# Diseño y Análisis de Algoritmos

TEMA 1. EFICIENCIA ALGORÍTMICA

Universidad
Rey Juan Carlos

# Tema 1. Eficiencia algorítmica

## Comparación de algoritmos

- Un **problema** se puede resolver con **varios** algoritmos ¿Cómo seleccionar el **mejor**?
- **Eficiencia**. Medida de los recursos que emplea un algoritmo en su ejecución
- Recursos **computacionales**: tiempo (*ejecución*), espacio (memoria), nº procesadores (arquitecturas paralelas), …
- Recursos **no computacionales**: dificultad de implementación, disponibilidad de bibliotecas, …

## Comparación de algoritmos

La comparación en tiempo depende de:

- Datos de **entrada**
- **Calidad** del código generado por el compilador
- Rapidez del **procesador**
- **Complejidad** intrínseca del algoritmo

Estudios sobre el tiempo:

- **Teórico** (a priori): función que acote el tiempo de ejecución para unos valores de los datos de entrada
- **Real** (a posteriori): tiempo de ejecución para una determinada entrada y en un ordenador concreto

# Principio de Invarianza

- Dado un algoritmo y dos implementaciones $I_1$ e $I_2$, que tardan $T_1(n)$ y $T_2(n)$ respectivamente, existe una constante real positiva $c \in \mathbb{R}^+$ y un natural $n_0 \in \mathbb{N}$ tales que:

$$\forall n \geq n_0 \quad T_1(n) \leq c \cdot T_2(n)$$

- El tiempo de ejecución de dos implementaciones distintas no va a diferir más que en una cte. multiplicativa

## Comparación de algoritmos

### Estudio real

- Medida del **tiempo** de ejecución de un algoritmo

### Estudio teórico

- Estimación del **comportamiento** de un algoritmo
- **Independiente** del ordenador
- No requiere una **ejecución**

## Comparación de algoritmos

Complejidad Algorítmica:

- Determina la **eficiencia** de un algoritmo
- No proporciona medidas **absolutas** (p.e. segundos) sino relativas al tamaño del problema
- Es **independiente** del ordenador en el que se ejecute el algoritmo

T(*n*): tiempo empleado para ejecutar el algoritmo con una entrada de tamaño *n*

Dada una entrada de tamaño *n*, no se mide en unidades de **tiempo** (seg) sino en "**pasos**" (#instrucciones)

## Ejemplos de complejidad de tiempo

**Problema 1:** sumar las componentes de un vector

Dato relevante: nº de componentes

```
...
para j <-1 hasta n hacer
        suma <- suma + c[j]
fpara
...
```

nº ejecuciones cuerpo bucle: $n$ => $T(n) = k * n$

# **Problema 2:** producto de 2 matrices *n* x *n*

Dato relevante: orden de las matrices

```
...
para i<-1 hasta n hacer
      para j <- 1 hasta n hacer
            c[i,j] <- 0;
            para k <- 1 hasta n hacer
                  c[i,j] <- c[i,j] + a[i,k] * b[k,j]
            fpara
      fpara
fpara
...
```

$T(n) = k * n * n * n = k * n^3$

## Ejemplos de complejidad de tiempo

# **Problema 3:** ordenar las componentes de un vector comparando claves

Dato relevante: nº de componentes

```
...
para i <- 1 hasta n-1 hacer
      para j <- 1 hasta  n-i hacer
            si c[j] > c[j+1]  entonces
                  intercambio (c[i], c[i+1])
            fsi
      fpara
fpara
...
```

Operaciones elementales

Para cada valor de *i*, *j* ejecuciones del bucle más interno

*i* = 1 , *j* = *n* -1

*i* = 2, *j* = *n* -2

*i* = 3, *j* = *n* -3

...

*i* = *n* -1, *j* = 1

$T(n) \approx (n-1) + (n-2) + (n-3) + \ldots + 1 = (n-1)\, n\, /2$

## Ejemplos de complejidad de tiempo

**Problema 4:** Búsqueda secuencial. Las operaciones no se ejecutan el mismo número de veces para cualquier *n*

```
...
i <- 1
mientras i<n y c[i] != elem hacer
      i <- i+1
fmientras
si i > n entonces
      i <- 0
fsi
...
```

T(*n*) varia si se encuentra en: 1, *n*/2, *n*, … pasos

# Tema 1. Eficiencia algorítmica

## Medidas asintóticas

**Tmax(n):** complejidad en el caso peor. Tiempo máximo para una entrada de tamaño *n*

**Tmin(n):** complejidad en el caso mejor. Tiempo mínimo para una entrada de tamaño *n*

**Tmed(n):** complejidad en el caso medio. Tiempo medio para una entrada de tamaño *n*

- Se suele suponer que toda las secuencias de entrada son equiprobables ¿Cuál utilizar?
  - Tmáx(n)
  - Caso mejor: poco representativo
  - Caso medio: puede resultar difícil de calcular

# Tema 1. Eficiencia algorítmica

## Medidas asintóticas

- Definir clases de equivalencia correspondientes a funciones que "**crecen de la misma forma**"
- **Asintótico**: valores de los datos suficientemente *grandes* ya que, para valores *pequeños*, la diferencia de eficiencia entre algoritmos puede ser marginal

Medidas del **comportamiento asintótico** de la complejidad

| | |
|---|---|
| Θ (theta) | orden exacto de la función |
| O (o mayúscula) | cota superior |
| Ω (omega) | cota inferior |

Es una cota superior

- Dada una función $f$, estudiamos funciones $g$ que, a lo sumo, crezcan **tan deprisa** como $f$
- Al conjunto de esas funciones se les llama **cota superior** de $f$ y es $O(f)$
- Conociendo $O(f)$ de un algoritmo, se puede asegurar que en ningún caso el tiempo será de un **orden superior** al de la cota

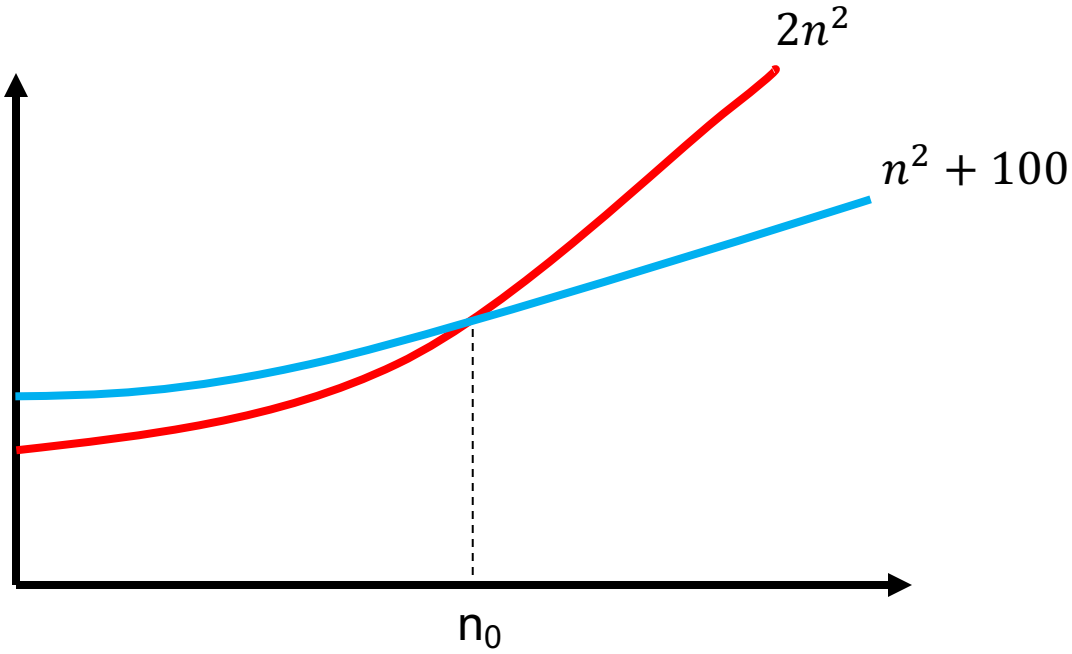
**Definición**: Sean $f, g: \mathbb{Z}^+ \to \mathbb{R}^+$, se dice que $f \in O(g)$ si existen dos constantes $n_0 \in \mathbb{N}$ y $\lambda \in \mathbb{R}^+$ tales que:

$$\forall n \geq n_o \qquad f(n) \leq \lambda g(n)$$

- La función $f$ no crece más deprisa que ninguna función proporcional a $g$
- Si $f(n) \in O(g(n))$ se dice que $f(n)$ está en O de $g(n)$ para todo $n$ suficientemente grande

# Tema 1. Eficiencia algorítmica

## Medidas asintóticas. Notación O




- $n^2 + 100$ está inicialmente por encima de $2n^2$, pero para $n > 10$ se da que $n^2 + 100 < 2n^2$
- Si tomamos $f(n) = n^2 + 100$ entonces $f(n) \in O(n^2)$, donde: $n_0 = 10, \lambda = 2, g(n) = n^2$

Una cota superior siempre se puede estimar al alza

$(5n + 3) \in O(n^2)$

$(5n + 3) \in O(n)$ es más preciso

**Propiedades de O**

1. Para cualquier $f$ se tiene que $f \in O(f)$ (reflexibilidad)
2. $f \in O(g) \Rightarrow O(f) \subset O(g)$
3. $O(f) = O(g) \Leftrightarrow f \in O(g)$ y $g \in O(f)$
4. Si $f \in O(g)$ y $g \in O(h) \Rightarrow f \in O(h)$ (transitividad)

5. Si $f \in O(g) \Rightarrow f \in O(kg) \;\; \forall\, k \in R^+$ (escalabilidad)

consecuencia: $O(\log_a n) = O(\log_b n) = O(\log n)$

*no hace falta expresar la base*

6. Si $f \in O(g)$ y $f \in O(h) \Rightarrow f \in O(\min(g,h))$

7. Regla de la suma o regla del máximo:

Si $f_1 \in O(g_1)$ y $f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(\max(g_1, g_2))$

siendo $\max(g_1, g_2)(n) = \max(g_1(n), g_2(n))$

Generalizando:

Si $f_i \in O(f)$ para todo $i=1,\ldots k$ entonces

$$c_1 f_1 + \ldots + c_k f_k \in O(f)$$

Si $p_k(n)$ es un polinomio de grado $k$ entonces

$$p_k(n) \in O(n^k)$$

8. **Regla del producto**:

$$\text{Si } f_1 \in O(g_1) \text{ y } f_2 \in O(g_2) \Rightarrow f_1 . f_2 \in O(g_1 . g_2)$$

Consecuencia: Si $p < q$ entonces $O(n^p) \subset O(n^q)$

9. Si existe $\lim_{n\to\infty} f(n) / g(n) = k$ tenemos:

a) Si $k \neq 0$ y $k < \infty$ entonces $O(f) = O(g)$

b) Si $k = 0$ entonces $f \in O(g)$, es decir, $O(f) \subset O(g)$, pero se verifica que $g \notin O(f)$

## Cálculo de la eficiencia

- Sentencias simples. Instrucciones de lectura, escritura, asignación, …
  - Tiempo **constante**
- Bloques de sentencias
  - **Suma** de tiempos y aplicar la regla del máximo
- Sentencias condicionales
  - **Máximo** entre el bloque `if` y bloque `else`

## Cálculo de la eficiencia

- Bucles
  - **Suma** de los tiempos de cada iteración (incluido la evaluación de la condición
  - Iteraciones idénticas => número de iteraciones **multiplicado** por el tiempo de una iteración

- Llamadas a funciones
  - Equivalente a la **complejidad** de la propia **función**

- Funciones recursivas
  - Método de **sustitución**
  - **Árbol** de recursividad
  - **Expansión** de recurrencias
  - **Ecuación característica**

## Cálculo de la eficiencia

- Recurrencias de divide y vencerás

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k \text{ con } a \geq 1, b \geq 2, k \geq 0, c > 0$$

$$T(n) = \begin{cases} O(n^k), & a < b^k \\ O(n^k \log_b n) & a = b^k \\ O(n^{\log_b a}) & a > b^k \end{cases}$$

- a: sub-problemas generados
- b: divisor del problema
- k: complejidad del método de combinación

# Ordenes de eficiencia más habituales

- Suponiendo 1micro segundo por operación elemental

| n | O(log n) | O(n) | O(n log n) | $O(n^2)$ | $O(2^n)$ | O(n!) |
|---|---|---|---|---|---|---|
| 10 | 3 µs | 10 µs | 30 µs | 0.1 ms | 1 ms | 4s |
| 25 | 5 µs | 25 µs | 0.1 ms | 0.6 ms | 33 s | $10^{11}$ años |
| 50 | 6µ | 50µs | 0.3ms | 2.5ms | 36 años | ... |
| 100 | 7 µs | 100 µs | 0.7 ms | 10 ms | $10^{17}$ años | ... |
| 1000 | 10 µs | 1 ms | 10 ms | 1s | ... | ... |
| 10000 | 13 µs | 10 ms | 0.1 s | 100 s | ... | ... |
| 100000 | 17 µs | 100 ms | 1.7 s | 3 horas | ... | ... |
| 1000000 | 20 µs | 1s | 20 s | 12 días | ... | ... |

*$O(1) << O(\log n) << O(n) << O(n \log n) << O(n^2) << O(n^3) << \ldots << O(2^n) << O(n!)$*

## Medidas asintóticas. Notación Ω

Es una cota inferior

- Dada una función $f$, estudiamos funciones $g$ que a lo sumo crezcan tan lentamente como $f$
- Al conjunto de esas funciones se les llama cota inferior de $f$ y es $\Omega(f)$
- Conociendo $\Omega(f)$ de un algoritmo, se puede asegurar que en ningún caso el tiempo será de un orden inferior al de la cota

# Tema 1. Eficiencia algorítmica

## Medidas asintóticas. Notación Ω

Sean $f, g: \mathbb{Z}^+ \to \mathbb{R}^+$, se dice que $f \in \Omega(g)$ si existen dos constantes $n_0 \in \mathbb{N}$ y $\lambda \in \mathbb{R}^+$ tales que:

$$\forall n \geq n_o \qquad f(n) \geq \lambda g(n)$$

- La función $f$ crece más deprisa que alguna función proporcional a $g$
- Si $f(n) \in \Omega(g(n))$ se dice que $f(n)$ está en $\Omega$ de $g(n)$ para todo $n$ suficientemente grande
- La función $f$ necesita para su ejecución un tiempo mínimo dado por la función $g$

## Medidas asintóticas. Notación Ω

- Ejemplo de cota inferior

$5n^2 \in \Omega(n^2)$ ya que para $n \geq 0$, $5n^2 \geq n^2$

$\lambda = 1$ y $n_0 = 0$, $5n^2 \in \Omega(n^2)$

- Una cota inferior siempre se puede estimar a la baja

$(n^3) \in \Omega(n^2)$, ya que si $n \geq 1$ $n^3 \geq n^2$

$(5n+3) \in \Omega(n)$ es más preciso

**Propiedades de Ω**

1. Para cualquier función *f* se tiene que $f \in \Omega(f)$ (reflexibilidad)

2. $f \in \Omega(g) \Rightarrow \Omega(f) \subset \Omega(g)$

3. $\Omega(f) = \Omega(g) \Leftrightarrow f \in \Omega(g)$ y $g \in \Omega(f)$

4. Si $f \in \Omega(g)$ y $g \in \Omega(h) \Rightarrow f \in \Omega(h)$ (transitividad)

5. Si $f \in \Omega(g) \Rightarrow f \in \Omega(kg) \ \forall\, k \in R^+$ (escalabilidad)

Consecuencia: $\Omega(\log_a n) = \Omega(\log_b n) = \Omega(\log n)$

no hace falta expresar la base

6. Si $f \in \Omega(g)$ y $f \in \Omega(h) \Rightarrow f \in \Omega(\min(g,h))$

7. **Regla de la suma** o regla del máximo:

Si $f_1 \in \Omega(g_1)$ y $f_2 \in \Omega(g_2) \Rightarrow f_1 + f_2 \in \Omega(\text{máx}\ (g_1, g_2))$

siendo $\text{máx}\ (g_1, g_2)(n) = \text{máx}\ (g_1(n), g_2(n))$

8. **Regla del producto**:

Si $f_1 \in \Omega(g_1)$ y $f_2 \in \Omega(g_2) \Rightarrow f_1 \cdot f_2 \in \Omega(g_1 \cdot g_2)$

9. Si existe $\lim_{n\to\infty} f(n) / g(n) = k$ tenemos:

a) Si $k \neq 0$ y $k < \infty$ entonces $\Omega(f) = \Omega(g)$

b) Si $k = 0$ entonces $g \in \Omega(f)$, es decir, $\Omega(g) \subset \Omega(f)$, pero se verifica que $f \notin \Omega(g)$

## Regla de la dualidad

$$f \in O(g) \Leftrightarrow g \in \Omega(f)$$

$$f(n) \leq \lambda\, g(n) \Leftrightarrow g(n) \geq (1/\lambda)\, f(n)$$

# Tema 1. Eficiencia algorítmica

## Medidas asintóticas. Notación $\Theta$

- Es el **orden exacto**
- Dada una función $f$, estudiamos funciones $g$ que crecen asintóticamente de la misma forma
- Al conjunto de esas funciones se les llama orden exacto de $f$ y es $\Theta(f)$
- Conociendo $\Theta(f)$ de un algoritmo, se puede asegurar que el tiempo es de dicho orden

## Medidas asintóticas. Notación Θ

**Definición**: Sean $f, g: \mathbb{Z}^+ \rightarrow \mathbb{R}^+$, se dice que $f \in \Theta(g)$ si $f \in O(g) \cap \Omega(g)$; es decir f pertenece tanto a $O(g)$ como a $\Omega(g)$

$f \in \Theta(g) \rightarrow f$ está en el orden exacto de $f$

$\Theta(g)$ es el conjunto de funciones de complejidad $f(n)$ para las que si existen constantes $n_0 \in \mathbb{N}$ y $c, d \in \mathbb{R}^+$ tales que:

$$\forall n \geq n_o \quad cg(n) \leq f(n) \leq dg(n)$$

Ejemplos:

$g(n)=5n^2 + 100n + 3 \quad \in \Theta(n^2)$

$t(n) = (n\text{-}1)\, n\, /2 = n^2/2 - n/2 \quad \in \Theta(n^2)$

**Propiedades de Θ**

1. Para cualquier función *f* se tiene que $f \in \Theta(f)$ (reflexibilidad)

2. $f \in \Theta(g) \Rightarrow \Theta(f) = \Theta(g)$

3. $\Theta(f) = \Theta(g) \Leftrightarrow f \in \Theta(g)$ y $g \in \Theta(f)$

4. Si $f \in \Theta(g)$ y $g \in \Theta(h) \Rightarrow f \in \Theta(h)$ (transitividad)

5. Si $f \in \Theta(g) \Rightarrow f \in \Theta(kg) \ \forall\ k \in R^+$ (escalabilidad)

6. **Regla de la suma** o regla del máximo:

Si $f_1 \in \Theta(g_1)$ y $f_2 \in \Theta(g_2) \Rightarrow f_1 + f_2 \in \Theta(\text{máx}\ (g_1, g_2))$

siendo $\text{máx}\ (g_1, g_2)(n) = \text{máx}\ (g_1(n), g_2(n))$

7. **Regla del producto**:

Si $f_1 \in \Theta(g_1)$ y $f_2 \in \Theta(g_2) \Rightarrow f_1 . f_2 \in \Theta(g_1 . g_2)$

8. Si existe $\lim_{n\to\infty} f(n) / g(n) = k$ tenemos:

a) Si $k \neq 0$ y $k < \infty$ entonces $\Theta(f) = \Theta(g)$

b) Si $k = 0$ entonces $\Theta(f) \neq \Theta(g)$