición secuencial:**
 - Para una secuencia de sentencias \rightarrow el máximo de todas ellas.

Universidad Europea

Técnicas de Programación Avanzadas (TPA)

Reglas para calcular la complejidad del código

- **Selección Condicional (If then else, Case):**
 - De entre todos los posibles caminos \rightarrow el máximo en complejidad.
- **Repeticiones:**
 - Σ (complej. cuerpo bucle + complej. condición terminación)
- **Llamadas a otros módulos (procedimientos o funciones):**
 - Como si fuesen otras partes del programa

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Reglas para calcular la complejidad del código

▪ Ejemplos:

– `x = x + 1` \longrightarrow $O(1)$

– `for (i=1;i<=N;i++) {
 X = X + 1;
}` $O(n)$

– `for (i=1;i<=N;i++) {
 for (j=1;j<=N;j++) {
 x = x + 1;
 }
}` $O(n^2)$

Reglas para calcular la complejidad del código

▪ Ejemplo:

```
void burbuja (TElemento A[]) {
    for (int i = 1; i <= A.length; i++) {
        for (j = A.length; j > i; j--) {
            if (A[j-1] > A[j]) {
                TElemento temp = A[j-1];
                A[j-1] = A[j];
                A[j] = temp;
            }
        }
    }
}
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

Técnicas de Programación Avanzadas (TPA) 

Reglas para calcular la complejidad del código

▪ **Fórmulas habituales:**

$$\sum_{i=1}^n 1 = n$$

$$\sum_{i=1}^n i^{-1} = \sum_{i=1}^n \frac{1}{i} = \log(n)$$

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

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Universidad Europea

Técnicas de Programación Avanzadas (TPA) 

Reglas para calcular la complejidad del código

- **Recursividad**
 1. Reducción por **SUSTRACCIÓN**
 2. Reducción por **DIVISIÓN**
- **Reducción por sustracción**

Complejidad del caso base

Complejidad del caso general

$$T(n) = \begin{cases} c_0 n^{k_0} & \text{si } 0 \leq n < b \\ a \cdot T(n-b) + c_1 n^k & \text{si } n \geq b \end{cases}$$

Parte Parte NO

a: número de llamadas recursivas idénticas en cada invocación del programa

n-b: tamaño de los subproblemas generados



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Reglas para calcular la complejidad del código

- **Recursividad**

1. Reducción por SUSTRACCIÓN
2. Reducción por DIVISIÓN

- **Reducción por división**

$$T(n) = \begin{cases} c_0 n^{k_0} & \text{si } 0 \leq n < b \\ a \cdot T(n/b) + c_1 n^k & \text{si } n \geq b \end{cases}$$

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

$$T(n) \in \Theta(n^k \log_b n) \text{ si } a = b^k$$

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

a: número de llamadas recursivas idénticas en cada invocación del programa

n-b: tamaño de los subproblemas generados

n^k: coste de las instrucciones que no son llamadas recursivas



**Universidad
Europea**

LAUREATE INTERNATIONAL UNIVERSITIES

Madrid Valencia Canarias

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99