




## Algorítmica y Lenguajes de Programación

---

### Eficiencia y notación asintótica (i)



### Eficiencia y notación asintótica. Introducción

---

- Para resolver un **problema** pueden existir **varios algoritmos**. Por tanto, es lógico elegir el **"mejor"**.
- Si el problema es sencillo o no hay que resolver muchos casos se podría elegir el más "fácil".
- Si el problema es complejo o existen muchos casos habría que elegir el algoritmo que **menos recursos utilice**.
- Los recursos más importantes son el **tiempo de ejecución** y el **espacio de almacenamiento**. Generalmente, el más importante es el tiempo.
- Al hablar de la **eficiencia** de un algoritmo nos referiremos a lo **"rápido"** que se ejecuta.
- **La eficiencia de un algoritmo dependerá, en general, del "tamaño" de los datos de entrada.**

## Eficiencia y notación asintótica. Ejemplares y problemas


- A la hora de estudiar la eficiencia de distintos algoritmos es necesario distinguir **problemas** de **ejemplares**.
- Un problema sería una "proposición dirigida a averiguar el modo de obtener un resultado cuando ciertos datos son conocidos."
- Un ejemplar, en cambio, es un "individuo de una especie o género."
- Ejemplos de **problemas** serían: "**multiplicar dos números enteros**" o "**resolver una ecuación de segundo grado**".
- **Ejemplares** de dichos problemas serían: (425,7) o (1,1,2). Es decir:  $425 \cdot 7$  y  $x^2 + x + 2$ .
- Un **problema** dado puede tener un número **infinito de ejemplares** (multiplicar dos números enteros) o **finito** (jugar una partida de ajedrez).
- **Un algoritmo que afirma resolver un problema debe resolver todos los ejemplares del mismo. Si hay un solo ejemplar que no sea resuelto por el algoritmo el algoritmo no es correcto.**

3

## Eficiencia y notación asintótica. Casos mejor, peor y medio

- **No todos los ejemplares de un problema son iguales.** En el caso de la "multiplicación de dos números enteros" no es lo mismo multiplicar (a y b  $\neq$  0) que a·0.
- Al no ser todos los ejemplares iguales, **la eficiencia del algoritmo puede variar en función del ejemplar o del grupo de ejemplares.**
- A la hora de analizar un algoritmo es necesario saber que pueden darse tres tipos de ejemplares o casos:
  - **Caso mejor:** se trata de aquellos ejemplares del problema en los que el algoritmo es más eficiente; **por ejemplo: multiplicar un número por cero, insertar en una lista vacía, ordenar un vector que ya está ordenado, etc.** Generalmente no nos interesa.
  - **Caso peor:** se trata de aquellos ejemplares del problema en los que el algoritmo es menos eficiente (no siempre existe el caso peor). **Ejemplos: insertar al final de una lista, ordenar un vector que está ordenado en orden inverso, etc.** Nos interesa mucho.
  - **Caso medio:** se trata del resto de ejemplares del problema. **Por ejemplo: multiplicar dos números enteros distintos de cero, insertar en a de una lista que no sea el principio ni el final, ordenar un vector que no está ordenado ni en orden directo ni inverso, etc.** Es el caso que más nos debería preocupar puesto que será el más habitual, sin embargo no siempre se puede calcular (habría que saber cuáles son las probabilidades de los distintos ejemplares).


4



## Eficiencia y notación asintótica. Operaciones elementales

- **Una operación elemental es aquella cuyo tiempo de ejecución tiene una cota superior constante que sólo depende de su implementación** (por ejemplo: el ordenador o el lenguaje de programación utilizado).
- **Las operaciones elementales son consideradas de coste unitario.**
- A la hora de analizar un algoritmo importarán, por tanto, el número de operaciones elementales que precisa.
- **La decisión de determinar que una operación determinada es de coste unitario dependerá de los ejemplares del problema que la utilice.** Por ejemplo, en la mayor parte de los casos la suma se considera de coste unitario puesto que al operar con los números que se manejan habitualmente en un ordenador los tiempos que emplea la suma son similares; sin embargo, en caso de manejar números muy grandes la suma no tendrá un coste unitario puesto que tardará más cuanto más grandes sean los números a sumar.

5



## Eficiencia y notación asintótica. Tamaño de los ejemplares

- **El tamaño de un ejemplar sería el número de *bits* necesarios para representarlo en el ordenador.**
- Sin embargo, no es necesario llegar a ese nivel de detalle; basta con **determinar algún parámetro que nos permita dirimir entre dos ejemplares cuál es el "mayor"**. Por ejemplo:
  - Si tenemos un algoritmo que trabaja con **vectores** parece obvio que cuantas más componentes tiene un vector "mayor" será; así pues, en este caso **el tamaño será el número de componentes**.
  - Si escribimos un algoritmo que permite **sumar números** de cualquier número de cifras **el tamaño de los números dependerá del número de cifras que contengan**.
  - Un algoritmo para multiplicar **matrices** considerará como **tamaño al producto del número de filas por el número de columnas**.
- En resumen, para cada problema es necesario analizar la naturaleza de los datos para determinar qué parámetro define el tamaño de los ejemplares.
- **La eficiencia de los algoritmos siempre vendrá dada en función de dichos tamaños:**
  - Algoritmos con vectores en función del número de componentes.
  - Algoritmos con matrices en función del número de filas y columnas.
  - Etc.

6

## Eficiencia y notación asintótica. Midiendo la eficiencia (i)

- Tenemos que nos interesa saber qué algoritmo utiliza más eficientemente los recursos, en general, el tiempo de ejecución.
- Esta eficiencia en tiempo depende del tamaño de los datos de entrada que vendrá dado por algún parámetro que define dichos datos (número de componentes, filas y columnas, etc.)
- La eficiencia además de depender del tamaño de los ejemplares depende de la naturaleza de los mismos: caso mejor, caso peor y caso medio.
- Para calcular la eficiencia es posible considerar sólo el número de veces que se debe ejecutar una operación elemental pues éstas tendrán un coste unitario.
- **¿En qué "unidad" mediremos esta eficiencia?**
- Podríamos pensar en utilizar segundos, sin embargo, el tiempo de ejecución depende del ordenador utilizado...

7

## Eficiencia y notación asintótica. Midiendo la eficiencia (ii)

- La solución está en el **principio de invarianza**, dicho principio afirma que **dos implementaciones distintas del mismo algoritmo no diferirán en su eficiencia más que en una constante multiplicativa**.
- Por ejemplo, si un algoritmo escrito en FORTRAN se compila en un Pentium III 1MHz y en un 486 tendríamos dos implementaciones distintas del mismo algoritmo siendo más eficiente la primera. Si al ejecutar este algoritmo en el Pentium III para un problema dado tarda 3 segundos y en el 486 para el mismo problema tarda 30 segundos, el principio de invarianza nos dice que un problema que en el Pentium III requiera 1 minuto necesitaría en el 486 aproximadamente 10.
- De esta forma, **no habrá unidad para medir la eficiencia, nos limitaremos a decir que el tiempo de ejecución de un algoritmo será  $t(n)$** , es decir, una función del tamaño de los ejemplares; en unas máquinas el tiempo de ejecución será  $a \cdot t(n)$  y en otras será  $b \cdot t(n)$ .

8

## Eficiencia y notación asintótica. Notación $O(n)$

– O grande de  $n$  – (i)

- Sabemos que podemos definir la eficiencia de un algoritmo como una función  **$t(n)$** .
- A la hora de **analizar un algoritmo** nos interesa, principalmente, la **forma en que se comporta el algoritmo al aumentar el tamaño de los datos**; es decir, cómo aumenta su tiempo de ejecución.
- Esto se conoce como **eficiencia asintótica de un algoritmo** y nos permitirá comparar distintos algoritmos puesto que deberíamos elegir aquellos que se comportarán mejor al crecer los datos.
- La notación asintótica se describe por medio de una función cuyo dominio es el conjunto de números naturales,  **$\mathbf{N}$** .
- Se describe la **notación O** o límite asintótico superior como:

$$O(g(n)) = \{f(n) / \exists c > 0, n_0 > 0, 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$$

$$f(n): \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

$$g(n): \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

9

## Eficiencia y notación asintótica. Notación $O(n)$

– O grande de  $n$  – (ii)

$$O(g(n)) = \{f(n) / \exists c > 0, n_0 > 0, 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$$

$$f(n): \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

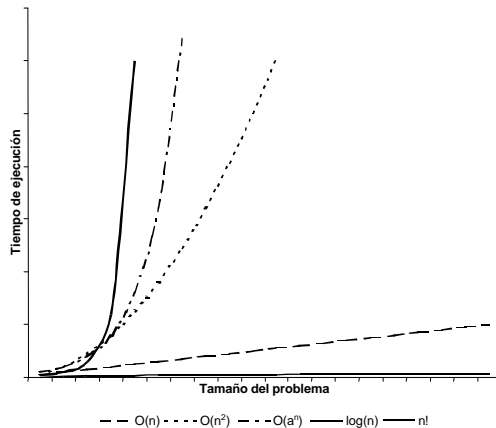
$$g(n): \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

- La notación anterior básicamente nos dice que si tenemos un algoritmo cuyo tiempo de ejecución es  **$f(n)$**  podemos encontrar otra función de  $n$ ,  **$g(n)$** , y un tamaño de problema  **$n_0$**  de tal forma que  $g(n)$  acota superiormente al tiempo de ejecución para todos los problemas de tamaño superior a  $n_0$ .
- **Esta notación**, como veremos a continuación, **facilitará en las comparaciones de eficiencia entre algoritmos diferentes**.

10

## Eficiencia y notación asintótica. Notación $O(n)$ – $O$ grande de $n$ – (iii)

Comparación de eficiencias asintóticas



- En la gráfica se puede ver distintos órdenes de complejidad "típicos".
- $O(\log(n))$  crece de manera imperceptible mientras que  $O(n!)$ ,  $O(a^n)$  y  $O(n^a)$  crecen muy rápidamente.
- La mayor parte de algoritmos son  $O(n^a)$ , los algoritmos más eficientes son  $O(\log(n))$ ; es muy difícil lograr algoritmos  $O(1)$  –tiempo constante–.

11

## Eficiencia y notación asintótica. Notación $O(n)$ – $O$ grande de $n$ – (iv)

- El orden de complejidad se define como una función que domina la ecuación que expresa en forma exacta el tiempo de ejecución del algoritmo.
- **$g(x)$  domina a  $f(x)$** , si dada una constante  $C$  cualquiera,  $C \cdot g(x) \geq f(x) \forall x$
- $g(x)$  domina asintóticamente a  $f(x)$ , si  $g(x)$  domina a  $f(x)$  para valores muy grandes de  $x$ .
- Por ejemplo, si  $f(n) = n^2 + 5n + 100$  y  $g(n) = n^2$  entonces  $g(n)$  domina a  $f(n)$ .

12

## Eficiencia y notación asintótica. Notación $O(n)$ – $O$ grande de $n$ – (v)

- En próximas lecciones se verá la forma de calcular complejidades de algoritmos. Sin embargo, podemos adelantar algunas reglas:
  - $O(C \cdot g) = O(g)$ ,  $C$  es una constante.
  - $O(f \cdot g) = O(f) \cdot O(g)$ , y viceversa.
  - $O(f / g) = O(f) / O(g)$ , y viceversa.
  - $O(f+g)$  = función dominante entre  $O(f)$  y  $O(g)$
  - $n \cdot \log_2 n$  domina a  $\log_2 n$
  - $b^n$  domina a  $c^n$  si  $b > c$
  - $n^k$  domina a  $n^m$  si  $k > m$
  - $\log_a n$  domina a  $\log_b n$  si  $b > a > 1$
  - $n!$  domina a  $b^n$
  - $b^n$  domina a  $n^a$  si  $a > 0$
  - $n$  domina a  $\log_a n$  si  $a > 1$
  - $\log_a n$  domina a 1 si  $a > 1$

13

## Eficiencia y notación asintótica. Notación $O(n)$ – $O$ grande de $n$ – (vi)

- Por ejemplo, el orden de complejidad del algoritmo siguiente...

```

para i de 1 a n
  para j de 1 a n
    escribir i+j
  fpara
fpara

```

- Se calcularía como se muestra a la derecha llegándose a la conclusión de que su complejidad es  $O(n^2)$ . Esto quiere decir, por ejemplo, que si el **tamaño de los datos aumenta en un orden de magnitud** (se multiplica por 10), **el tiempo de ejecución aumentaría** (en realidad estaría acotado superiormente) **en dos órdenes de magnitud** (se multiplicaría por 100).

### Paso 1

```

para i de 1 a n
  para j de 1 a n
    escribir i+j
  fpara
fpara

```

La complejidad de esta sentencia es constante

### Paso 2

```

para i de 1 a n
  para j de 1 a n
    escribir i+j
  fpara
fpara

```

La complejidad del bucle es el producto del número de ejecuciones por la complejidad de la/s sentencia/s ejecutada/s.

### Paso 3


```

para i de 1 a n
  para j de 1 a n
    escribir i+j
  fpara
fpara

```

La complejidad del bucle es el producto del número de ejecuciones por la complejidad de la/s sentencia/s ejecutada/s.


14



## Eficiencia y notación asintótica. Resumen (i)

- Para resolver un **problema** pueden existir **varios algoritmos**. Por tanto, es lógico elegir aquel que use menos recursos (tiempo o espacio).
- Al hablar de la **eficiencia** en tiempo de ejecución de un algoritmo nos referiremos a lo **"rápido"** que se ejecuta.
- La **eficiencia de un algoritmo dependerá, en general, del "tamaño de los datos de entrada"**.
- El **tamaño de un ejemplar sería el número de bits necesarios para representarlo en el ordenador**. Sin embargo, basta con **determinar algún parámetro que nos permita dirimir entre dos ejemplares cuál es el más eficiente** (número de componentes, filas y columnas, etc.).
- Ejemplos de **problemas** serían: **"multiplicar dos números enteros"** o **"resolver una ecuación de segundo grado"**.
- **Ejemplares** de dichos problemas serían:  $(425, 7)$  o  $(1, 1, 2)$ . Es decir:  $425 \cdot 7$  y  $x^2 + x + 2$ .
- Un **problema** dado puede tener un número **infinito de ejemplares** (multiplicar dos números enteros) o **finito** (jugar una partida de ajedrez).
- Un **algoritmo que afirma resolver un problema debe resolver todos los ejemplares del mismo**. Si hay un solo ejemplar que no sea resuelto por el algoritmo el algoritmo no es correcto.

15



## Eficiencia y notación asintótica. Resumen (ii)

- **No todos los ejemplares de un problema son iguales**. Por ello, la **eficiencia del algoritmo puede variar en función del ejemplar o del grupo de ejemplares**.
- A la hora de analizar un algoritmo es necesario saber que pueden darse tres tipos de ejemplares o casos: **caso mejor, caso peor y caso medio**.
- Una **operación elemental es aquella cuyo tiempo de ejecución tiene una cota superior constante que sólo depende de su implementación**. Se considera que son de **coste unitario**.
- La **eficiencia no tiene unidades de medida, se usa la notación  $O(n)$** .
- La notación anterior básicamente nos dice que si tenemos un algoritmo cuyo tiempo de ejecución es  $f(n)$  podemos encontrar otra función de  $n$ ,  $g(n)$ , y un tamaño de problema  $n_0$  de tal forma que  $g(n)$  acota superiormente al tiempo de ejecución para todos los problemas de tamaño superior a  $n_0$ .
- Esta notación, como veremos a continuación, **facilitará en las comparaciones de eficiencia entre algoritmos diferentes**.
- Existen reglas y métodos para calcular el orden de complejidad de nuestros algoritmos para poder compararlos.

16