

Algoritmia
Grado en Ingeniería Informática del Software
Escuela de Ingeniería Informática – Universidad de Oviedo

Complejidad de los algoritmos

Juan Ramón Pérez Pérez

jrpp@uniovi.es

Técnicas de Diseño de Algoritmos

*Rosa Guerequeta, Antonio
Vallecillo*

*Servicio de Publicaciones de la
Universidad de Málaga, 1998*

Capítulo 1. La complejidad de los algoritmos





*Tengo varios algoritmos
para resolver un problema:*

*¿Cómo puedo comparar y
decidir cuál es el más
eficiente?*

Cómo se mide el tiempo de ejecución: **Estudio empírico**


- Medir tiempos de ejecución del algoritmo
- Obligatorio programar el algoritmo
- Medir tiempos sobre una máquina concreta
- Invertir tiempo para medir distintos casos

Cómo se mide el tiempo de ejecución: **Estudio analítico**

- Independiente del ordenador
- Sin necesidad de ejecutar el algoritmo
- Facilita una generalización para tamaños grandes

$T(n)$ – N° de **instrucciones básicas** que ejecuta el algoritmo para una entrada de tamaño n .

- **Tamaño de la entrada** (n) el número de componentes sobre los que se va a ejecutar el algoritmo.



***¿Qué pasa si luego
implementamos el
algoritmo con diferentes
lenguajes o en diferentes
arquitecturas de
procesador? ¿Seguirá
siendo válido este estudio
analítico?***

Principio de la invariancia

El tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no diferirán más que en una constante multiplicativa.

Cambio del algoritmo en función de la entrada

- Caso mejor - $T_{mejor}(n)$
- Caso peor - $T_{peor}(n)$
- Caso medio - $T_{medio}(n)$

$$T_{medio}(n) = \sum_{i=1}^{N_{casos}} p_i T_i(n)$$

N_{casos} – Número de entradas posibles para n

p_i - Probabilidad de cada caso de entrada

Búsqueda secuencial

¿Cuál es el caso mejor?

Caso mejor
 $a[0] == x$

```
public static int buscar(int[] a, int x) {  
    int i = 0;  
    int n = a.length;  
    while (i < n) {  
        if (a[i] == x) return i;  
        i++;  
    }  
    return -1;  
}
```

Instrucción	Num. Veces
$i = 0$	1
$n = a.length$	1
$i < n$	1
$a[i] == x$	1
return i	1

$x = 8$

$a =$

8	4	3	5	6	9	2	1
---	---	---	---	---	---	---	---

$$T(n) = 1 + 1 + 1 + 1 + 1 = 5 \rightarrow O(1)$$

Búsqueda secuencial

¿Cuál es el caso peor?

Caso peor
x no exista

```
public static int buscar(int[] a, int x) {  
    int i= 0;  
    int n= a.length;  
    while (i<n) {  
        if (a[i]==x) return i;  
        i++;  
    }  
    return -1;  
}
```

Instrucción	Num. Veces
i= 0	1
n= a.length	1
i<n	n+1
a[i]==x	n
i++	n
return -1	1

x= 7

a=

8	4	3	5	6	9	2	1
---	---	---	---	---	---	---	---

$$T(n) = 1 + 1 + (n+1) + n + n + 1 = 3n + 4 \rightarrow O(n)$$

Búsqueda secuencial

¿Cuál es el caso medio?

- Caso medio

- $T_{medio}(n) = \sum_{i=1}^{N_{casos}} p_i T_i(n)$
- $N_{casos} = n + \text{Caso de que no exista}$
- Suponemos:
 - $P_{no_existe} = \frac{1}{2}$
 - $P_{exista \text{ en una posición}} = \frac{1}{2} \cdot \frac{1}{n}$

$$T_{medio}(n) = \frac{1}{2}(3n + 4) + \frac{1}{2n}5 + \frac{1}{2n}(3 \cdot 2 + 4) + \dots + \frac{1}{2n}(3(n - 1) + 4) \rightarrow O(n)$$

Ejercicio 1.1: Analizar la búsqueda dicotómica

¿Cuáles son y que complejidad tienen?

- Caso mejor
- Caso peor
- Caso medio

```
public static int buscar(int[] a, int x) {  
    int inicio= 0;  
    int fin= a.length-1;  
    int medio;  
    while (inicio<=fin) {  
        medio= (inicio+fin)/2;  
        if (a[medio]==x) return medio;  
        else  
            if (a[medio]>x) fin= medio-1;  
            else inicio= medio+1;  
    }  
    return -1;  
}
```

Cómo se mide el consumo de memoria

$M(n)$ – N° de bytes que se necesitan para ejecutar un algoritmo.

*Especial atención a las **estructuras de datos** con múltiples elementos cuando se realizan **copias**.*

- Es frecuente que el consumo de memoria sea inversamente proporcional al tiempo de ejecución

Complejidades

Objetivo: clasificar funciones de tiempo de ejecución para que podamos compararlas

Sea $f: \mathbf{N} \rightarrow [0, \infty)$. Se define el conjunto de funciones de orden O de f como:

$$O(f) = \{g: \mathbf{N} \rightarrow [0, \infty) \mid \exists c \in \mathbf{R}, c > 0, \exists n_0 \in \mathbf{N} \bullet g(n) \leq cf(n) \forall n \geq n_0\}.$$

Diremos que una función $g: \mathbf{N} \rightarrow [0, \infty)$ es de orden O de $f(n)$ si $t \in O(f(n))$

Para entenderlo: realizar la representación gráfica

Propiedades fundamentales de $O(f(n))$

1. $O(C \cdot f) = O(f)$

2. Regla de la suma: $O(f_1 + f_2) = \max(O(f_1), O(f_2))$.

3. Regla del producto: $O(f_1 \cdot f_2) = O(f_1) \cdot O(f_2)$

4. $O(\log_a n) = O(\log_b n) = O(\log n)$

5. $O(\log(n^K)) = O(K \cdot \log n) = O(\log n)$

Cuidado: $O(\log n \cdot \log n \cdot \dots \cdot \log n) = O(\log^K n) \neq O(\log n)$

Jerarquía Órdenes de complejidad

Eficiente	$O(1)$	Constante
	$O(\log n)$	Logarítmica
	$O(n)$	Lineal
	$O(n \log n)$	
	$O(n^{1,x})$	
	$O(n^2)$	Cuadrática
	$O(n^2 \log n)$	
	$O(n^3)$	Cúbica
	$O(n^k)$	Polinómica
Ineficiente	$O(1,x^n)$	Exponencial
	$O(2^n)$	
	$O(n!)$	Factorial
	$O(n^n)$	

Cálculo de tiempos de ejecución

Un programa de complejidad temporal $O(f(n))$ tarda t_1 segundos para un tamaño de problema n_1 ¿Cuánto tarda para un tamaño superior n_2 ?

Relaciones a tener en cuenta

Sobre el tamaño del problema

$$(1) \quad n_2 = Kn_1 \Rightarrow K = \frac{n_2}{n_1}$$

Suponiendo que n_1 y n_2 son suficientemente grandes, el algoritmo se comporta según su complejidad. Sigue la siguiente proporción:

$$f(n_1) \text{ ---- } t_1$$

$$f(n_2) \text{ ---- } t_2 \text{ (que es la incógnita)}$$

Aplicación para el caso general: un algoritmo de complejidad $O(n)$

Si para $f(n_1)$ tarda $t_1 \rightarrow$ para $f(n_2)$ tardará t_2

$$(2) \quad t_2 = \frac{f(n_2)}{f(n_1)} \cdot t_1$$

- $f(n) = O(n^c) \rightarrow t_2 = K^c \cdot t_1$
- $f(n) = O(\log n) \rightarrow t_2 = \frac{\log K + \log n_1}{\log n_1} \cdot t_1$

Ejercicio 1.2: Cálculo de tiempo de ejecución

- Tenemos un método que itera a través de todos los elementos (números) de una matriz cuadrada de orden n , haciendo un cálculo (la complejidad de dicho cálculo es $O(1)$). Teniendo en cuenta la complejidad de la operación completa:
- Si para $n = 1\,000$ el método tarda 10 minutos, ¿cuánto tiempo tardaría si $n = 1\,000\,000$? ¿tardaría más o menos de 10 años?

Ejercicio 1.2B: Cálculo de tiempo de ejecución

- Tenemos un algoritmo de complejidad logarítmica $O(\log n)$.
- Si para $n = 1\,000$ el método tarda 10 minutos, ¿cuánto tiempo tardaría si $n = 1\,000\,000$? ¿tardaría más o menos de 10 años?

Solución: ¿?

Cálculo del aumento del tamaño del problema al aumentar el tiempo

Un programa de complejidad temporal $O(f(n))$ tarda t_1 segundos para un tamaño de problema n_1 . ¿Qué tamaño de problema se resolverá si tenemos un tiempo superior t_2 ?

Relaciones a tener en cuenta

Sobre el tiempo de ejecución del problema

$$(1) \quad t_2 = K t_1 \Rightarrow K = \frac{t_2}{t_1}$$

Suponiendo que n_1 y n_2 son suficientemente grandes, el algoritmo se comporta según su complejidad:

Si para $f(n_1)$ tarda $t_1 \Rightarrow$ para $f(n_2)$ tardará t_2

$$(2) \quad f(n_2) = \frac{t_2}{t_1} \cdot f(n_1) = K \cdot f(n_1) \\ \Rightarrow n_2 = f^{-1}(K \cdot f(n_1))$$

Ejercicio 1.3: Cálculo del tamaño del problema en función del tiempo

- Tenemos un algoritmo de complejidad cuadrática $O(n^2)$.
- Si para $n = 1\,000$ el método tarda 10 minutos, ¿Qué tamaño de problema podríamos resolver si disponemos de 2 horas y 40 minutos?

Reglas para el cálculo de la eficiencia (1): secuencias

Para calcular la eficiencia de un algoritmo partiendo de su código

Instrucciones de asignación, entrada/salida o expresiones aritméticas, siempre que no dependan del tamaño n del problema.

$$O(1)$$

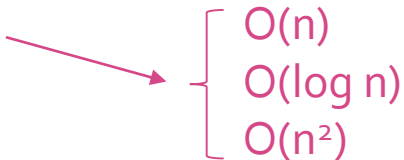
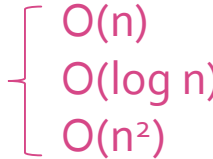
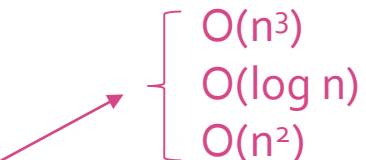
Estructura secuencial de instrucciones, se aplica la regla de la suma planteada en las propiedades fundamentales de O .

$$O(f_1(n)) + O(f_2(n)) = O(\max(f_1(n), f_2(n)))$$

Reglas para el cálculo de la eficiencia (2): bucles

Estructura iterativa de instrucciones, se aplica la regla del producto.

$$O(f_{bucle}(n)) \cdot O(f_{repeticiones}(n)) = O((f_{bucle}(n) \cdot f_{repeticiones}(n)))$$

1. `for (i = 1; i <= n; i++) { O(1) }` 
2. `for (i = 1; i <= n; i += 2) { O(1) }`
3. `for (i = 1; i <= n; i *= 2) { O(1) }` 
4. `for (i = 1; i <= n; i *= c) { O(1) }` 
5. `for (i = 1; i <= n*n*n/2; i *= c) { O(1) }`

Aplicación a bucles anidados (1)

1.

```
for (int i= 0; i< n; i++)  
    for (int j= 0; j<n; j++) { O(1) }
```

$\left\{ \begin{array}{l} O(n) \\ O(\log n) \\ O(n^2) \end{array} \right.$
2.

```
for(i = n/2; i <=n; i++)  
    for(j = 1; j <= n/3; j+=2) {O(1)}
```

$\left\{ \begin{array}{l} O(n) \\ O(\log n) \\ O(n^2) \end{array} \right.$
3.

```
for (i= 1; i<=n; i++)  
    for (j= 1; j<=i; j++) { O(1) }
```

$\left\{ \begin{array}{l} O(2n) \\ O(n^2) \\ O(n \log n) \end{array} \right.$
4.

```
for (i= 1; i<=n; i++)  
    for (j= n; j>=1; j--)  
        for (k=0; k<n; k++) { O(1) }
```

$\left\{ \begin{array}{l} O(n^3) \\ O(n^2) \\ O(n \log n) \end{array} \right.$

Reglas prácticas para el cálculo de la eficiencia (3)

Para calcular la eficiencia de un algoritmo partiendo de su código

Estructura condicional de instrucciones, aquí aparecen distintos casos en función de la alternativa que se ejecute:

- Caso **mejor** $\rightarrow O(\min(f_1(n), f_2(n), \dots, f_K(n)))$
- Caso **peor** $\rightarrow O(\max(f_1(n), f_2(n), \dots, f_K(n)))$
- Caso **medio** $\rightarrow \sum_{i=1}^K p_i O(f_i(n))$ Siendo: K , el número de alternativas.

Ejercicio 1.4: ¿Qué complejidades tienen estas estructuras de código?

1) `for (i = 1; i <= n; i++)`
 `for (j = 1; j <= i; j++)`
 `for (k = 1; k <= j; k++) { O(1) }`

$\left\{ \begin{array}{l} O(n^2) \\ O(n^2 \log n) \\ O(n^3) \end{array} \right.$

2) `for (i = 2*n; i >= 0; i -= 3)`
 `for (j = 1; j <= n*n*n; j *= 2) { O(1) }`

$\left\{ \begin{array}{l} O(n) \\ O(n^2) \\ O(n \log n) \end{array} \right.$

3) `for (i = n; i >= 1; i /= 2)`
 `for (j = 1; j <= n/3; j++) { O(1) }`

$\left\{ \begin{array}{l} O(n) \\ O(n^2) \\ O(n \log n) \end{array} \right.$

Para repasar y aprender algo más



Cómo ANALIZAR tus ALGORITMOS
(en Ingeniería Informática)

<https://youtu.be/IZgOECOnIbw>

¿Qué es eso del problema P
versus NP?

<https://youtu.be/UR2oDYZ-Sao>

