

# Algoritmos Avanzados

---

Asignatura: Introducción a la Ingeniería en Computación  
Prof. Mónica Saettone

## Contenido

Propiedades.....	2
Representación de los Vectores en pseudocódigo: .....	2
Pseudocódigo para agregar elementos a un arreglo: .....	3
Pseudocódigo para mostrar los elementos de un arreglo: .....	3
Ejemplo de la ejecución del algoritmo mostrar los elementos de un arreglo: .....	3
Ejemplos varios: .....	5
Algoritmos basados en el intercambio de posiciones.....	5
Algoritmo de intercambio de afuera hacia dentro (invertir arreglo).....	5
Métodos de Búsqueda .....	6
Búsqueda Lineal .....	6
Búsqueda Binaria (dividir para vencer) .....	6
Algoritmos de Ordenación .....	9
Algoritmo de ordenamiento por el método de la Burbuja: <i>Bubble Sort</i> .....	9
Algoritmo de ordenamiento por el método de Inserción: <i>Insertion Sort</i> .....	10
Algoritmo de ordenamiento por el método de Selección .....	11
Ejercicios: .....	12

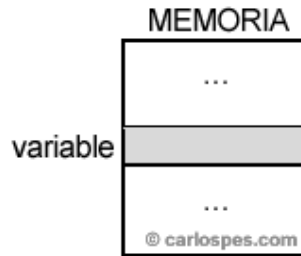
## Utilizando vectores en la construcción de algoritmos

---

En la guía anterior, Introducción a los Algoritmos, se explicó la representación de distintos tipos de datos en la construcción de algoritmos, como lo son: Entero, real, lógico, carácter, pero el tipo de datos Cadena, solo fue mencionado, por lo que en este apartado se explicará en detalle.

El tipo de datos Cadena, también conocido como vector o arreglo, es un tipo de datos compuesto que tiene características especiales.

Se sabe que en programación, una variable representa a un espacio de memoria en el cual se puede almacenar un dato. Gráficamente, se puede representar como:



Por lo que un arreglo de N posiciones, ocupa el espacio de N variables.

Tomado de: <http://www.scribd.com/doc/3207178/Arreglos>

Un vector es un conjunto de elementos del mismo tipo que comparten un nombre común, como una variable que puede almacenar al mismo tiempo más de un valor. Los vectores reciben también el nombre de tablas, listas o arrays.

Un vector es un conjunto ordenado y homogéneo. Ordenado porque el primer elemento, segundo, tercero, ..., n-ésimo puede ser identificado y homogéneo porque sus elementos son todos del mismo tipo (numéricos o alfanuméricos, pero no una combinación de ambos).

El tipo más simple de array es el denominado array unidimensional o vector. Es unidimensional porque sólo se necesita un subíndice o índice para designar la posición de un elemento dentro del array. Existen datos que están mejor representados en forma de tablas o matrices con dos o más subíndices. Gráficamente, un vector se representa como una tabla. De igual forma que cualquier variable, un vector debe tener un nombre.

A 

12	43	25	68	10	5	3	17	87	24
----	----	----	----	----	---	---	----	----	----

Los elementos que están en el vector A ocupan todos, una determinada posición dentro de él:

A 

	0	1	2	3	4	5	6	7	8	9
	12	43	25	68	10	63	3	92	87	24

Así, el número "68" se encuentra en la posición 3, el "92" en la posición 7.  $A(3) = 68$ ;  $A(7) = 92$ .

## Propiedades

- Los datos individuales de un vector se denominan *elementos*.
- Todos los elementos deber ser del mismo tipo de datos.
- Todos los elementos se almacenan en posiciones contiguas de la computadora y el subíndice (o índice) del primer elemento es cero (0).
- El nombre de un vector es un valor constante que representa la dirección del primer elemento del vector.

## Representación de los Vectores en pseudocódigo:

La sintaxis de los arreglos o vectores en la construcción de pseudocódigos se representa de la siguiente manera:

### Arreglo (índice)

Donde **Arreglo** representa el nombre de la variable, luengo y entre paréntesis el valor del **índice**. Es decir que para obtener el valor que se encuentra en la posición 6, se debe escribir **A(6)**.

### Pseudocódigo para agregar elementos a un arreglo:

La carga de un vector se hace por medio de la estructura de repetición *desde...fin\_desde*.

<i>Pseudocódigo</i>	
Variables	
i : entero	
arreglo Vector(): entero	
<b>Inicio</b>	
<b>Desde</b> i = 0 hasta 9	
<b>Leer</b> Vector(i)	
<b>Fin_desde</b>	
<b>Fin</b>	

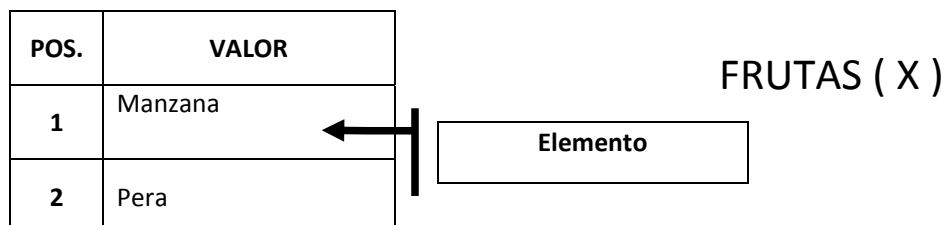
### Pseudocódigo para mostrar los elementos de un arreglo:

<i>Pseudocódigo</i>	
Variables	
i : entero	
arreglo Vector(): entero	
<b>Inicio</b>	
<b>Desde</b> i = 0 hasta 9	
<b>Escribir</b> Vector(i)	
<b>Fin_desde</b>	
<b>Fin</b>	

### Ejemplo de la ejecución del algoritmo mostrar los elementos de un arreglo:

Asumiendo que tenemos el siguiente arreglo llamado FRUTAS. El cual tiene los siguientes valores:

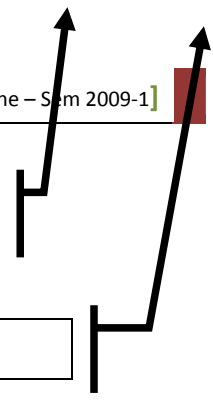
Tomado de: <http://www.uprb.edu/profesor/ntorres/sici-3012-modulo-recorrer-arreglo.doc>



3	Uva
4	Toronja
5	Guineo
6	Mango
7	Quenepa

Arreglo

Indice



El siguiente modulo en Pseudocódigo nos permite recorrer la lista para mostrar todos los valores del arreglo.

Variables

x : entero  
 arreglo x(): cadena

**Inicio**

**Escribir** "Listado de frutas"

**Desde** x = 1 hasta 7

**Escribir** x + ":" FRUTAS( x )

**Fin\_desde**

**Fin**

La ejecución de este algoritmo mostrará:

```
Listado de Frutas

1: Manzana
2: Pera
3: Uva
4: Toronja
5: Pera
6: Mango
7: Quenepa
```

## Ejemplos varios:

### Algoritmos basados en el intercambio de posiciones

En un arreglo, una operación fundamental es poder intercambiar los valores de dos posiciones  $i$  y  $j$ , o sea dejar en la posición  $i$  lo que había en la posición  $j$  y viceversa.

Para realizar esta operación, usamos el siguiente algoritmo:

```
1. tmp <-- a(i)
2. a(i) <-- a(j)
3. a(j) <-- tmp
```

Paso a paso:

1. guardamos en tmp el contenido de la posición  $i$
2. asignamos lo que hay en la posición  $j$ , a la posición  $i$
3. rescatamos lo que había en la posición  $i$  (tmp), asignándolo a la posición  $j$

El paso siguiente, es poder hacer a través de un ciclo iterativo un intercambio de posiciones de afuera hacia dentro.

Aquí la condición del ciclo, es que  $i$  tome un valor inicial de cero,  $j$  tome un valor inicial de la longitud del arreglo  $-1$ , y que el ciclo se ejecute mientras  $i$  sea menor que  $j$ .

Ahora bien, para que el ciclo termine alguna vez, en cada iteración hay que incrementar la variable  $i$ , y decrementar la variable  $j$ .

### Algoritmo de intercambio de afuera hacia dentro (invertir arreglo)

```
Variables
  i, j : entero
  arreglo a(): entero
Inicio

// asumiendo que el arreglo a ya se encuentra cargado con datos

  i <-- 0
  j <-- Longitud(a) - 1
  Mientras(i > j) hacer

    tmp <-- a(i)
    a(i) <-- a(j)
    a(j) <-- tmp

    i <-- i + 1
    j <-- j - 1

  Fin mientras
Fin
```

## Métodos de búsqueda y Ordenación

Los problemas más comunes en la informática son la búsqueda y la ordenación. Por lo tanto, la eficiencia de la búsqueda es importante. La ordenación consiste en ordenar los elementos de un conjunto con el fin de acelerar la búsqueda.

- Búsqueda lineal
- Búsqueda binaria
- Esquemas sencillos de ordenación
  - Algoritmo de la Burbuja: Bubble Sort
  - Algoritmo de Inserción: Insertion Sort
  - Algoritmo de Selección: Selection Sort

## Métodos de Búsqueda

### Búsqueda Lineal

La búsqueda lineal es un algoritmo para encontrar un elemento determinado dentro de una colección dada (arreglo), por lo tanto cumple con ciertas características:

- La búsqueda se realiza sobre una estructura de datos de tamaño fijo y conocido, por ejemplo, un vector.
- Los algoritmos sirven para hacer búsquedas sobre cualquier tipo de datos, siempre que sea posible realizar comparaciones ('igualdad', 'menor que') sobre este tipo.

#### Algoritmo de Búsqueda Lineal

##### Variables:

arreglo  $V(10)$  : entero

$i, n, \text{elem}$  : entero

##### Inicio

$i \leftarrow 0$  //contador

$n \leftarrow 10$  // cantidad de elementos

$\text{elemBuscado} \leftarrow 6$

**Mientras** ( (  $i < n$  ) **y** (  $V(i) \neq \text{elemBuscado}$  ) ) **hacer**  
 $i \leftarrow i + 1$

**Fin mientras**

**Si** (  $i \leq n$  ) **entonces**

mostrar "El elemento buscado es: " +  $V(i)$

**sino**

mostrar "No se encontró el valor buscado"

**Fin si**

**fin**

Podemos utilizar la estructura repetitiva **Para** en lugar de **Mientras**, reemplazando por:

**Para**  $i=0$  **Hasta**  $n-1$

Si  $\text{ElemBuscado} = V(i)$  entonces

mostrar "El elemento buscado es: " +  $V(i)$

**Proximo**

### Búsqueda Binaria (dividir para vencer)

La búsqueda lineal es la primera idea que ocurre para el problema de la búsqueda. Sin embargo, su eficiencia puede ser mejorado de forma considerable. Para hacerlo es necesario

suponer que los elementos del vector estén *ordenados*. Suponemos que el vector esta ordenado de forma *ascendente* (de menor a mayor).

Si el vector está ordenado (de manera ascendente o descendente), es posible aplicar búsqueda binaria. La idea es hacer servir la propiedad adicional del vector para acelerar el proceso de búsqueda:

1. Dividir el vector en dos partes iguales.
2. Si el elemento en el centro del vector es mayor que el elemento buscado, buscar en la primera mitad.
3. Si el elemento en el centro del vector es menor que el elemento buscado, buscar en la segunda mitad.

### Algoritmo de Búsqueda Binaria

Variabes

Arreglo V() : entero  
 elemBuscado, n : entero  
 E, D, medio : entero

**Inicio**

```

n ← 10 // cantidad de elementos
Inf ← 0
Sup ← n - 1
elemBuscado ← 6

mientras (Inf <= Sup) hacer
  medio ← (inf+sup) / 2
  Si( V(medio)= clave)
    mostrar "El elemento buscado es: " + V( medio )
    Salir mientras
  Sino
    Si (ElemBuscadoe < V(medio))
      sup ← medio - 1
    Sino
      inf ← medio
    fin si
  fin si
fin mientras
  
```

La búsqueda binaria **sólo se puede implementar si el arreglo está ordenado**. La idea consiste en ir dividiendo el arreglo en mitades.

Por ejemplo supongamos que tenemos este vector:

Arreglo vector(10) = {2,4,6,8,10,12,14,16,18,20}

El elemento que queremos buscar es 6.

El algoritmo funciona de la siguiente manera:

1. Se determinan un indice superior y un indice inferior, Inf=0 y Sup=9 respectivamente.
2. Se determina un indice central, medio = (Sup+Info)/2, en este caso quedaría medio = 4.
3. Evaluamos si V(medio) es menor al elemento buscado, si es igual ya encontramos la clave y Mostramos elemento V(medio).
4. Si son distintos, evaluamos si V(medio) es mayor o menos que el elemento buscado, como el arreglo está ordenado al hacer esto ya podemos descartar una mitad del arreglo asegurandonos que en esa mitad no está la clave que buscamos. En nuestro

caso  $V(\text{medio}) = 4 < 6$ , entonces la parte del arreglo  $V(0 \dots 4)$  ya puede descartarse.

5. Reasignamos  $\text{Sup}$  o  $\text{Inf}$  para obtener la nueva parte del arreglo en donde queremos buscar.  $\text{Sup}$ , queda igual ya que sigue siendo el tope.  $\text{Inf}$  lo tenemos subir hasta 5, entonces quedaría  $\text{Sup} = 9, \text{inf} = 5$ . Y volvemos al paso 2.

Si elemento buscado no fue encontrada en algún momento  $\text{Inf} > \text{Sup}$ , y no se cumple la condición de la sentencia Mientras y se sale del ciclo en tal caso.

Otro Ejemplo de algoritmo de búsqueda binaria tomado de: <http://www.dtic.upf.edu/~jonsson/pll09/Apuntes/TeoriaSemana1-2.ppt>

Variables

Arreglo  $V()$  : entero  
 elemBuscado,  $n$  : entero  
 Inf, Sup, medio : entero

**Inicio**

$n \leftarrow 10$  // cantidad de elementos  
 $\text{Inf} \leftarrow 0$   
 $\text{Sup} \leftarrow n - 1$   
 elemBuscado  $\leftarrow 6$

mientras ( $\text{inf} \leq \text{Sup}$ ) hacer  
     medio  $\leftarrow (\text{inf} + \text{sup}) / 2$   
     si ( $V(\text{medio}) < \text{elemBuscado}$ ) entonces  
          $\text{inf} \leftarrow \text{medio} + 1$   
     sino  
          $\text{sup} \leftarrow \text{medio}$   
     Fin si  
 Fin mientras  
 si ( $(\text{inf} = n + 1) \underline{\text{O}} (V(\text{inf}) \neq \text{elemBuscado})$ ) entonces  
     mostrar "No se encontró el valor buscado"  
  
 sino  
     mostrar "El elemento buscado es: " +  $V(\text{inf})$   
 fin si

**Fin**

Otro ejemplo, tomado de: <http://latecladeescape.com/w0/con-nombre-propio/busqueda-dicotomica-o-binaria-y-sus-peligros.html#ixzz0Kxz3IFR6&C>

**Algoritmo de Búsqueda Binaria**

```
variables:
  arreglo v():entero
  b:entero
  inferior, superior, medio:entero
  resultado:entero;

inicio
  //empezamos suponiendo que no hemos
  //encontrado el valor
  resultado=-1;
  //el primer elemento que consideramos
  inferior=1;
  //el ultimo elemento que consideramos
  superior=n;
```

```

mientras(resultado<0 y inferior<=superior)
  //calculamos la posición del elemento medio
  //entre inferior y superior
  medio=(inferior+superior)/2; //division entera
  si v[medio]=x //si lo hemos encontrado en medio
    resultado=medio
  si no
    si b<v[medio] //esta en la mitad inferior
      superior=medio-1
    si no //esta en la mitad superior
      inferior=medio+1
    fin si
  fin si
fin mientras
devolver resultado
fin
    
```

## Algoritmos de Ordenación

- Algoritmo de la Burbuja: Bubble Sort
- Algoritmo de Inserción: Insertion Sort
- Algoritmo de Selección: Selection Sort

Se ha demostrado que la búsqueda se puede realizar con más eficiencia si los elementos están ordenados. Si necesita buscar muchos datos en un mismo conjunto, vale la pena ordenar los elementos primero. Igual que para la búsqueda, la ordenación se puede realizar sobre cualquier tipo de elementos, siempre que se puedan comparar ('menor que').

Como la ordenación es un problema importante, existen un gran número de algoritmos de ordenación. Los algoritmos existentes se pueden utilizar en diferentes estructuras de datos (p.ej., un vector). Imponen diferentes tipos de requerimientos sobre los datos a ordenar. También varía su eficiencia, tanto a nivel de memoria como a nivel de tiempo de ejecución.

## Algoritmo de ordenamiento por el método de la Burbuja: *Bubble Sort*

El algoritmo de *ordenación de la burbuja* es uno de los más fáciles de recordar.

Su nombre describe de manera intuitiva su funcionamiento.

Imaginamos que los números menores 'pesan menos' y 'suben a la superficie' como una burbuja.

Se basa en el intercambio entre pares de items

Esta técnica de ordenación compara elementos consecutivos de la lista, de modo que si en una pasada no ocurrieran intercambios, significaría que la lista esta ordenada. Este método es muy clásico y sencillo

3	1	1	1	1	1	
9	3	2	2	2	2	
7	9	3	3	3	3	
1	7	9	5	5	5	
5	2	7	9	7	7	
2	5	5	7	9	9	
	<i>Inicio</i>	<i>i=1</i>	<i>i=2</i>	<i>i=3</i>	<i>i=4</i>	<i>i=5</i>



**Método de ordenamiento Burbuja****Variables**

Arreglo a() : entero

i, j, temp : entero

**Inicio****Para** i ← 0 **hasta** n-2 **hacer****Para** j ← i+1 **hasta** n-1 **hacer****Si** a(i) > a(j) **entonces:**

temp ← a(i)

a(i) ← a(j)

a(j) ← temp

**fin si****fin para****fin para****fin**

Links recomendados para ver animación de método de la burbuja:

- <http://www.youtube.com/watch?v=myKIT30n15Y>
- <http://mis-algoritmos.com/wp-content/uploads/2006/08/burbuja.php>
- [http://sziami.cs.bme.hu/~gsala/alg\\_anims/3/bsort-e.html](http://sziami.cs.bme.hu/~gsala/alg_anims/3/bsort-e.html)
- <http://www.cs.ubc.ca/spider/harrison/Java/>
- <http://www.cs.hope.edu/~alganim/animator/Animator.html>

**Algoritmo de ordenamiento por el método de Inserción: *Insertion Sort***

Este algoritmo se basa en hacer *comparaciones*, así que para que realice su trabajo de ordenación son imprescindibles dos cosas: un arreglo o estructura similar de elementos comparables y un criterio claro de comparación, tal que dados dos elementos nos diga si están en orden o no.

Es un algoritmo *estable* de ordenación *interna* y su complejidad temporal en el peor caso es de  $O(n^2)$  y en el mejor caso -que el arreglo ya esté totalmente ordenado- puede llegar a  $\Omega(n)$  siendo n el tamaño del arreglo a ordenar.

El algoritmo consiste en realizar varias pasadas sobre el arreglo. En cada pasada se analiza un elemento, y se intenta encontrar su orden relativo entre los analizados en pasadas anteriores. Con esto se logra ir manteniendo una lista ordenada constantemente. Cada elemento a analizar se desplaza por esa lista hasta encontrar su lugar. Cuando todos los elementos del arreglo han sido analizados, la lista está completamente ordenada. Esa es **una diferencia importante** con el algoritmo de la Burbuja y el de Selección. En ellos, en cada pasada se colocaba una carta en su sitio definitivo. En InsertionSort, el arreglo no está totalmente ordenado hasta que el algoritmo termina.

Tomado de: <http://latecladeescape.com/w0/con-nombre-propio/ordenacion-por-insercion-directa-insertionsort.html#ixzz0KyosorX4&C>

**Algoritmo de ordenamiento método de inserción****Variables**

Arreglo a() : entero  
 i, j, temporal : entero

**Inicio**

```

//estas son las pasadas, desde 2 hasta n
//en cada una intentaremos encontrar la posición
//relativa del elemento i entre los anteriores
para i ← 2 hasta n
  j=i-1
  //vamos "descendiendo" el elemento
  //haciendo intercambios
  mientras ( j>= 1) Y (A(j) > A(j+1)) hacer
    //intercambio de la posición j y la siguiente
    Temporal ← A(j+1)
    A(j+1) ← A(j)
    A(j) ← temporal
    J ← j-1
  fin mientras
fin para
fin

```

Links recomendados para ver animación de método de inserción:

- <http://www.youtube.com/watch?v=gTxFvgvZmQs>
- [http://sziami.cs.bme.hu/~gsala/alg\\_anims/3/isort-e.html](http://sziami.cs.bme.hu/~gsala/alg_anims/3/isort-e.html)
- <http://www.cs.ubc.ca/spider/harrison/Java/>
- <http://www.cs.hope.edu/~algnim/animador/Animator.html>

**Algoritmo de ordenamiento por el método de Selección**

El ordenamiento por selección es un algoritmo de ordenamiento que requiere  $O(n^2)$  operaciones para ordenar una lista de  $n$  elementos.

Su funcionamiento es el siguiente:

- Buscar el mínimo elemento de la lista
- Intercambiarlo con el primero
- Buscar el mínimo en el resto de la lista
- Intercambiarlo con el segundo

Y en general:

- Buscar el mínimo elemento entre una posición  $i$  y el final de la lista
- Intercambiar el mínimo con el elemento de la posición  $i$

De esta manera se puede escribir el siguiente pseudocódigo para ordenar una lista de  $n$  elementos indexados desde el 1:

**Variables**

Arreglo Lista() : entero

i, minimo, temporal : entero

**Inicio****para** i ← 1 **hasta** n-1

minimo ← i;

**para** j ← i+1 **hasta** n        **si** lista[j] < lista[minimo] **entonces**

minimo ← j

**fin si**    **fin para**

Temporal ← Lista(i)

Lista(minimo) ← Lista(i)

Lista(minimo) ← temporal

**fin para****fin**

Links recomendados para ver animación de método de Selección:

- <http://www.youtube.com/watch?v=pVmGMHA5cXk&NR=1>
- [http://sziami.cs.bme.hu/~gsala/alg\\_anims/3/ssort-e.html](http://sziami.cs.bme.hu/~gsala/alg_anims/3/ssort-e.html)
- <http://www.cs.hope.edu/~algaanim/animator/Animator.html>
- <http://www2.hig.no/~algmet/animate.html>

**Ejercicios:**EJERCICIOS 1 - ARREGLOS

1. Hacer un algoritmo que lea un arreglo de dimensión fija de enteros X , y otro entero C, y obtenga el número de veces que aparece C en X.
2. Hacer un algoritmo que lea un arreglo de dimensión fija de reales X, y calcule el promedio de los elementos de X.
3. Determinar la salida del siguiente programa si la entrada es 20, 60, 70, 10, 0, 40, 30, 90

Variables

entero i

entero j

entero k

entero A(8)

inicio

Para j en (0,7)

leer A(j)

fin para

i ← 0

j ← 1

Mientras j &lt; 8 Y A(j-1) &lt; A(j)

i ← i+1

j ← j+1

fin mientras

Para k en (0,i)

escribir A(k)

fin para

4. Desarrollar un algoritmo que lea un arreglo de dimensión fija X y un real C, y obtengan el número de elementos de X menores iguales que C.
5. Desarrollar un algoritmo que lea un arreglo de dimensión fija de reales X y otro real C, y cambie todos los valores de X que sean menores a C a 0. Ej:
 

```
Para
X={1, 4.1, 6.3, 2, 3.2, 8}
C = 3
el arreglo debe quedar:
X={0, 4.1, 6.3, 0, 3.2, 8}
```
6. Hacer un algoritmo que lea una cadena de caracteres, y luego escriba en pantalla la cantidad de veces que aparece cada letra (sin mostrar las que no aparecen). Ej:
 

```
Palabra ingresada: "conocido"
c : 2
d : 1
i : 1
n : 1
o : 3
```

### EJERCICIOS 2 - ORDENAMIENTO

1. Implementar el método de la burbuja para ordenar de forma ascendente un arreglo de 15 posiciones con 15 letras del alfabeto
2. Implementar los tres métodos básicos de ordenamiento (burbuja, selección e inserción) para ordenar los elementos de un arreglo de 15 posiciones, el cual tiene almacenado 15 números enteros, muestre la traza.
3. Cual de los tres métodos básicos es mas eficiente si los datos están previamente ordenados y cual si los datos están ordenados inversamente?.
4. Existen otros métodos de ordenamiento,? Nómbralos y explique en que consiste su funcionamiento.
5. Modificar el algoritmo del ejercicio 6 del apartado de ARREGLOS, para que muestre los resultados en orden decreciente por cantidad de apariciones. Ej:
 

```
Palabra ingresada: "conocido"
o : 3
c : 2
d : 1
i : 1
n : 1
```
6. Diseñar un algoritmo que al introducir una cantidad de dinero expresado en bolívares nos indique cuántos billetes y monedas se puede tener como mínimo, es decir que descomponga el billete, almacenando en un arreglo los resultados y los muestre de forma ordenada de menor a mayor, donde el índice del mismo será el valor del billete o moneda respectiva.
7. Dado un arreglo de N elementos, elabore un algoritmo que ordene el arreglo e imprima el arreglo original y el ordenado uno al lado del otro.
 Ejemplo:
 

Desordenado	Ordenado
20	10
40	20
90	40
10	50
50	80
80	90

8. Dada la información correspondiente a las edades de un grupo de personas. Elabore un algoritmo que determine la mayor edad y que ordene las edades a partir de la posición del mayor. Imprima los dos vectores el original y el modificado.  
Edad 15 **17** 16 13 12 10  
Edad modificado 15 10 12 13 16 17
9. Escribir un programa que permita ingresar N números enteros en un arreglo llamado "numeros", que calcule la suma total de los números en el arreglo y que despliegue el conjunto de números así como la suma de los mismos.

### EJERCICIO 3 - BUSQUEDA

1. Desarrollar un algoritmo que dado un arreglo de N posiciones, busque un en el mismo un elemento X dado por el usuario y que muestre en que posición del arreglo se encuentra.
2. Dados 3 números X,Y, Z, determinar la suma de ellos si cada uno se encuentra en el arreglo A(n).

Links recomendados:

- <http://www.dtic.upf.edu/~jonsson/pli09/Apuntes/TeoriaSemana1-2.ppt>
- <http://mate.uprh.edu/~jse/cursos/4097/prontuario.html>
- <http://www.efn.uncor.edu/dep/computacion/materias/algoritmos/>
- <http://www.uemc.edu/campusvuemc/catalogo/Asignatura.ASP?asignatura=411>
- <http://www.programacion.com/tutorial/introprog/11/>
- <http://www.uprb.edu/profesor/ntorres/sici-3012-modulo-recorrer-arreglo.doc>
- <http://www.scribd.com/doc/3207178/Arreglos>
- <http://www.youtube.com/watch?v=myKIT30n15Y>