Sesión 4

Como continuación de la sesión 3, completar la hoja de estilos CSS que se proporciona, estilo.css, de forma que la presentación visual del documento sea similar a la que aquí se muestra.



## Orden de aplicación de estilos

1. Cabecera del cuerpo del documento (elemento header) y descendientes
2. Menú principal de navegación (elemento nav)
3. Algoritmos, completar para que se muestran algo a la derecha (igual que en el documento)
4. Centrar las tablas y su pie
5. Centrar ecuaciones como aparecen en el documento
6. Centrar la Figura 3
7. Presentar las Figuras 4 y 5 igual que en el documento
8. Utilizar el modelo de caja flexible para presentar los sitios de interés como se muestra en la imagen previa
9. Para anchos del *viewport* mayor o igual a 900px ubicar los sitios de interés como una columna a la derecha de la sección principal del documento (<section id=”main”>). Además, el ancho máximo del bloque de contención inicial ha de ser de 70em y mostrase centrado en el *viewport* (ver la imagen de la siguiente página)
10. Numerar los capítulos de los documentos HTML igual que en el documento Word



Contenido

[1 Análisis de algoritmos 3](#_Toc399142322)

[1.1 Complejidad temporal 3](#_Toc399142323)

[1.1.1 Mejor caso, peor caso y caso promedio 4](#_Toc399142324)

[1.2 Notación asintótica 5](#_Toc399142325)

[1.2.1 Orden y omega 5](#_Toc399142326)

[1.2.2 Orden exacto 6](#_Toc399142327)

[1.2.3 Órdenes de complejidad 6](#_Toc399142328)

[1.3 Complejidad espacial 7](#_Toc399142329)

# Análisis de algoritmos

Puede ocurrir que la solución a un problema se pueda llevar a cabo con algoritmos significativamente diferentes. Por otra parte, ciertas decisiones sobre la representación de una abstracción de datos pueden afectar de forma distinta a las operaciones del tipo de dato. Y, en estos casos, cabe preguntarse, cómo se pueden comparar distintos algoritmos que resuelven el mismo problema.

Una solución factible para comparar la rapidez de los algoritmos sería mediante un procedimiento empírico, programando los algoritmos (en algún lenguaje de programación) y ejecutándolos para medir el tiempo que tarda en ejecutarse cada algoritmo para diferentes entradas (tamaño y casos distintos). Sin embargo, esta aproximación empírica tiene ciertos inconvenientes, no sólo depende del tamaño del problema (tamaño o talla de los datos de entrada), también depende de cómo se elijan los datos de entrada (casos seleccionados para realizar el experimento) y de factores externos que no aportan información sobre los algoritmos como, por ejemplo, el lenguaje de programación elegido y la máquina en la que se realice la ejecución.

En este apartado se presentarán criterios de comparación de algoritmos basados en la eficiencia, es decir, en el mejor aprovechamiento de los recursos computacionales. Se estudiaran dos factores:

*Coste* o *complejidad espacial*: la cantidad de memoria que consume la estructura de datos.

*Coste* o *complejidad temporal*: el tiempo que necesita el algoritmo para resolver un problema.

Ambos determinan el **coste** o **complejidad computacional**. No siempre coincidirán consumo espacial óptimo con mínimo coste temporal; es más, por lo general ambos factores entrarán en competencia, siendo necesario adoptar un compromiso razonable entre ambos costes.

## Complejidad temporal

Una alternativa para calcular el tiempo de ejecución de un algoritmo en función del tamaño n del problema, es contar el número de instrucciones a ejecutar y multiplicarlo por el tiempo requerido para cada instrucción. Considérese, por ejemplo, el siguiente algoritmo:

i ← 0;

k ← 1;

*mientras* i < n *hacer*

i ← i + 1;

k ← k \* i;

*fin mientras*

Figura . Algoritmo de cálculo de n!

Los valores que tomaría la variable i durante la ejecución serían: 0, 1, 2, …, n. Las sentencias del bucle se ejecutarían n veces y, así, tendríamos la Tabla 1 que indica el número de veces que se ejecuta cada instrucción.

|  |  |  |  |
| --- | --- | --- | --- |
| Asignaciones | Sumas | Productos | Comparaciones |
|  |  |  |  |

Tabla . Conteo del número de instrucciones en la ejecución del algoritmo n!

Si, además, queremos independizar el cálculo del tiempo de ejecución de un algoritmo de factores externos (lenguaje de programación, máquina de ejecución, etc.), no debe importarnos el coste concreto de cada operación elemental. Bastará con saber cuántas operaciones elementales ejecuta un programa y cómo depende ese número de la talla del problema a resolver. Siendo una operación elemental (o paso) un segmento de código cuyo tiempo de proceso no depende de la talla del problema considerado y está acotado por alguna constante.

Entre las operaciones que se consideran **pasos** están las siguientes: *operaciones aritméticas*, *operaciones lógicas*, *asignaciones*, *acceso o asignación a elementos de un array* y *operaciones de entrada/salida de valores de tipos de datos simples* (entero, real, carácter o lógico).

El coste de las operaciones elementales que no son pasos se expresan en función del número de pasos con el que podrían efectuarse y el **coste computacional temporal** de un algoritmo se define como el número de pasos expresado en función de la talla del problema. Por ejemplo, el coste computacional temporal del algoritmo de la Figura 1 es (que es la suma de los conteos correspondientes de la Tabla 1). No obstante, los factores de cada término en la expresión previa no son relevantes, sustituyéndose por constantes arbitrarias .

*Cualquier secuencia de pasos cuya longitud no depende de la talla del problema cuenta como una cantidad constante de pasos*.

### Mejor caso, peor caso y caso promedio

En la práctica, la mayor parte de los algoritmos incluyen alguna sentencia condicional y, en consecuencia, el coste computacional temporal además de depender de la talla del problema también va a depender de los datos concretos que se le presenten (casos). Esto hace que más que calcular un valor del coste computacional temporal haya que calcular un rango de valores para el mismo

los valores extremos de este rango se conocen habitualmente como **mejor caso** y **peor caso**. Y, entre ambos, se hallará algún **caso promedio** o más frecuente.

Considérese, por ejemplo, el cuerpo de la función de la Figura 2 que busca un valor x en un *array* a[0..n‑1] de n elementos.

i ← 0;

*mientras* i < n *hacer*

*si* a[i] = x

*entonces* *retorna* cierto;

i ← i + 1;

*fin mientras*;

*retorna* falso;

Figura . Búsqueda del valor x en el array a

Dando por hecho que la construcción *retorna* fuerza la terminación de la función y produce un resultado (en nuestro caso, cierto o falso), para este algoritmo no resulta tan obvio determinar los valores que va a tomar la variable i durante la ejecución y el número de veces que se ejecuta el bucle mientras. La razón es que éstos van a depender de los datos de entrada (casos), concretamente de si el *array* contiene o no el valor x buscado y, en el caso de que así se sea, la posición del *array* en que este valor se encuentre.

Cuando se habla del mejor de los casos nos referimos a los datos de entrada que, para cada valor particular de la talla n del problema, se resuelven más rápidamente con el algoritmo. Para el ejemplo de la Figura 2, cuando el elemento x buscado se encuentra en la primera posición del *array*. En este caso, el bucle se ejecuta una única vez (y no completamente) y el algoritmo termina con una secuencia constante de pasos que no depende de la talla del problema; es decir

El peor de los casos lo determinan los datos de entrada que, para cada valor particular de n, hacen que el algoritmo se ejecute con el mayor número posible de pasos. Para el ejemplo, el peor caso es cuando el algoritmo termina retornando falso; es decir, cuando el valor x buscado no se encuentra en el *array*. Siendo este el caso, el bucle se ejecuta n veces una cantidad constante de pasos; es decir

El cálculo analítico del coste computacional temporal para el caso promedio (o más frecuente) es algo más complejo y sobrepasa los objetivos que se pretenden alcanzar aquí y, además, para comparar el coste temporal de los algoritmos, por lo general, será suficiente con comparar el peor de los casos.

## Notación asintótica

En general, el análisis de coste de los algoritmos es especialmente relevante cuando estos se aplican a problemas de gran tamaño (o talla). Casi siempre los problemas pequeños se pueden resolver de cualquier forma y las limitaciones aparecen al tratar problemas grandes. En todo caso, debe tenerse en cuenta que cualquier técnica de ingeniería, si funciona, acaba aplicándose al problema más grande que sea posible: las tecnologías de éxito, antes o después, acaban llevándose al límite de sus posibilidades.

Estas consideraciones llevan a estudiar el comportamiento de un algoritmo cuando se fuerza el tamaño n del problema al que se aplica. Matemáticamente hablando, cuando n tiende a infinito, es decir, su comportamiento asintótico (sin considerar constantes). Se introducirán, por tanto, ciertas herramientas matemáticas fundamentales que simplifican notablemente el análisis de costes y permiten expresar de forma muy concisa los resultados. Aprenderemos a caracterizar el coste mediante funciones simples que acoten superior e inferiormente el coste de todas las posibles entradas para tallas suficientemente grandes. Para ello se necesitan definir familias de cotas.

### Orden y omega

La notación , que ha de leerse **del orden de** , es útil para estimar una cota superior del tiempo de ejecución de un algoritmo para entradas de talla . La Figura 3 muestra qué significa que una función sea .

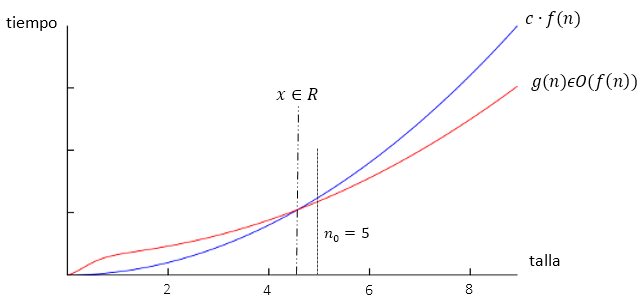


Figura . La función g(n) "es del orden de" f(n)

También es interesante poder estimar también una cota inferior del tiempo de ejecución de un algoritmo para entradas de talla . La notación , que se lee **omega de** , es útil para estimar una cota inferior del tiempo de ejecución de un algoritmo para entradas de talla .

### Orden exacto

El análisis asintótico de un algoritmo resultaría más satisfactorio si pudiéramos acotar, a la vez superior e inferiormente, su tiempo de ejecución por una misma función . Para ello se introduce una última notación

que se lee **del orden exacto de** .

### Órdenes de complejidad

Se dice que define un orden de complejidad. Las funciones que pertenecen a cada orden tienen un adjetivo que las identifican y que se recoge en la siguiente tabla:

|  |  |  |  |
| --- | --- | --- | --- |
| Sublineales |  | Constantes |  |
| Logarítmicas |  |
|  |  |
| Lineales |  |  |  |
| Superlineales |  |  |  |
| Polinómicas | Cuadráticas |  |
| Cúbicas |  |
| Exponencial |  |  |
| Factorial |  |  |

Tabla . Nombres de las funciones pertenecientes a distintos órdenes

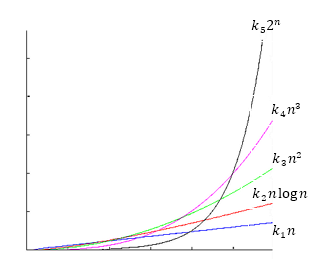
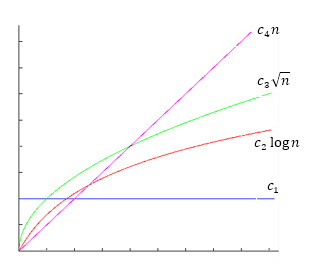
Se dice que el coste temporal de un algoritmo es lineal cuando es de y logarítmico cuando es de ), etc. En la Figura 4 y Figura 5 se han comparado los órdenes sublineales y superlineales con el orden lineal, respectivamente.

Figura 5. Órdenes lineal y superlineales

Figura 4. Órdenes sublineales y lineal

#### Impacto práctico

Para hacernos una idea de la importancia de los órdenes de complejidad en la Tabla 3 se presentan los tiempos utilizados por las funciones de complejidad para resolver un problema de talla .

Se observa que los algoritmos de complejidad y son los que muestran un comportamiento más natural, al doblar el número de datos procesados se duplica el tiempo necesario para procesarlos.

Los algoritmos de complejidad logarítmica crecen muy lentamente conforme crece. Necesitan poco más tiempo para procesar el doble de datos.

Los algoritmos de complejidad polinómica presentan dificultades a medida que crece la talla, la práctica viene a decirnos que son el límite de lo tratable. Los cuadráticos dejan de ser útiles para tallas medias o grandes y los cúbicos sólo son útiles para problemas pequeños, complejidades polinómicas de mayor potencia prácticamente son inaceptables.

Cualquier algoritmo por encima de una complejidad polinómica se dice intratable y sólo será aplicable a problemas muy pequeños.

**Funciones de complejidad**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Talla |  |  |  |  |  |  |  |
| 5 | 3 | 5 | 12 | 25 | 125 | 32 | 120 |
| 10 | 4 | 10 | 33 | 100 | 1.000 | 1.024 | 3,63 106 |
| 100 | 7 | 100 | 664 | 104 | 106 | 1,27 1030 | > 10100 |
| 200 | 8 | 200 | 1.529 | 4 104 | 8 106 | 1,6 1060 | > 10100 |
| 1.000 | 10 | 1.000 | 9.965 | 106 | 109 | > 10100 | > 10100 |
| 2.000 | 11 | 2.000 | 2,2 104 | 4 106 | 8 109 | > 10100 | > 10100 |
| 10.000 | 14 | 104 | 1,33 105 | 108 | 1012 | > 10100 | > 10100 |

Tabla . Tiempo en función de la talla para las funciones de complejidad más comunes

El comportamiento de las funciones de complejidad a medida que crece la talla del problema expuesto en la Tabla 3, explica el porqué de la búsqueda de algoritmos de complejidad lineal incluso, si es factible y con algo de suerte, de complejidad logarítmica. De no encontrarse un algoritmo de complejidad lineal, un algoritmo de complejidad no es mala alternativa. Si se encuentran soluciones polinomiales, se puede tratar con ellas a pesar de las limitaciones que tienen para problemas de talla media y grande; pero ante soluciones de complejidad exponencial más vale seguir buscando.

## Complejidad espacial

La complejidad espacial es el estudio de la eficiencia de los algoritmos en lo que respecta a su consumo de memoria.

Todo lo expuesto para la complejidad temporal es igualmente válido para el estudio de la complejidad espacial, sustituyendo tiempo empleado por espacio de memoria utilizado. Un razonamiento similar al seguido con el coste temporal lleva a considerar únicamente la evolución del consumo de memoria con la talla del problema. En el estudio asintótico no es relevante si un algoritmo consume la mitad o el doble que otro, pero sí que utilice una cantidad de memoria que crece con el cuadrado de la talla cuando otro sólo requiere una cantidad de memoria constante, por ejemplo. El concepto de **paso** también es utilizable aquí, asociado al concepto de celda de memoria, no importa el número de bytes que ocupa una variable, sólo importa si su tamaño es o no función (y de qué orden) de , la talla del problema.