Estrategias de programación y estructuras de datos

Grado en Ingeniería Informática

Grado en Tecnologías de la Información

Departamento de Lenguajes y Sistemas informáticos

3

Análisis de la eficiencia de los algoritmos

Coste asintótico temporal

Javier Vélez Reyes *jvelez@lsi.uned.es*

José Ignacio Mayorga Toledano *nmayorga@lsi.uned.es*

Departamento de Lenguajes Y Sistemas Informáticos

UNED

UNED

ETS de Ingeniería Informática

Índice

Análisis de la eficiencia de los algoritmos

Objetivos generales

Objetivos

- Aprender qué es la eficiencia de los algoritmos
- Aprender los criterios en torno a los cuales ésta puede clasificar
- Conocer las principales métricas de medida de la eficiencia
- Aprender cómo se puede calcular el coste de un algoritmo bajo 3 supuestos
  - Medida del tiempo de ejecución
  - Medida asintótica O
  - Hipótesis de caso peor
- Aprender cómo se mide la eficiencia algorítmica de
  - Algoritmos recursivos
  - Algoritmos iterativos
- Adquirir una visión crítica sobre la complejidad algorítmica
- Adquirir herramientas para seleccionar el mejor algoritmo para un problema
- Adquirir herramientas para seleccionar la estructura más eficiente para un problema

Algoritmo

- **Conjunto finito de pasos especificados sin ambigüedades que deben seguirse para resolver un problema**
- Los programas de ordenador son implementaciones de los pasos de un algoritmo utilizando algún lenguaje de programación

¿Qué es la eficiencia de los algoritmos?

El análisis de la eficiencia de los algoritmos permite establecer criterios comparativos y clasificatorios entre algoritmos que ayudan a entender cómo se comportan y escalan éstos en cuanto a la consumición de recursos – tiempo y memoria fundamentalmente – con respecto a la magnitud de sus parámetros de entrada

Definición

El análisis de la eficiencia de los algoritmos es un estudio teórico, formal, comparativo e independiente de la implementación que tiene por objeto clasificar a los algoritmos en familias de complejidad de acuerdo a cómo se comportan según crece la magnitud de sus parámetros de entrada

Estudio teórico

- Estudio formal
- Teórico
- Comparativo
- Clasificatorio

Independencia

- De recursos de máquina
- Del lenguaje de programación
- De aspectos constantes
- De aspectos de escala

Análisis de Algoritmos

- Estimación de la cantidad de recursos (normalmente tiempo y memoria) que un algoritmo necesita para resolver un problema
- Es útil para
  - Determinar "cuan bueno" es un algoritmo para resolver un problema
  - Comparar "la bondad" de distintos algoritmos para resolver un mismo problema
  - Predecir el comportamiento de un algoritmo

¿Qué es la eficiencia de los algoritmos?

El análisis de la eficiencia de los algoritmos permite establecer criterios comparativos y clasificatorios entre algoritmos que ayudan a entender cómo se comportan y escalan éstos en cuanto a la consumición de recursos – tiempo y memoria fundamentalmente – con respecto a la magnitud de sus parámetros de entrada

Criterios de medida de la eficiencia

Criterios de medida

I. Según el tipo de recurso

La eficiencia de los algoritmos puede medirse de acuerdo a la consumición de diferentes recursos. La medida de la eficiencia de un algoritmo puede ser distinta según el recurso

- **Espacial**: Se analiza la relación funcional existente entre el tamaño del problema expresado en términos de sus parámetros de entrada y la consumición del recurso memoria
- **Tiempo**: Se analiza la relación funcional existente entre el tamaño del problema expresado en términos de sus parámetros de entrada y la consumición del recurso tiempo de cómputo ◀ **Foco de atención**
- **Otros**: Se analiza la relación funcional existente entre el tamaño del problema expresado en términos de sus parámetros de entrada y la consumición de otros recursos como periféricos empleados, ancho de banda, consumo eléctrico, etc.

II. Según el tipo de comportamiento

Fijado un tipo de recurso, la eficiencia de los algoritmos puede medirse bajo distintos supuestos en los que se supone se encuentra el problema medio a analizar

II. Según la métrica utilizada

Fijado un tipo de recurso y un determinado supuesto, debe escogerse una métrica comparativa que permita clasificar a los algoritmos en familias con un comportamiento interno similar

Análisis del tiempo de ejecución

- Se representa mediante una función de coste que depende del tamaño de datos de entrada al algoritmo
- La función se obtiene analizando el número de veces que se ejecuta la **operación básica** del algoritmo

$$T(N) = c_{op} * C(N)$$

N: tamaño de datos de entrada

T(N): tiempo de ejecución

C_{op}: coste de la operación básica

C(N): número de repeticiones de la operación básica

Análisis del tiempo de ejecución

Problema	Tamaño de la entrada	Operación básica
Buscar un valor dado en una lista de N elementos	Número de elementos de la lista	Comparación de valores
Multiplicación de dos matrices	Dimensión de las matrices o número total de elementos	Multiplicación de dos valores
Comprobación de si un entero N es primo	Tamaño de N (número de bit en la representación binaria)	División

¿Qué es la eficiencia de los algoritmos?

El análisis de la eficiencia de los algoritmos permite establecer criterios comparativos y clasificatorios entre algoritmos que ayudan a entender cómo se comportan y escalan éstos en cuanto a la consumición de recursos – tiempo y memoria fundamentalmente – con respecto a la magnitud de sus parámetros de entrada

Criterios de medida de la eficiencia

Criterios de medida

I. Según el tipo de recurso

La eficiencia de los algoritmos puede medirse de acuerdo a la consumición de diferentes recursos. El tipo de recurso establece una partición del espacio de algoritmos distinta

II. Según el tipo de comportamiento

Fijado un tipo de recurso, la eficiencia de los algoritmos puede medirse bajo distintos supuestos en los que se supone se encuentra el problema medio a analizar

- **Peor caso**: Se estima la máxima cantidad de recursos que un algoritmo puede necesitar consumir para su ejecución en función de su entrada. Supone una cota superior de estos recursos
- **Caso medio**: Se estima el comportamiento a partir de una hipótesis sobre la distribución estadística de los datos de entrada que se supone más probable. También llamado análisis probabilístico
- **Mejor caso**: Se estiman los recursos necesarios para su ejecución en condiciones óptimas en función de la entrada. Esta medida afecta al ejemplar más sencillo del problema

◀ **Foco de atención**

II. Según la métrica utilizada

Fijado un tipo de recurso y un determinado supuesto, debe escogerse una métrica comparativa que permita clasificar a los algoritmos en familias con un comportamiento interno similar

¿Qué es la eficiencia de los algoritmos?

El análisis de la eficiencia de los algoritmos permite establecer criterios comparativos y clasificatorios entre algoritmos que ayudan a entender cómo se comportan y escalan éstos en cuanto a la consumición de recursos – tiempo y memoria fundamentalmente – con respecto a la magnitud de sus parámetros de entrada

Criterios de medida de la eficiencia

Criterios de medida

I. Según el tipo de recurso

La eficiencia de los algoritmos puede medirse de acuerdo a la consumición de diferentes recursos. El tipo de recurso establece una partición del espacio de algoritmos distinta

II. Según el tipo de comportamiento

Fijado un tipo de recurso, la eficiencia de los algoritmos puede medirse bajo distintos supuestos en los que se supone se encuentra el problema medio a analizar

II. Según la métrica utilizada

Fijado un tipo de recurso y un determinado supuesto, debe escogerse una métrica comparativa que permita clasificar a los algoritmos en familias con un comportamiento interno similar

Cota superior. O

Se mide el coste asintótico de la ejecución de un algoritmo de acuerdo a una cota superior. Es decir, la ejecución nunca, a partir de cierto valor, sobrepasará la cota de coste establecida

◀ **Foco de atención**

Cota inferior. Ω

Se mide el coste asintótico de la ejecución de un algoritmo de acuerdo a una cota superior. Es decir, la ejecución nunca, a partir de cierto valor, será menor que la cota establecida

Cota exacta. Θ

Se mide el coste asintótico de la ejecución de un algoritmo de acuerdo a una cota resultante de la intersección entre O y Ω. Es decir, la ejecución crece al ritmo de la cota establecida

¿Cómo se mide la eficiencia de los algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Medida asintótica de la eficiencia

Si hacemos medidas experimentales sobre sucesivas ejecuciones de un algoritmo y en cada ejecución duplicamos el tamaño del problema, observamos que el tiempo de ejecución también se duplica. En ese caso se puede decir que el coste asintótico del algoritmo es lineal

Búsqueda de un elemento en un vector

Tamaño	Tiempo medido	Tiempo de ejecución
n	A * n + B	t
2n	A * 2n + B	2t
3n	A * 3n + B	3t
...	...	...
kn	A * kn + B	kt

O (n)

El tamaño del problema es proporcional al número de elementos que contiene el vector. Sin embargo el tiempo de ejecución no se ve afectado por constantes multiplicativas o sumativas ya que el coste temporal se duplica al duplicar el tamaño del problema

Análisis del tiempo de ejecución

- **Función lineal**: p.e. $10N+5$
  - El tiempo es proporcional al tamaño de entrada (N)
  - $T(N)$ para N, $10T(N)$ para $10N$
- **Función cuadrática**: p.e. $3N^2+6N+12$
  - El tiempo es proporcional a N^2
  - $T(N)$ para N, $100T(N)$ para $10N$
- **Función cúbica**: p.e. $10N^3 + N^2 + 40N + 80$
  - El tiempo es proporcional a N^3
  - $T(N)$ para N, $1000T(N)$ para $10N$

Análisis del tiempo de ejecución

- Para caracterizar el algoritmo no nos interesa el tiempo exacto, ya que depende de numerosos factores ajenos al mismo
  - Eficiencia de la combinación lenguaje/compilador
  - Velocidad del computador en el que se ejecute
  - Recursos hardware disponibles (p.e. memoria caché)
- Para caracterizar a un algoritmo se usará la tasa de crecimiento del término dominante de la función que describe su tiempo de ejecución

Análisis del tiempo de ejecución

- ¿Por qué nos interesan los términos dominantes?
  1. Para N suficientemente grande, el valor de la función está determinado por su término dominante

$$10N^3 \approx 10N^3 + N^2 + 40N + 80$$

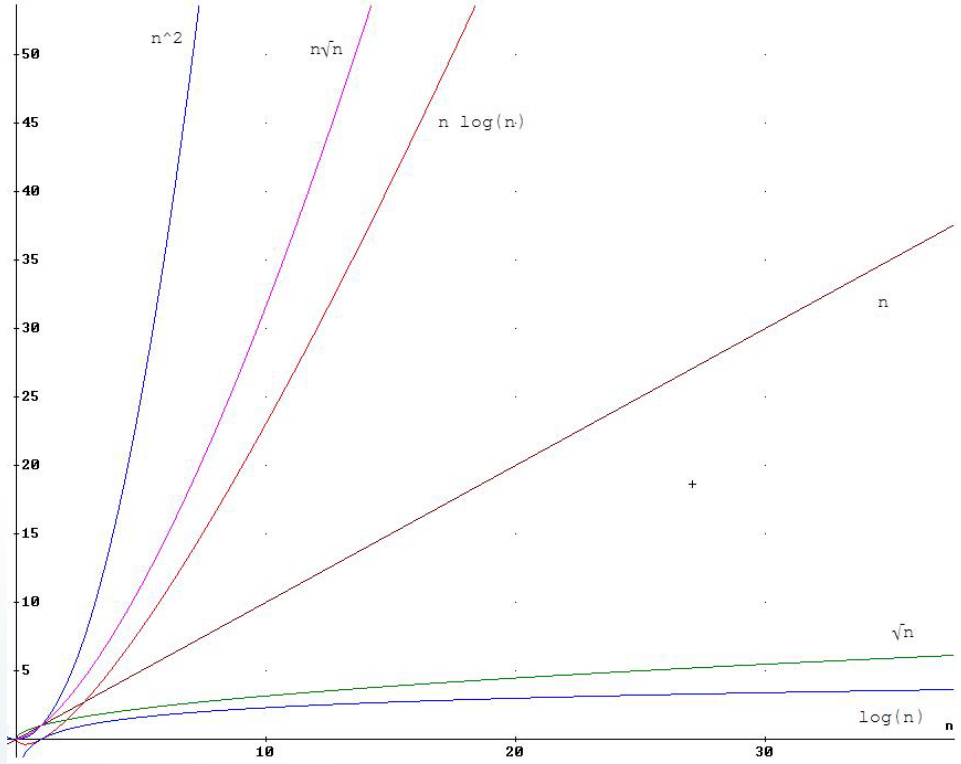
  2. El valor del coeficiente constante del término dominante varía con las condiciones de ejecución
     - Por tanto se ignoran los coeficientes constantes
  3. Las diferencias entre funciones para valores de N pequeños son insignificantes
     - Para N=20 incluso el algoritmo cúbico acaba antes de 5ms

¿Cómo se mide la eficiencia de la algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Ordenes de eficiencia

Cada orden de eficiencia representa un nivel de crecimiento con respecto al tamaño del problema. En la grafica adjunta puede verse la relación comparativa de los niveles de crecimiento de los primeros órdenes de eficiencia anteriores

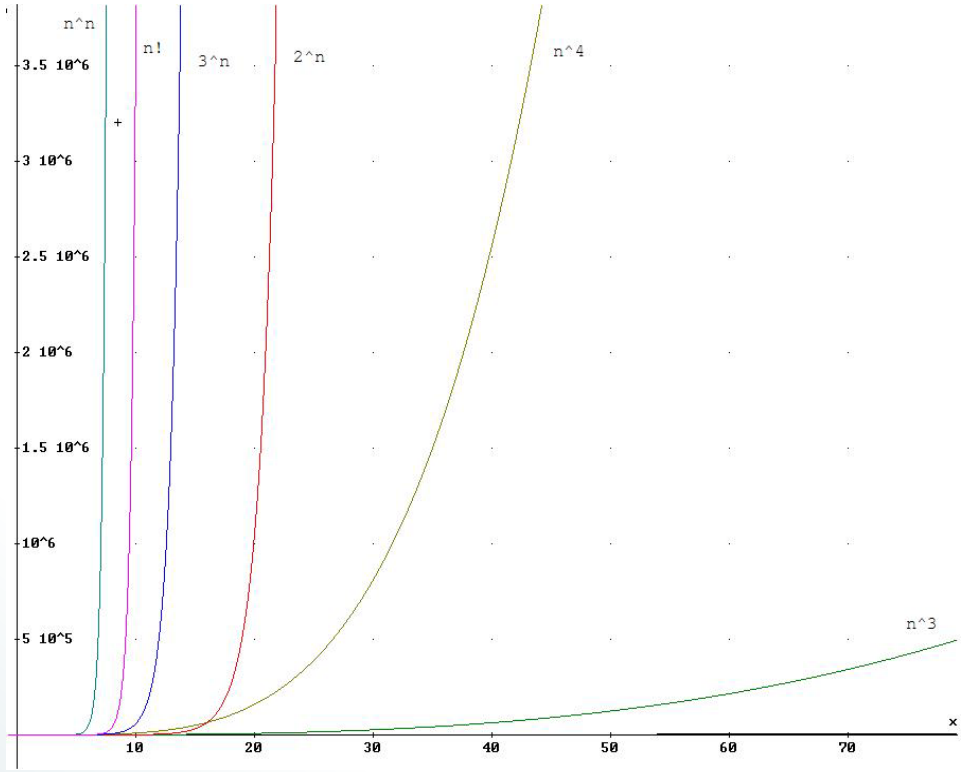


¿Cómo se mide la eficiencia de la algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Ordenes de eficiencia

Cada orden de eficiencia representa un nivel de crecimiento con respecto al tamaño del problema. En la grafica adjunta puede verse la relación comparativa de los niveles de crecimiento de los órdenes de eficiencia siguientes a n^2



La notación O

- Establece un orden relativo entre las tasas de crecimiento de las funciones
  - No tiene en cuenta valores constantes
  - Ignora los tamaños de entrada pequeños

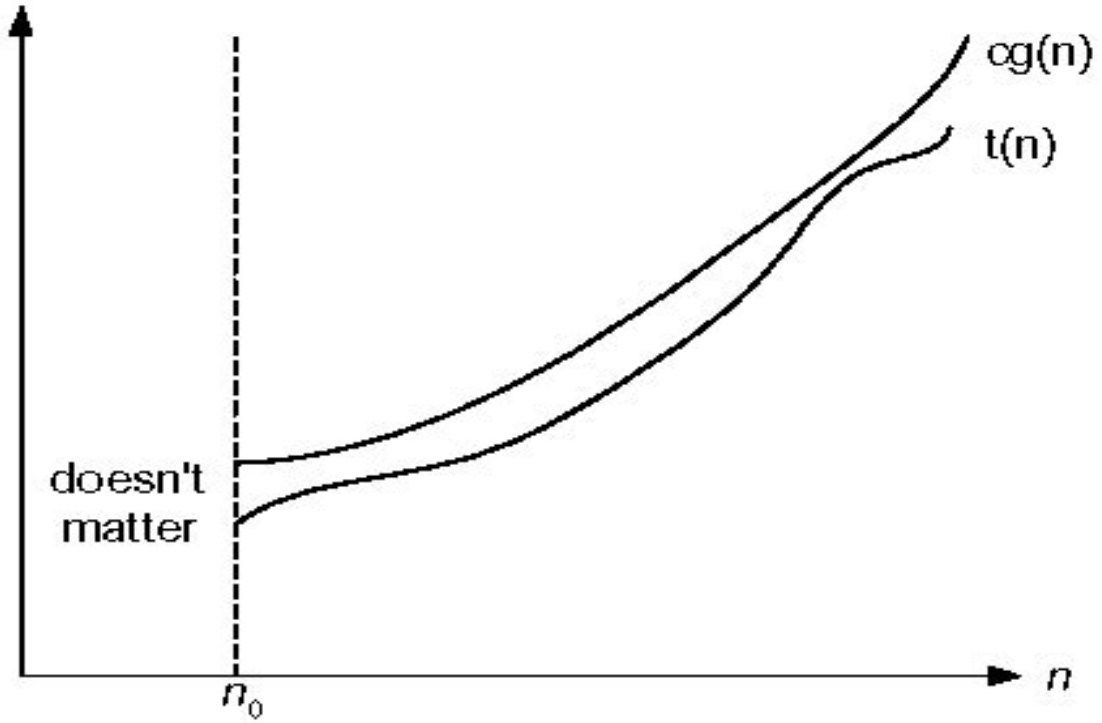
figure 5.9

Meanings of the various growth functions

Mathematical Expression	Relative Rates of Growth
$T(N) = O(F(N))$	Growth of $T(N)$ is $\leq$ growth of $F(N)$.
$T(N) = \Omega(F(N))$	Growth of $T(N)$ is $\geq$ growth of $F(N)$.
$T(N) = \Theta(F(N))$	Growth of $T(N)$ is $=$ growth of $F(N)$.
$T(N) = o(F(N))$	Growth of $T(N)$ is $<$ growth of $F(N)$.

La notación O

- $T(N)$ es $O(g(N))$ si existen costantes positivas c y N_0 tales que para $N \geq N_0$ se cumple que $T(N) \leq cg(N)$



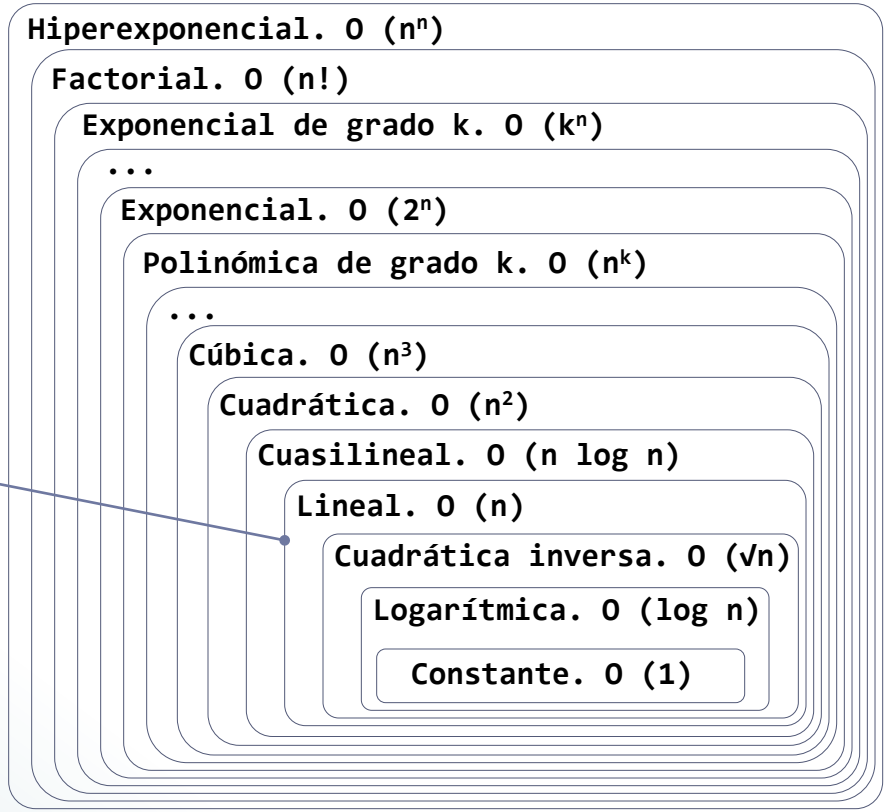
¿Cómo se mide la eficiencia de la algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Ordenes de eficiencia

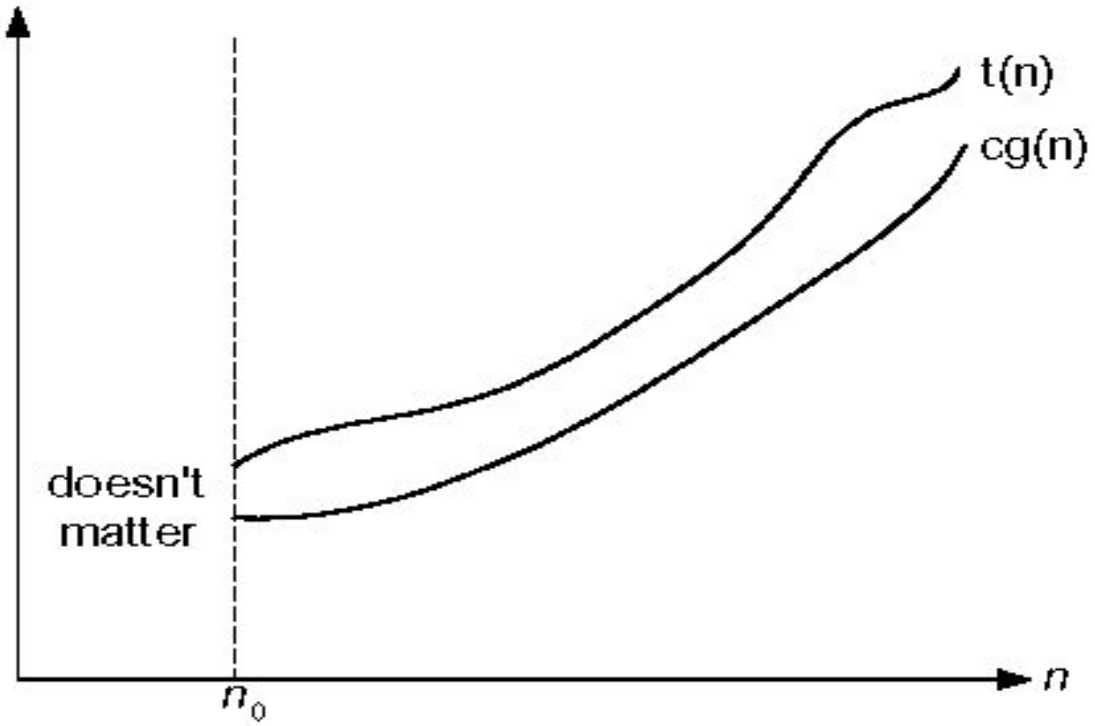
El análisis asintótico nos permite clasificar el espacio de algoritmos en distintas familias que se corresponden con órdenes de complejidad diferentes

```
int factorial (int n) {
  if (n == 0) return 1;
  else return n * factorial (n - 1);
}
```



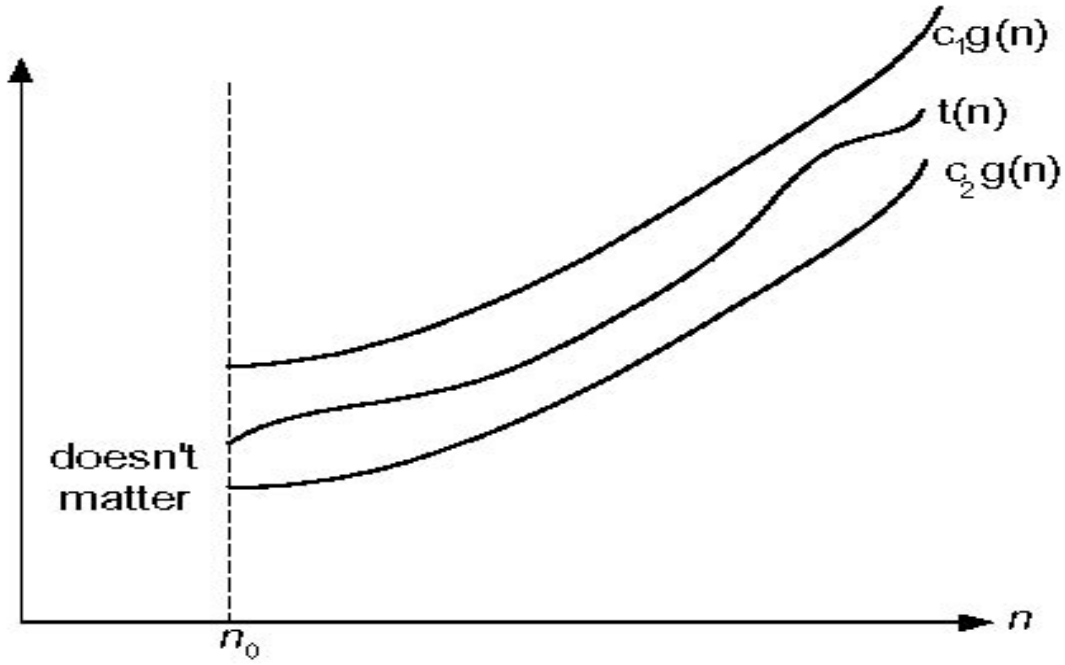
La notación O

- $T(N)$ es $\Omega(g(N))$ si existen costantes positivas c y N_0 tales que para $N \geq N_0$ se cumple que $T(N) \geq cg(N)$



La notación O

- $T(N)$ es $\Theta(g(N))$ si existen costantes positivas c_1, c_2 y N_0 tales que para $N \geq N_0$ se cumple $c_1 g(N) \geq T(N) \geq c_2 g(N)$



La notación O

- $T(N)$ es $o(g(N))$ si existen costantes positivas c y N_0 tales que para $N \geq N_0$ se cumple que $T(N) < cg(N)$

La notación O

- Limitaciones
  - Sólo es apropiado para tamaños de entrada grandes
    - Para tamaños pequeños usar el algoritmo más simple
  - Hay casos en los que las contantes son importantes

    $2N\log(N)$ mejor que $1000N$
    - El análisis es una aproximación
    - No tiene en cuenta aspectos que afectan al rendimiento
      - Accesos a memoria o disco, fallos de página, etc.
  - Al usarse el caso peor, puede sobreestimarse el coste
    - Algoritmos con coste en el caso peor >> caso promedio
      - Si no puede ignorarse, se sobreestima el coste
    - Es más fácil obtener el caso peor que el promedio

¿Cómo se mide la eficiencia de los algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Algoritmos iterativos

El coste temporal de un algoritmo iterativo se calcula acumulativamente a partir del coste de cada una de sus instrucciones constituyentes. Estudiemos su coste

Instrucción	Coste
Operaciones básicas[1]	
- Entrada / salida	$O(1)$
- Asignación	$O(1)$
- Expresiones escalares	$O(1)$
Secuencias sentencias	
- $s_1; s_2; s_3$	$\Sigma\ O(s_i) = \max\{O(s_i)\}$
Sentencias condicionales	
- if (e) $\{b_1\}$ else $\{b_2\}$	$\max\{O(e), O(b_1), O(b_2)\}$
- switch (e) { case c_1 : b_1; case c_2 : b_2; }	$\max\{O(e), O(b_1), O(b_2)\}$

Instrucción	Coste
Sentencias iterativas	
- for (ini;e;inc) {b}	$\max\{O(ini), n\cdot \max\{O(e), O(inc), O(b)\}\}$
- while (e) {b}	$n\cdot \max\{O(e), O(b)\}$
- do {b} while (e)	$n\cdot \max\{O(e), O(b)\}$
Invocación subprogramas	
- f (e_1, e_2)	$\text{Max}\{O(e_1), O(e_2), O(f)\}$
Reglas de producto y suma	
- O (f + g)	$O(f) + O(g) = \max(O(f), O(g))$
- O (f * g)	$O(f) * O(g)$

[1] Las Operaciones básicas no dependen del tamaño del problema

¿Cómo se mide la eficiencia de los algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Algoritmos iterativos

A partir de los análisis anteriores de coste de cada instrucción analicemos la eficiencia de un algoritmo iterativo

```
boolean ordenar (T[] v) {
  int temp;                                          O(1)
  for (int i = 1; i < v.length; i++)                 n * {
    for (int k = v.length - 1; k >= i; k--)            (n-i) * {
      if (v[k] < v[k-1]) {                               max { O(1), max {
        temp = v[k];                                                 O(1),
        v[k] = v[k-1];                                               O(1)
        v[k-1]= temp;                                                O(1) }
      }                                                  }
}                                                    }
```

$= O(1) + n \cdot (n-i)\ O(1)$

$= \max \{ O(1) + n \cdot (n-i)\ O(1) \}$

$= n \cdot (n-i)\ O(1) = O(n^2)$

Algoritmos recursivos

- Los algoritmos recursivos se definen en función de sí mismos
  - Existe una llamada recursiva que resuelve un subproblema (o varios) más pequeño que el problema actual
  - Los resultados devueltos por la recursión son combinados para producir la solución del problema actual
- La función de coste también es recursiva y se denomina **recurrencia**
  - Resolver de forma exacta una recurrencia puede ser muy complejo, así que simplemente calcularemos a qué orden de complejidad pertenece.

¿Cómo se mide la eficiencia de los algoritmos?

La medida de la eficiencia asintótica de algoritmos pretende clasificar cada algoritmo en una familia de complejidad asintótica determinada prescindiendo de consideraciones constantes o de escala. En nuestro estudio utilizaremos medida temporal, caso peor y cota superior

Algoritmos recursivos

El cálculo del coste temporal de un algoritmo recursivo se expresa en términos del coste de los casos base y los casos de recursión del mismo. Para ilustrar cómo se procede consideremos la función del cálculo de un número factorial en su versión recursiva. Entonces, si utilizamos T(n) para representar el coste de computar la ejecución con tamaño n…

```
int factorial (int n) {
  if (n == 0) return 0;
  else return n * factorial (n - 1);
}
```

$$T(n) = \begin{cases} C_1 & \text{si } n == 0 \quad \text{Coste caso base} \\ T(n-1) + C_2 & \text{si } n > 0 \quad \text{Coste caso recurrente} \end{cases}$$

Resolución para factorial (5):

$$T(5) = T(4) + C_2 = T(3) + C_2 + C_2 = T(2) + C_2 + C_2 + C_2 = T(1) + C_2 + C_2 + C_2 + C_2$$
$$= T(0) + C_2 + C_2 + C_2 + C_2 + C_2 = C_1 + C_2 + C_2 + C_2 + C_2 + C_2 = C_1 + 5 * C_2$$
$$= O(n)$$

Dado que el coste temporal resulta linealmente proporcional al tamaño del problema el orden de complejidad es lineal

Algoritmos recursivos

Reducción mediante substracción

$$T(n)=\begin{cases} c_b(n) & si\ 0\le n<b \\ a\cdot T(n-b)+c_{nr}(n) & si\ n\ge b \end{cases}$$

Forma general de la recurrencia

$$T(n)\in\begin{cases} O(n\cdot c_{nr}(n)+c_b(n)) & si\ a=1 \\ O(a^{(n\ \mathbf{div}\ b)}\cdot(c_{nr}(n)+c_b(n))) & si\ a>1 \end{cases}$$

Reglas prácticas para el cálculo del orden

- $c_b(n)$ → coste del caso base
- $c_{nr}(n)$ → coste de las operaciones no recursivas en cada llamada
- a → número de llamadas recursivas
- n-b → tamaño de los subproblemas

Algoritmos recursivos

Reducción mediante substracción

Ejemplo de casos prácticos habituales

- Recursión lineal (a = 1)
  - $c_b(n) \in O(1)$ → $T(n) \in O(n \cdot c_{nr}(n))$
  - $c_{nr}(n) \in O(n^k)$ y $c_b(n) \in O(1)$ → $T(n) \in O(n^{k+1})$
- Recursión múltiple (a > 1)
  - $c_b(n) \in O(1)$ → $T(n) \in O(a^{n \text{ div } b} \cdot c_{nr}(n))$
  - $b = 1$ y $c_b(n), c_{nr}(n) \in O(1)$ → $T(n) \in O(a^n)$

Algoritmos recursivos

Reducción mediante división

$$T(n)=\begin{cases} c_b(n) & si\ 1\le n<b \\ a\cdot T(n\ \mathbf{div}\ b)+c_{nr}(n) & si\ n\ge b \end{cases}$$

Forma general de la recurrencia

$$T(n)\in\begin{cases} O(\log_b(n)\cdot c_{nr}(n)+c_b(n)) & si\ a=1 \\ O(n^{\log_b(a)}\cdot(c_{nr}(n)+c_b(n))) & si\ a>1 \end{cases}$$

Reglas prácticas para el cálculo del orden

- $c_b(n)$ → coste del caso base
- $c_{nr}(n)$ → coste de las operaciones no recursivas en cada llamada
- a → número de llamadas recursivas
- n **div** b → tamaño de los subproblemas

Algoritmos recursivos

Reducción mediante división

Ejemplo de casos prácticos habituales

- Recursión lineal (a = 1)
  - $c_b(n) \in O(1)$ → $T(n) \in O(\log_b(n) \cdot c_{nr}(n))$
  - $c_{nr}(n) \in O(n^k)$ y $c_b(n) \in O(1)$ → $T(n) \in O(n^k \cdot \log_b(n))$
- Recursión múltiple (a > 1)
  - $c_b(n) \in O(1)$ → $T(n) \in O(a^{n \text{ div } b} \cdot c_{nr}(n))$
  - $a = b = 2$ y $c_b(n), c_{nr}(n) \in O(1)$ → $T(n) \in O(n^{\log_2(2)}) = O(n)$

La notación O

- Comprobación empírica
  - Medir los factores de incremento en los tiempos de ejecución al variar el tamaño de la entrada
    - Pasar de N a $10N$ aumenta $T(N)$ en un factor de 10, 100 o 1000 para $O(N)$, $O(N^2)$ y $O(N^3)$ respectivamente
  - Puede ser dificil observar estos aumentos
    - Los coeficientes de términos no dominantes son grandes
    - Las funciones tienen tasas de crecimento próximas para tamaños pequeños

$$N \approx N \log(N)$$

La notación O

- Comprobación empírica
  - Calcular T(N)/F(N) para una serie de valores de N
    - Si converge a c>0, T(N)=O(F(N))
    - Si converge a 0, O(F(N)) es una sobreestimación para T(N)
    - Si diverge, O(F(N)) es una subestimación para T(N)

La notación O

- Comprobación empírica
  - N búsquedas aleatorias mediante búsqueda binaria

N	CPU Time *T* (milliseconds)	*T* / *N*	*T* / N^2	T / (*N* log *N*)
10,000	100	0.01000000	0.00000100	0.00075257
20,000	200	0.01000000	0.00000050	0.00069990
40,000	440	0.01100000	0.00000027	0.00071953
80,000	930	0.01162500	0.00000015	0.00071373
160,000	1,960	0.01225000	0.00000008	0.00070860
320,000	4,170	0.01303125	0.00000004	0.00071257
640,000	8,770	0.01370313	0.00000002	0.00071046

figure 5.13

Empirical running time for *N* binary searches in an *N*-item array

OK: converge a 0.0007

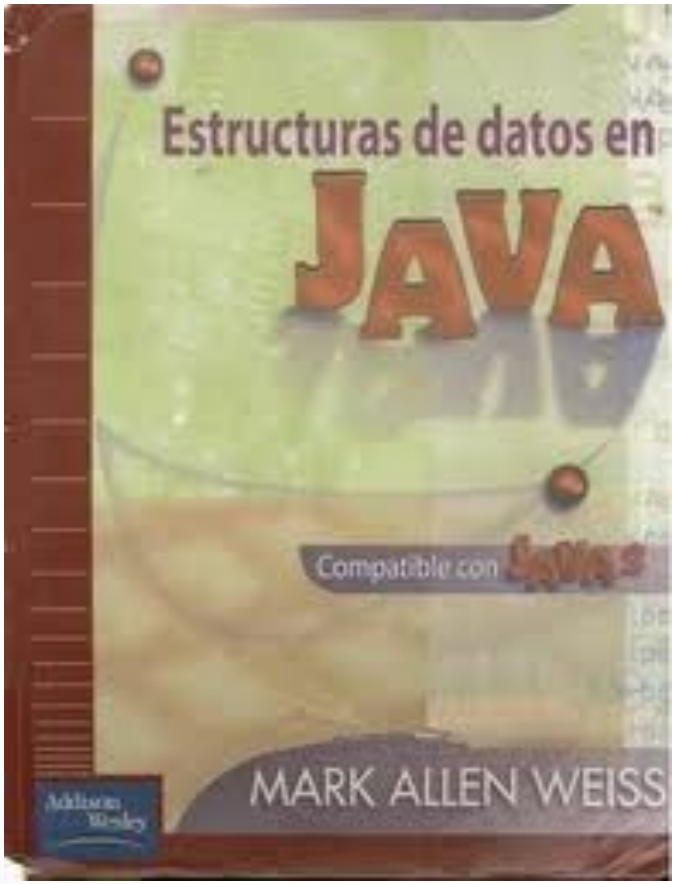
Sobreestimación: converge a 0

Subestimación: diverge

Bibliografía

Bibliografía básica

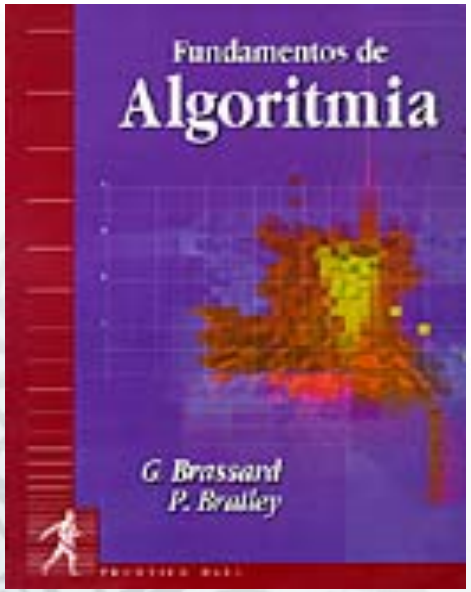
Estructuras de datos en java. Weiss, Mark Allen. Pearson Addison – Wesley. ISBN 9788478290352



Bibliografía

Bibliografía complementaria

Fundamentos de Algoritmia. G. Brassard, P. Bratley. Prentice Hall. SBN: 84-89660-00-X 1997



Diseño de programas. Formalismo y abstracción. Ricardo Peña Marí. Pearson - Prentice Hall. ISBN 13: 9788420541914