Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 18

Búsqueda y Ordenamiento

Búsqueda

- La búsqueda de elementos es una tarea prácticamente diaria.
- A lo largo de la materia hemos realizado búsquedas en varias ocasiones.
- Existen mejoras al proceso de búsqueda elemento a elemento. Una de ellas es realizar la búsqueda sobre los elementos de manera ordenada (**secuencia ordenada**). De esa manera, no será necesario recorrer todos los elementos cada vez que se necesita buscar uno.
- A continuación veremos **dos alternativas** para **buscar** sobre una **secuencia ordenada de elementos**. Utilizaremos los **arreglos** para **representar esos elementos**.

Búsqueda secuencial

- Es la búsqueda más sencilla.

- Analizaremos a continuación un algoritmo genérico, utilizando arreglos. Pero puede aplicarse la misma idea a LSE, LDE y Archivos (de texto y binarios).

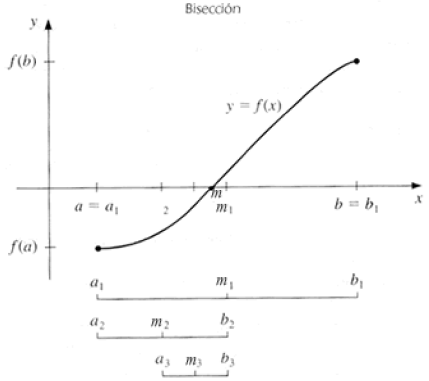
Búsqueda secuencial
Arreglo ordenado de tipo genérico

```
Algoritmo BusquedaSecuecial
Lexico
   s ∈ arreglo [limInf...limSup] de Telem
   e ∈ Telem
   i ∈ (limInf...limSup+1)
Inicio
  i ← limInf
  mientras (i<limSup+1 y s[i]<e) hacer
    i ← i+1
  fmientras
  según
    i>limSup: {e0 no está en s}
    i≤limSup: según
                 s[i]>e: {e0 no está en s}
                 s[i]=e: {e0 está en s[i]}
               fsegún
  fsegún
Fin
```

Búsqueda dicotómica o binaria

Se basa en el método de bisección (matemática).

Es uno de los métodos más sencillos y de fácil intuición para resolver ecuaciones en una variable. Se basa en el Teorema de los Valores Intermedios (TVI).
Básicamente el Teorema del Valor Intermedio nos dice que toda función continua en un intervalo cerrado, una vez que alcanzó ciertos valores en los extremos del intervalo, entonces debe alcanzar todos los valores intermedios.



Es el método más elemental y antiguo para determinar las raíces de una ecuación.

Consiste en partir de un intervalo $[a_1,b_1]$ se va reduciendo el intervalo sucesivamente hasta hacerlo tan pequeño como exija la precisión que hayamos decidido emplear.

Búsqueda dicotómica o binaria

Determina si una secuencia ordenada con acceso directo contiene un elemento **e** dado.

Principio general:

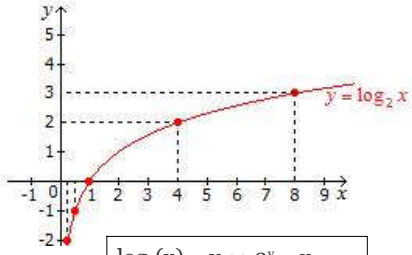
- se compara **e** con el elemento medio de la secuencia.
- si **e** es igual, entonces ha sido **hallado**
- si **e** es mayor que el elemento medio: se busca en la **mitad derecha** de la secuencia
- si **e** es menor que el elemento medio: se busca en la **mitad izquierda** de la secuencia.

Búsqueda dicotómica o binaria

- Los datos deben estar en un determinado orden, si los datos son compuestos, deben estar ordenados por al menos un campo (que se llamará clave).
- Para poder aplicar éste tipo de búsqueda es necesario contar con formas de acceso directo a los elementos.
- Por tal motivo, pueden utilizarse arreglos y archivos binarios, pero no LSE ni LDE.

Búsqueda dicotómica o binaria

Algoritmo BusquedaDicotomica
Léxico
  s ∈ arreglo [*limInf*...*limSup*] de *Telem*
  e ∈ *Telem*
  k, inf, sup ∈ (*limInf*...*limSup*)
Inicio
 según
  e<s[*limInf*] o e>s[*limSup*]: {e_0 *no está en S}*
  s[*limInf*]≤e≤s[*limSup*]: inf ← *limInf*
  sup ← *limSup*
  mientras inf<sup **hacer**
  k ← (inf+sup) div 2
  según
  e>s[k]: inf ← k+1
  e≤s[k]: sup ← k
  fsegún
  fmientras
  según
  e=s[inf]: {e_0 *está en s[k]*}
  otros: {e_0 *no está en s*}
  fsegún
 fsegún
Fin



$\log_2(x) = y \leftrightarrow 2^y = x$

$\log_2(1) = 0$
$\log_2(2) = 1$
$\log_2(4) = 2$
$\log_2(8) = 3$
$\log_2(16) = 4$
$\log_2(32) = 5$
$\log_2(64) = 6$
$\log_2(128) = 7$
$\log_2(256) = 8$

Búsqueda dicotómica o binaria

Comparación entre métodos de búsqueda

n es la cantidad de elementos a analizar

		Peor caso	Promedio
Asociativa o secuencial	No ordenado →	n	n
	Ordenado →	n	n/2
Binaria o dicotómica	→	$\log_2 n$	$\log_2 n$*

Para evaluar **complejidad** de los algoritmos se toma el **peor caso**.

Elementos necesarios en una búsqueda binaria:
$\log_2(n)$ donde n = cantidad de elementos a analizar.
Ejemplo: $\log_2(1.000.000) \approx 20$

*Este resultado es válido para el algoritmo dado, puede ser mejorado modificando la condición de terminación del ciclo.

Ordenamiento

¿Qué es ordenar?

Ordenar una secuencia es organizar sus elementos de modo que formen una secuencia no decreciente.

Para ello debe estar definida una relación de orden lineal $\leq$ entre los elementos.

Dada la secuencia $r_1, r_2, ..., r_n$ debe resultar la misma secuencia en orden $r_{i1}, r_{i2}, ..., r_{in}$ tal que $r_{i1} \leq r_{i2} \leq ... \leq r_{in}$

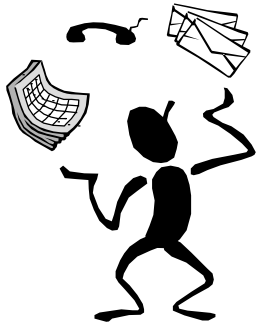
Ordenamiento

¿Para qué ordenar?

- **Facilitar los procesos de búsqueda.**
- **Elegir el algoritmo de tratamiento más adecuado. Búsqueda dicotómica, por ejemplo.**

Ejemplos

- **La Biblioteca de la UNRC ordena sus libros por número para encontrarlos más fácilmente.**
- **Los alumnos se ordenan alfabéticamente para tomar asistencia o examen.**



Ordenamiento
Características

- Los elementos a ordenar pueden ser ***simples*** o ***compuestos*** (registros).
- Asumiremos que cada elemento contiene al menos un campo llamado ***clave***.
- Cualquier ***campo*** puede ser ***clave*** mientras esté definida la relación de ordenamiento lineal ≤.
- Usaremos ***arreglos*** para analizar los distintos algoritmos de ordenamiento.

Ordenamiento
Nociones importantes

Estabilidad

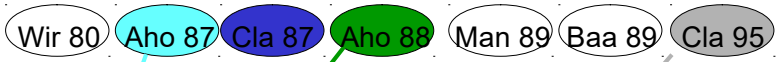
Un algoritmo de ordenamiento es *estable* cuando mantiene el orden relativo de los elementos con el mismo valor.

Eficacia o Eficiencia

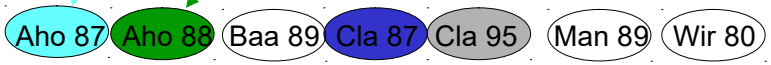
Está asociada a dos conceptos:
- Cantidad de pasos realizados
- Cantidad de movimientos realizados

Ordenamiento
Estabilidad

Supongamos que tenemos una secuencia de referencias bilbiográficas, compuestas por autor y año, **ordenadas** por el campo **año**.

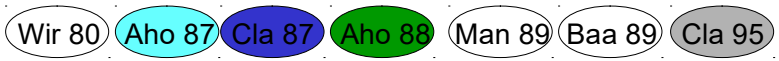


Si cambiamos el campo de **ordenamiento** de año por **autor** obtendremos los elementos de la secuencia dispuestos de otra manera, en este caso, **ordenados** por **autor**.



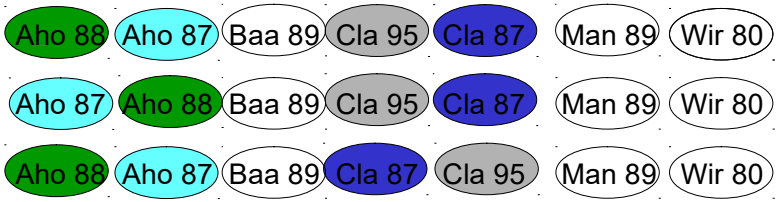
Que el algoritmo sea estable significa en este caso que si hay referencias bibliográficas que **comparten el campo clave** (Aho y Cla), las mismas deben **mantener el orden entre ellas**, es decir, las que estaban primeras (Aho 87 y Cla 87 en el ejemplo) deben seguir primeras respecto de segundas (Aho 88, Cla 95 en el ejemplo), y así sucesivamente.

Ordenamiento
Estabilidad



Estable

No Estable



Ordenamiento
Estabilidad

El concepto de **estabilidad** tiene sentido **solo** para secuencias cuyos **elementos** sean **compuestos**, es decir, tienen más de un campo, y son factibles de ser ordenadas por más de uno de esos campos (varios campos pueden ser clave).

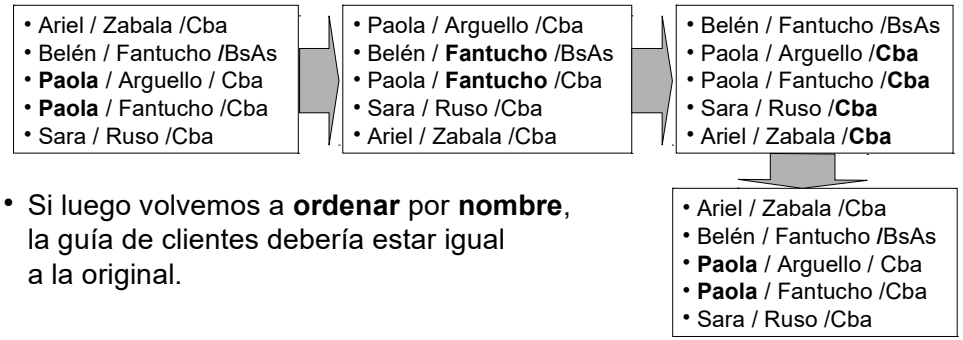
Otro caso clásico son las guías telefónicas o guías de cualquier otro tipo (clientes, proveedores, usuarios de un sistema, etc.)

Supongamos por ejemplo que tenemos **ordenada** una guía de clientes por **nombre** y luego por **apellido**.

Nombre		Apellido		Ciudad
Ariel	/	Zabala	/	Cba
Belén	/	Fantucho	/	BsAs
Paola	/	Arguello	/	Cba
Paola	/	Fantucho	/	Cba
Sara	/	Ruso	/	Cba

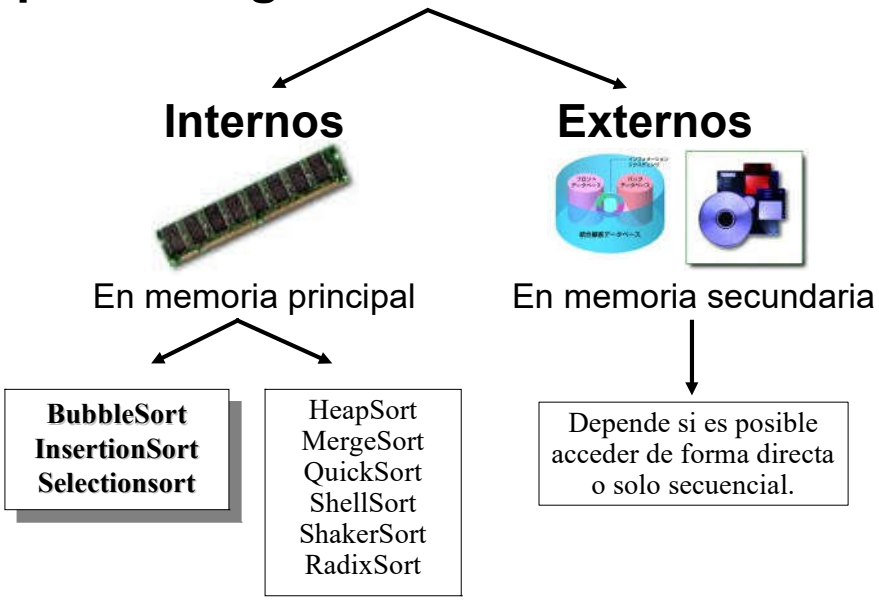
Ordenamiento
Estabilidad

- Si la ahora **ordenamos** solo por **apellido**, la estabilidad garantizaría que, en el caso de que haya apellidos repetidos, aparezcan primero los clientes cuyos nombres sean menores (según en el orden lexicográfico).
- Si luego la **ordenamos** solo por **ciudad**, la estabilidad garantizaría que, en el caso de que haya ciudades repetidas, se mantenga el orden entre los elementos que comparten la misma ciudad.



- Si luego volvemos a **ordenar** por **nombre**, la guía de clientes debería estar igual a la original.

Tipos de Algoritmos de ordenamiento



Algoritmos de ordenamiento

Los tres algoritmos básicos:

BubbleSort (Ordenamiento por el método de Burbuja o Intercambio)

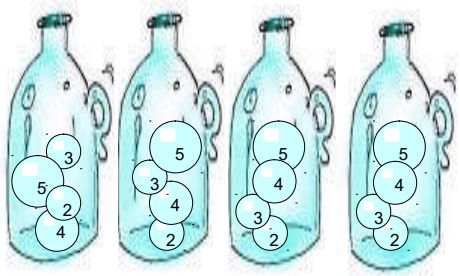
InsertionSort (Ordenamiento por Inserción)

SelectionSort (Ordenamiento por Selección)

BubbleSort (Burbuja)

Idea: El valor más grande de la secuencia gradualmente se va desplazando hacia la última posición (como una burbuja sube en una botella), mientras los valores más pequeños se van hacia adelante (el fondo de la botella).

Las burbujas (elementos) se comparan de a pares, realizado el **intercambio** entre ellos solo cuando corresponda.



Se recorre la secuencia S llevando el mayor elemento encontrado hacia la última posición. Luego se repite el proceso para la subsecuencia resultante de no considerar ese último elemento.

Para Knuth es un algoritmo decepcionante, solo famoso por su nombre [Wir80].

Algoritmo BubbleSort

Algoritmo Burbuja
Lexico
  N = 4
  s ∈ arreglo [1...N] de *Telemento*
  i, j ∈ (1...N+1)
Acción Intercambiar (**dato-resultado** x,y ∈ *Telemento*)
{Pre-cond: $x=x_0 \wedge y=y_0$} / {Pos-cond: $x=y_0 \wedge y=x_0$}
Inicio
{Pre-cond: $s[1...n]=s_0$}
  i ← n
  mientras i>1 **hacer**
    j ← 1
    mientras j<i **hacer**
      si s[j]>s[j+1] **entonces** Intercambiar(s[j],s[j+1])
      fsi
      j ← j+1
    fmientras
    i ← i-1
  fmientras
{Pos-cond: $perm(s,s_0) \wedge ordenado(s[1...n])$}
Fin

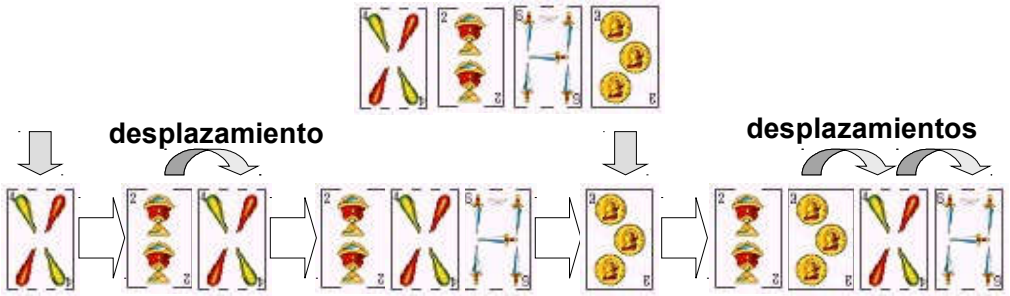
Algoritmo BubbleSort (cont.)

Traza con N=4

	Indices					inter-des.	comp.	iter.
	1	2	3	4				
	4	2	5	3		1	1	1
	2	4	5	3		1	2	1
	2	4	5	3		2	3	1
	2	4	3	**5**		2	3	1
	2	4	3	**5**		2	4	2
	2	4	3	**5**		3	5	2
	2	3	**4**	**5**		3	5	2
	2	3	**4**	**5**		3	6	3
	2	**3**	**4**	**5**		3	6	3

InsertionSort (Inserción)

Idea: Sigue la lógica de un jugador de cartas: Toma una carta a la vez y la ordena respecto a las cartas con las que ya cuenta en la mano.



Los valores son considerados uno a la vez, y cada nuevo valor a ordenar es **insertado** en su posición adecuada respecto a los valores ordenados previamente [Knu76].

Cada valor S[i] es insertado en forma ordenada entre los valores de la subsecuencia ordenada S[1...i-1]

Algoritmo InsertionSort (cont.)

Algoritmo Inserción
Lexico
  N=4
  s ∈ arreglo [1...N] de *Telemento*
  aux ∈ *Telemento*
  i, j ∈ (1...N+1)
Inicio
{Pre-cond $s[1...n]=s_0$}
  i ← 2
  mientras i≤n **hacer**
    aux ← s[i]
    j ← i-1
    mientras j>0 y s[j]>aux **hacer**
      s[j+1] ← s[j]
      j ←j-1
    fmientras
    s[j+1] ← aux
    i ← i+1
  fmientras
{ Pos-cond: $perm(s,s_0) \wedge ordenado(s[1...n])$}
Fin

Algoritmo InsertionSort (cont.)

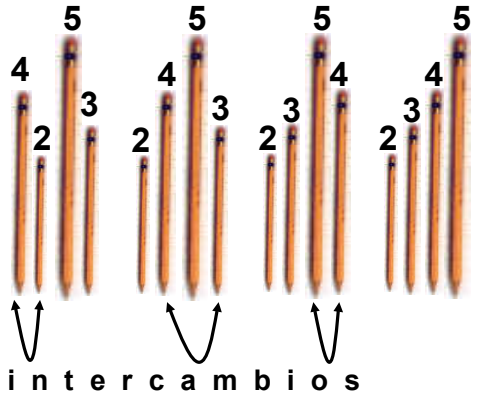
Traza con N=4

	Indices					inter-des.	comp.	iter.
	1	2	3	4				
aux 2	4	2	5	3			1	1
aux 2	4	4	5	3		1	1	1
aux 5	2	4	5	3		1	2	2
aux 3	2	4	5	3		1	3	3
aux 3	2	4	5	5		2	3	3
aux 3	2	4	4	5		3	3	3
aux 3	2	3	4	5		3	3	3

SelectionSort (cont.)

Idea: Consiste en **seleccionar** la **posición** del elemento **menor** de la secuencia y, una vez encontrada, coloca el elemento que está en dicha posición al comienzo de la secuencia (**intercambiando** los elementos).

Luego se busca nuevamente el menor, pero sin considerar la primera posición, y lo **intercambia** con el valor que se encuentra en la segunda posición, y así sucesivamente.



Busca el menor elemento de la subsecuencia no ordenada s[i...n] y lo intercambia con s[i].

SelectionSort (Selección)

```
Algoritmo Selección
Lexico
  N=4
  s ∈ arreglo [1...n] de Telemento
  i, j ∈ (1...n+1), min: (1...N) {índice del menor}
Acción Intercambiar(dato-resultado x,y ∈ Telemento)
{Pre-cond: x=x0 ∧ y=y0} {Pos-cond: x=y0 ∧ y=x0}
Inicio
{Pre-cond: s[1...n]=s0}
  i ← 1
  mientras i≤n hacer
    j ← i
    min ← i
    mientras j≤n hacer
      si s[j]<s[min] entonces min ← j  fsi
      j ← j+1
    fmientras
    Intercambiar (s[i], s[min])
    i ← i+1
fmientras
{Pos-cond: perm(s,s0) ∧ ordenado(s[1...n])}
Fin
```


SelectionSort (cont.)

Traza con N=4

	Indices					inter-des.	comp.	iter.
	1	2	3	4				
	4	2	5	3		0	3	1
	2	4	5	3		1	3	1
	2	4	5	3		1	5	2
	2	3	5	4		2	5	2
	2	3	5	4		2	6	3
	2	3	4	5		3	6	3

¿Cuál es el mejor Algoritmo?

Diversos factores

- Tamaño de la secuencia. Porcentaje.
- Tamaño de los elementos.
- Tiempo de corrida crítico.
- Recurso de cómputo disponible.
- Tiempo de utilización vs. Tiempo de desarrollo.

	Estabilidad	Eficacia o Eficiencia	
		# pasos	# intercambios o desplazam.
BubbleSort	Sí	n^2	n^2
InsertionSort	Sí	n^2	n^2
Selectionsort	No	n^2	n

Se calcula la cantidad de pasos e intercambios (o desplazamientos) para secuencias de *n* elementos considerando en el PEOR CASO.

Bibliografía

www.bib.unrc.edu.ar

- [Cla87] **Clavel, G. y Biondi, J., "Introducción a la Programación. Tomo 2: Estructuras de Datos"**, Masson, 1987. Capítulo 2. Pags. 33 a 60. **Comentario**: nivel y profundidad adecuado para nuestra materia. En Biblioteca: buscar por: 681.3.068 B 615 v.2
- [Wir80] **Wirth, N. "Algoritmos + Estruturas de Datos = Programas"**. Ediciones del Castillo, 1980. **Comentario**: Estudio detallado pero fácilmente abordable. En Biblioteca: buscar por: 681.3.068 W 799a.
- [Aho87] **Aho, A., J. Hopcroft and J. Ullman, "Data Structures and Algorithms"**, Reading MA, Addison-Wesley, 1987. Chapter 8. Pags. 253 a 260. **Comentario**: Buen análisis de los algortimos básicos y otras más avanzados. En Biblioteca: buscar por: 511.8 A 286. Versión en castellano en librerías.
- [Man89] **Manber, U. "Introduction to Algorithms. A Creative Approach".** Addison-Wesley, 1989. Chapter 6. Pags. 127 a 130 **Comentario**: se concentra en otros algoritmos más complejos e interesantes. En Biblioteca: buscar por: 511.8 M 269
- [Baa89] **Baase, S. "Computer Algorithms. Introduction to Design and Analysis".** Second Edition. Addison-Wesley, 1989. Chapter 2. Pags. 47 a 51. **Cometario**: análisis exhaustivo del algoritmo de Inserción. En Biblioteca: no disponible.
- [Wei95] **Weiss, M. "Data Structures y Algorithm Analysis".** Second Edition. Benjamin/Cummings. 1995. Chapter 7. Pags. 213 a 215. **Cometario**: formalizaciónlos algoritmos y análisis exhaustivo del de Inserción. En Biblioteca: buscar por: 681.3.068 B 615
- [Knu73] **Knuth, D. E. "Ther Art of Computer Programming. Volume 3: Sorting and Searching"**. Addison Wesley, 1973. **Comentario**: realiza un análisis profundo. En Biblioteca: buscar por: 519.4 K 74 v.3

Enlaces Recomendados

- http://www.cs.brockport.edu/cs/javasort.html State University of New York College at Brockport.

Cometario: Explicación de cada método. Animación on line y Código Java. Muy didáctico.

- http://olli.informatik.uni-oldenburg.de/ Universitat Olderburg. Alemania

Cometario: Animación de Bubblesort. Muy didáctico, con burbujas y código paso a paso.

- http://www.cs.ubc.ca/spider/harrison/Java/ University of British Columbia. Vancouver.

Cometario: Demostración gráfica on line y código en java de gran cantidad de métodos.

- http://www.cs.rit.edu/~atk/Java/Sorting/sorting.html Rochester Institute of Technology.

Comentario: Animaciones de algunos algoritmos. Del mismo estilo que el anterior.

- http://icc2.act.uji.es/e02/E02/T7 Universitat Jaume I de Castellón.

Comentario: material teórico en formato pdf. En castellano.

- http://www.monografias.com/trabajos/algordenam/algordenam.shtml

Comentario: Explicación breve del tema y clasificación de los algoritmos. En castellano.

- http://labcqmaster.qro.itesm.mx/~compu2

Comentario: Código en C++. Ejecutables con ejemplos de los tres.

- http://c.conclase.net/orden/index.html

Comentario: Explicación general, los algoritmos básicos con ejemplos, ventajas, desventajas y código en C. En castellano. Ojo!!! Tiene errores en el tema de Estabilidad.