







## Tema 23. Diseño de algoritmos. Técnicas descriptivas

<b>Introducción: algoritmos</b>	<b>2</b>
<b>Diseño de algoritmos</b>	<b>2</b>
Características	2
Complejidad computacional: eficiencia y costes	2
<b>Técnicas descriptivas</b>	<b>3</b>
	3
	4
	5
<b>Técnicas de diseño de algoritmos</b>	<b>6</b>
	6
	6
	7
	8
<b>Conclusión</b>	<b>9</b>
<b>Bibliografía y webgrafía</b>	<b>10</b>

Páginas	10
Palabras	3000
Esquemas/dibujos	4
Tablas	2
<b>Nota:</b> máx. 2700 palabras. Cada tabla/esquema unas 100-150 palabras	

- **Complejidad espacial.** Son los recursos de almacenamiento que un algoritmo necesita para su ejecución (la memoria).

En realidad, la **eficiencia** de un algoritmo **depende de múltiples factores**: los externos (máquina en la que se ejecuta, los datos de entrada, el compilador, el lenguaje en el que se codifica, etc.) y los internos (estos son el nº de instrucciones asociadas al algoritmo y la forma en que tienen estas de ejecutarse).

Generalmente, para realizar un adecuado cálculo de la complejidad del algoritmo **debemos tener en cuenta únicamente los factores internos**, que son los que determinarán la eficiencia del algoritmo ante cualquier situación. Esto se realiza a través de cálculos matemáticos que permiten determinar la eficiencia sin necesidad de ejecutar el algoritmo como tal. Para ello, se utiliza generalmente la **notación asintótica Big-Omicron** (O), utilizada para representar normalmente la cota superior; es decir, el caso más desfavorable de ejecución del algoritmo (por ejemplo, a la hora de ordenar una lista de forma ascendente, el caso más desfavorable sería que la lista estuviera ordenada de forma descendente). Sin embargo, **en muchas ocasiones se describen los algoritmos en función de su caso promedio**, ya que es el más probable que pudiera ocurrir en una ejecución arbitraria del mismo (ej. el algoritmo de ordenación **Quick Sort** tiene como caso más desfavorable  $O(N^2)$ , pero como caso promedio  $O(N \cdot \log N)$ ).

**Utilizando la notación O se puede establecer el orden o magnitud del algoritmo en función de N**, siendo N el nº de datos que debe tratar el problema (el cual es, generalmente, un valor de entrada del mismo). De esta forma,  $O(N)$  indicará que el algoritmo debe tratar todos los datos del problema;  $O(N^2)$  que, por cada dato, deberá tratar a su misma vez, de nuevo, todos los datos (imaginemos comparar cada dato con todo el resto de datos). De esta forma podemos establecer la complejidad del algoritmo como O en función de N, quedando las distintas magnitudes ordenadas de menor a mayor coste de la siguiente forma:

$$O(1) < O(\log N) < O(N) < O(N \cdot \log N) < O(N^2) < O(N^3) < \dots < O(a^N) < O(N!) < O(N^N)$$

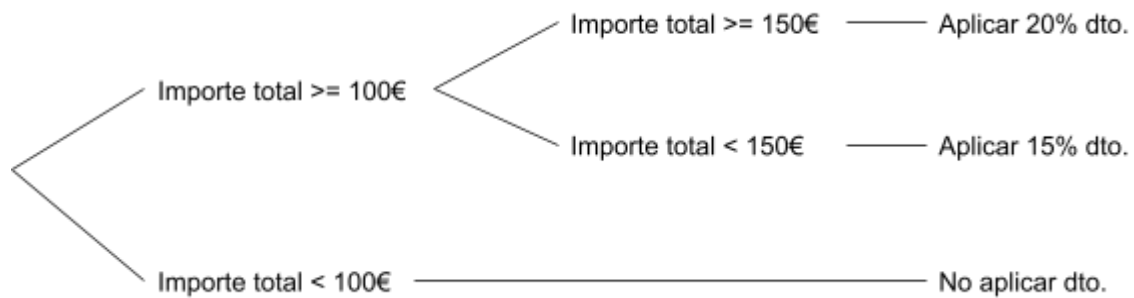
Como ejemplo, podemos pensar en el algoritmo de ordenación **Merge Sort**, con coste  $O(N \cdot \log N)$ , más eficiente que el **Bubble Sort**, con coste  $O(N^2)$ .

### 3. Técnicas descriptivas

Puesto que **un algoritmo debe ser independiente del lenguaje en el que se implemente**, existen diversas técnicas descriptivas que permiten especificar el funcionamiento del mismo de tal forma que tanto analistas como programadores lo puedan entender:

#### 3.1. Lenguaje natural y pseudocódigo

- **Lenguaje natural.** Consiste en describir el algoritmo o transformación empleando un lenguaje natural. Tiene la ventaja de que se entiende muy fácilmente, pero es una técnica imprecisa y redundante.

Ejemplo árbol de decisión:

#### 4. Técnicas de diseño de algoritmos

Algunas de las técnicas para el diseño de algoritmos más conocidas, generalizadas y utilizadas son las siguientes:

##### 4.1. Algoritmos voraces

Es la técnica más simple. Este tipo de algoritmos **se utilizan generalmente para tratar de resolver problemas matemáticos de optimización**. Estos algoritmos funcionan en fases, y en cada fase se toma la decisión más óptima local (es decir, sin tener en cuenta posibles consecuencias futuras ni si esa decisión permitirá obtener la solución más óptima global).

La elección de la solución más óptima local se realiza a través de una función llamada “de selección”. Después de cada selección, el algoritmo comprueba si todas las decisiones tomadas hasta el momento siguen admitiendo una solución factible del problema (es decir, no se incumplen restricciones que hacen que las decisiones actuales hayan conducido a una no solución) y, también, si ya se ha encontrado una solución final del problema (en este caso, finaliza el algoritmo). De lo contrario, en el siguiente paso del algoritmo se vuelven a realizar los mismos pasos, pero con una versión más reducida del problema, ya que ya hay un conjunto de decisiones que se han tomado y que no se pueden revertir.

Un **ejemplo** es el problema del cambio de monedas (tenemos que devolver un importe  $M$  utilizando un conjunto de  $N$  tipos de monedas, con infinitas monedas de cada tipo):

- La **función de selección** en cada paso sería: el tipo de moneda más grande de  $N$  que no supere el importe pendiente a devolver.
- Comprobar si la **solución actual es factible** o no: que la suma del total de monedas escogidas sea  $\leq M$ .
- Una **solución al problema** es: que la suma total de monedas escogidas =  $M$ .

Otros ejemplos de algoritmos voraces son los del **cálculo del camino más corto en un grafo**, como el algoritmo de Dijkstra (muy utilizado por ejemplo en redes, en algoritmos de enrutamiento en los routers), o **hallar el árbol de expansión mínima en un grafo**, a través del algoritmo de Kruskal.

##### 4.2. Divide y vencerás

La técnica de divide y vencerás consiste en **dividir un problema en varios subproblemas más pequeños**. Para ello, se solucionan los subproblemas y se combinan las soluciones de

- **Orden 36/2012**, para el currículo de **ASIR** (Administración de Sistemas Informáticos en Red).
- **Orden 58/2012**, para el currículo de **DAM** (Desarrollo de Aplicaciones Multiplataforma).
- **Orden 60/2012**, para el currículo de **DAW** (Desarrollo de Aplicaciones Web).
- **Decreto 87/2015**, para el currículo de la ESO y el Bachillerato.

Dónde se imparte	Módulo profesional / Asignatura	Curso
[REDACTED]	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]
[REDACTED]	[REDACTED]	[REDACTED]

## 6. Bibliografía y webgrafía

- Alcalde, E. y García, M. (1995). *Metodología de la programación*. McGraw-Hill.
- [REDACTED]
- [REDACTED]
- [REDACTED]
- Wirth, N. (1999). *Algoritmos + estructuras de datos = programas*. Madrid: Ediciones del Castillo.