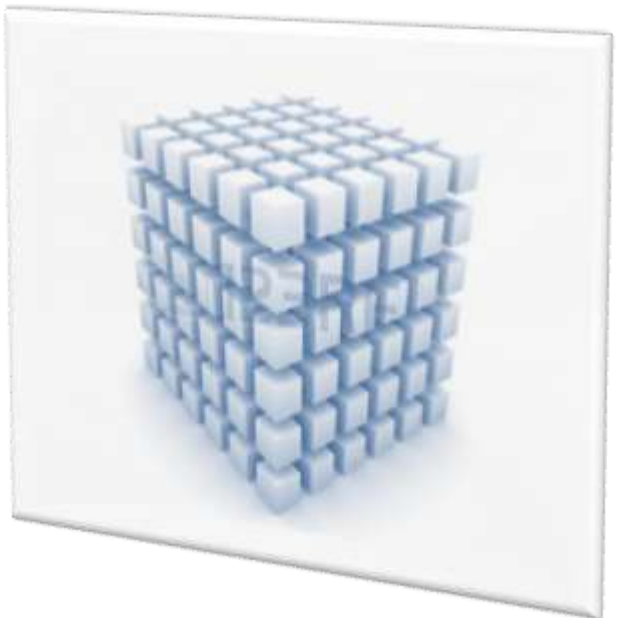
UNIDAD V: ARREGLOS



Arreglos

Un arreglo es un conjunto finito de elementos del mismo tipo cuyo acceso se realiza a través de índices. Los índices permiten identificar en forma individual a cada elemento del arreglo. En general, el valor del índice está entre *cero* y *N-1* (*N*: número de elementos del arreglo). Sin embargo, muchos lenguajes permiten utilizar índices negativos.

Desde el punto de vista del programa un arreglo se define como una zona de almacenamiento contiguo (posiciones de memoria consecutivas) que contiene una serie de elementos, todos del mismo tipo (enteros, reales, lógicos, etc.) que pueden ser procesados individualmente o en conjunto.

De acuerdo a como se organizan los elementos de los arreglos, estos se clasifican en:

- Arreglos Unidimensionales o Vectores
- Arreglos Bidimensionales o Matrices
- Arreglos Multidimensionales

Arreglos Unidimensionales

El tipo más simple de arreglo es el arreglo unidimensional o vector, cuyos elementos pueden ser accedidos a través de un índice. Por ejemplo, el siguiente es un vector de 6 elementos de tipo real cuyo nombre es *valores*.

2.30	5.90	1.44	3.11	7.05	0.32
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6

Para acceder a un elemento específico del arreglo valores, se indica el nombre del arreglo y el índice que especifica su posición. Por ejemplo para acceder al dato *1.44* se hace referencia a *valores[3]*.

¿Cómo se declara un vector?

En general, los vectores se declaran usando la siguiente sintaxis:

```
PROGRAMA ejemplo_vectores
CONSTANTES
        MAX_ELEMENTOS=10
TIPOS
        tipo_vector=ARREGLO [1..MAX_ELEMENTOS] de tipo_dato (entero, real, lógico, etc.)
VARIABLES
        nombre_variable: tipo_vector
```

Puede observarse que en la declaración de tipos se define el tipo y tamaño del arreglo, que luego será utilizado para declarar variables en el programa.

Índices de Vectores

Los índices permiten referenciar directamente posiciones específicas de un vector. Los índices pueden ser valores constantes ordinales (por ejemplo, *valores[5]*), variables ordinales (por ejemplo, *valores[i]*) o expresiones de tipo ordinal (por ejemplo, *valores[i+2]*)

El valor mínimo permitido para el índice de un vector se denomina límite inferior (*LI*) y el valor máximo permitido para el índice de un vector se denomina límite superior (*LS*). Por ejemplo, podría definirse un arreglo cuyos índices estuvieran entre 1 y 30, donde *LI=1* y *LS=30*.

El número de elementos de un vector se conoce como *Rango del Vector*. Este se determina mediante la siguiente expresión:

Rango de Vector=LS – LI +1

Por ejemplo, si el *vector A* tiene *LI=1* y *LS=15* entonces el *Rango del Vector* es 15-1+1=15 (cantidad de elementos).

Por ejemplo, si el *vector B* tiene *LI=-3* y *LS=7* entonces el *Rango del Vector* es 7-(-3)+1=11 (cantidad de elementos).

Arreglos Bidimensionales

Un arreglo bidimensional, matriz o tabla es un conjunto de elementos, todos del mismo tipo, que está organizada en filas y columnas por lo que requiere de 2 índices para identificar a cada posición. El primer índice especifica la fila y el segundo la columna. Por ejemplo, en la matriz letras de tipo carácter, el dato *"a"* se encuentra en la fila 2 columna 3, que se referencia como *letras[2,3]*.

	Columna 1	Columna 2	Columna 3
Fila 1	**f**	**w**	**u**
Fila 2	**l**	**z**	**a**
Fila 3	**r**	**m**	**v**

¿Cómo se declara una matriz?

En general, los matrices se declaran usando la siguiente sintaxis:

```
PROGRAMA ejemplo_matrices
CONSTANTES
        MAX_FILAS=12
        MAX_COLUMNAS=14
TIPOS
  tipo_matriz=ARREGLO [1..MAX_FILAS,1..MAX_COLUMNAS] de tipo_dato (entero, real, lógico, etc.)
VARIABLES
        nombre_variable: tipo_matriz
```

¿Cómo determinar la cantidad de elementos de una matriz?

Para determinar la cantidad de elementos de una matriz se emplea la siguiente expresión:

Rango de la matriz=(LS_F-LI_F+1)*(LS_C-LI_C+1)

donde

LS_F: límite superior del índice de filas
LI_F: límite inferior del índice de filas
LS_C: límite superior del índice de columnas

LI_C: límite inferior del índice de columnas

Por ejemplo, para determinar el rango de una matriz 3x5 la expresión 3-1+1)*(5-1+1) vale 15 (una matriz de 15 elementos o posiciones).

Si la matriz tiene igual número de filas que de columnas se dice que se trata de una matriz cuadrada. En las matrices cuadradas los elementos que ocupan las posiciones donde los índices de fila y columna son iguales reciben el nombre de diagonal principal. Por ejemplo, en la matriz letras los elementos de la diagonal principal son: f, z, v.

Arreglos Multidimensionales

Un arreglo puede definirse de tres, cuatro o hasta n dimensiones. De acuerdo a la cantidad de dimensiones será necesario especificar la cantidad de índices para acceder a los elementos individuales del arreglo. Por ejemplo, si el arreglo tiene 3 dimensiones, requiere de 3 índices; si el arreglo tiene 4 dimensiones, requiere de 4 índices; y si el arreglo tiene n dimensiones, entonces requiere de n índices.

Considerando un arreglo de 3 dimensiones de tipo entero denominado *calculo,* para acceder al elemento ubicado en la posición 2,4,6 se debe especificar: *calculo[2,4,6]*.

¿Cómo se declaran los arreglos multidimensionales?

En general, los arreglos de *n* dimensiones se declaran usando la siguiente sintaxis:

```
PROGRAMA ejemplo_multidimensional
CONSTANTES
        MAX_D1=12
        MAX_D2=4
        …
        MAX_DN=7
TIPOS
    tipo_multi=ARREGLO [1..MAX_D1,1..MAX_D2,. . . ,1..MAX_DN] de tipo_dato (entero, real, lógico, etc.)
VARIABLES
        nombre_variable: tipo_multi
```

¿Cómo determinar la cantidad de elementos de un arreglo multidimensional?

Para determinar la cantidad de elementos de un arreglo multidimensional se emplea la siguiente expresión:

Rango del arreglo multidimensional=$(LS_1-LI_1+1)*(LS_2-LI_2+1)*$ …$*(LS_n-LI_n+1)$

Por ejemplo, si consideramos el arreglo calculo 5x4x6, el rango del arreglo es (5-1+1)*(4-1+1)*(6-1+1) =120, es decir que el arreglo tiene 120 elementos.

Operaciones sobre Arreglos

Sobre los arreglos pueden realizarse las siguientes operaciones:

- Asignación
- Lectura/Escritura
- Recorrido (acceso secuencial)
- Actualización (agregar, insertar y borrar)
- Búsqueda
- Ordenación

Asignación

La asignación de valores a un elemento del arreglo se realizará con la instrucción de asignación:

A[27]←43 Se asigna el valor 43 al elemento 27 del arreglo A

Si se desea asignar valores a todos los elementos del arreglo, se debe utilizar estructuras repetitivas (repetir, mientras o para) e incluso selectivas (si, según). Por ejemplo, si se quiere inicializar en cero los 30 elementos del vector A, se puede utilizar la siguiente estructura:

```
…
PARA i DESDE 1 HASTA 30 HACER
  A[i]←0
FIN PARA
…
```

Lectura/Escritura

La lectura/escritura de datos en arreglos u operaciones de entrada/salida normalmente se realizan con estructuras repetitivas, aunque puede también hacerse con estructuras selectivas. Las instrucciones de lectura/escritura simple son:

LEER(A[4])	Asignar el valor ingresado por teclado a la posición 4 del arreglo A
ESCRIBIR(A[28])	Visualizar el contenido de la posición 28 del arreglo A

Recorrido (Acceso Secuencial)

Se puede acceder a los elementos de un vector para introducir datos en él (LEER) o bien para visualizar su contenido (*ESCRIBIR*). A la operación de efectuar una acción general sobre todos los elementos de un arreglo se la denomina recorrido del arreglo. Estas operaciones se realizan utilizando estructuras repetitivas, cuyas variables de control (por ejemplo, *i*) se emplean como índices del arreglo. El incremento de la variable utilizada como índice producirá el tratamiento sucesivo de los elementos del arreglo. Por ejemplo, el siguiente programa ilustra la carga y visualización de un arreglo unidimensional de enteros de 25 elementos.

```
PROGRAMA recorrido
CONSTANTES
    MAX=25
TIPOS
    vector=ARREGLO [1..MAX] de enteros
VARIABLES
    numeros:vector
    i:entero
INICIO
// Aquí se cargan los valores en el vector //
    PARA i DESDE 1 HASTA MAX HACER
        LEER números[i]
    FIN PARA
// Aquí se muestran los valores almacenados en el vector //
    PARA i DESDE 1 HASTA MAX HACER
        ESCRIBIR números[i]
    FIN PARA
FIN
```

Si se trata de una matriz, el recorrido se realiza utilizando los dos índices (fila y columna). Por ejemplo, el siguiente programa ilustra la carga y visualización de una matriz de 3x4 de reales.

```
PROGRAMA recorrido_2
CONSTANTES
    FILAS=3
    COLUMNAS=4
TIPOS
    matriz=ARREGLO [1..FILAS,1..COLUMNAS] de reales
VARIABLES
    datos:matriz
    i,j:entero
INICIO
// Aquí se cargan los valores en la matriz//
    PARA i DESDE 1 HASTA FILAS HACER
        PARA j DESDE 1 HASTA COLUMNAS HACER
            LEER datos[i,j]
        FIN PARA
    FIN PARA
```

```
// Aquí se muestran los valores almacenados en la matriz //
    PARA i DESDE 1 HASTA FILAS HACER
        PARA j DESDE 1 HASTA COLUMNAS HACER
            ESCRIBIR datos[i,j]
        FIN PARA
    FIN PARA
FIN
```

Actualización (Agregar, Insertar y Borrar)

La operación de actualizar implica 3 posibles acciones:

- Agregar datos al arreglo
- Insertar datos en el arreglo (en orden)
- Borrar datos del arreglo

Se denomina agregar datos a la operación de añadir un nuevo elemento al final del arreglo. La única condición necesaria para esta operación consistirá en la comprobación de espacio de memoria suficiente para el nuevo elemento, es decir, que el vector no tenga ocupadas todas sus posiciones. Por ejemplo, considerando el arreglo *números* de 6 posiciones (con 3 elementos cargados) se desea agregar el dato 21 que pasará a ocupar la cuarta posición.

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6
30	59	44			

Agregar 21

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6
30	59	44	21		

El procedimiento que permite agregar datos en un arreglo de enteros de tamaño MAX se presenta a continuación:

```
//El procedimiento Agregar_Vector introduce un elemento en el vector en la último posición disponible (agrega al
final) siempre y cuando el vector no esté completo.//
PROCEDIMIENTO agregar_vector (E/S num:vector, E/S ocupado:entero, E nuevo:entero)
INICIO
    SI ocupado=MAX ENTONCES
        ESCRIBIR 'VECTOR COMPLETO, NO PUEDE INGRESAR MAS DATOS')
    SINO
        ocupado←ocupado+1
        num[ocupado]←nuevo
     FINSI
FIN
```

Observe que se utiliza la variable *ocupado* para controlar la cantidad de elementos actualmente cargados en el vector. Por ello, esta variable se debe actualizar tras cada operación de agregar.

La operación de insertar un dato consiste en introducir el nuevo elemento en el interior del arreglo, en una posición específica (generalmente en orden). En este caso, se produce un desplazamiento de los elementos del arreglo que deben quedar por encima del insertado. Por ejemplo, considerando el arreglo números de 6 posiciones (con 4 elementos cargados en orden creciente) se desea insertar el dato 33 que deberá ocupar la tercera posición del arreglo por lo que los valores 47 y 56 se desplazarán 2 posiciones.

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6
14	31	47	56		

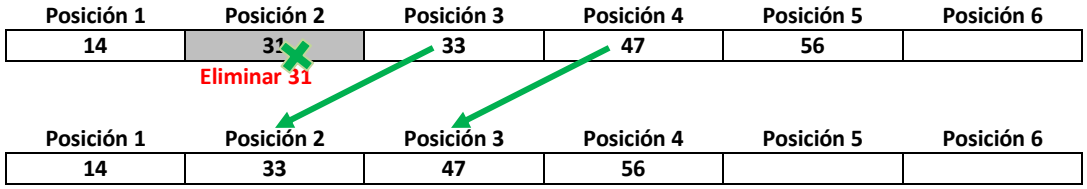
Insertar 33

Posición 1	Posición 2	Posición 3	Posición 4	Posición 5	Posición 6
14	31	33	47	56	

El procedimiento que permite insertar datos en un arreglo de enteros de tamaño MAX se presenta a continuación:

```
//El procedimiento Insertar_Vector introduce un elemento (en orden) en el vector, lo que supone el desplazamiento de
los valores mayores (salvo que se inserte al final) y la actualización de la variable ocupado.//
PROCEDIMIENTO insertar_vector (E/S num:vector, E/S ocupado:entero, E nuevo:entero)
VARIABLES
  i,j:entero
INICIO
   SI ocupado=MAX ENTONCES
        ESCRIBIR 'VECTOR COMPLETO, NO PUEDE INGRESAR MAS DATOS'
   SINO
        i←1
        MIENTRAS (i<=ocupado) Y (num[i]<nuevo) HACER
          i←i+1
        FIN MIENTRAS
        j←ocupado
        MIENTRAS j>=i HACER
           num[j+1]←num[j]
           j←j-1
         FIN MIENTRAS
        num[i]<-nuevo
        ocupado←ocupado+1
     FINSI
FIN
```

La operación de borrar un dato del final del arreglo no presenta ningún inconveniente, en cambio, borrar un dato del interior del arreglo implicará el desplazamiento de los elementos del arreglo que se encuentran por encima del eliminado. También se debe tener presente que no pueden eliminarse elementos de un arreglo vacío. Por ejemplo, considerando el arreglo números de 6 posiciones (que tiene 5 elementos cargados) se desea eliminar el elemento 31 que ocupa la segunda posición, lo que provocará que se desplacen los elementos 33, 47 y 56 ubicados por encima del elemento a eliminar.



El procedimiento que permite borrar datos en un arreglo de enteros de tamaño MAX se presenta a continuación:

```
//El procedimiento Borrar_Vector permite eliminar un elemento del vector (valor que se pasa como parámetro)
siempre que éste exista. La eliminación de un elemento implica la actualización de la variable ocupado.//
PROCEDIMIENTO borrar_vector(E/S num:vector, E/S ocupado:entero, E borrado:entero)
VARIABLES
  i:entero
  encontrado:lógico
INICIO
   SI ocupado=0 ENTONCES
        ESCRIBIR 'EL VECTOR NO CONTIENE ELEMENTOS'
   SINO
     encontrado←FALSO
     i←1
     MIENTRAS (i<=ocupado) Y NO encontrado HACER
          SI borrado=num[i] ENTONCES
            encontrado←VERDADERO
          SINO
            i←i+1
          FINSI
     FIN MIENTRAS
     SI encontrado=VERDADERO ENTONCES
        MIENTRAS i<ocupado HACER
          num[i] ←num[i+1]
          i←i+1
        FIN MIENTRAS
```

```
            ocupado←ocupado-1;
          SINO
            ESCRIBIR 'EL ELEMENTO NO EXISTE'
          FIN SI
        FIN SI
      FIN
```

Búsqueda, Ordenación e Intercalación

La *búsqueda*, la *ordenación* y la *intercalación* son operaciones básicas en programas de gestión de información. Estas operaciones permiten manipular los grandes volúmenes de datos almacenados en sistemas de archivos, bases de datos, etc. agilizando así el funcionamiento de los procesos de negocio.

La operación de *búsqueda* consiste en explorar una colección de datos para determinar la presencia de ciertos elementos, de acuerdo a algún criterio preestablecido.

La operación de *ordenación* permite generar a partir de una colección de datos otra colección cuya disposición se ajuste a un orden especificado.

La operación de *intercalación* permite combinar 2 colecciones de datos (de igual estructura) en una sola.

A continuación se presentan estas operaciones aplicadas a estructuras de tipo arreglo.

Búsqueda en Arreglos

Los métodos más usuales de búsqueda en arreglos son:

- Búsqueda Secuencial
- Búsqueda Binaria

Búsqueda Secuencial

El método más sencillo de búsqueda consiste en explorar secuencialmente (uno a uno) los elementos de un arreglo comparando cada uno con el criterio de búsqueda hasta que éste se encuentra o hasta que el arreglo se lee por completo (recorrido desde el primer al último elemento).

La búsqueda secuencial no requiere de ningún requisito y, por consiguiente, no se necesita que el arreglo esté ordenado. El recorrido del vector se realiza normalmente con estructuras repetitivas.

El algoritmo de *búsqueda secuencial* se presenta a continuación:

```
{El procedimiento Busqueda_Sec recorre el vector comparando cada posición con la variable buscado. Si el buscado
se encuentra en el vector se visualiza un mensaje que indica su posición, de lo contrario se notifica que el valor no
está presente.}
PROCEDIMIENTO busqueda_sec(E num:vector; E buscado:entero)
VARIABLES
  i:entero
  encontrado:lógico
INICIO
   encontrado←FALSO
   i←1
   MIENTRAS (i<=MAX) Y NO encontrado HACER
      SI buscado=num[i] ENTONCES
        encontrado<-VERDADERO
      SINO
        i←i+1
      FIN_si
   FIN_MIENTRAS
   SI encontrado=VERDADERO ENTONCES
     ESCRIBIR 'El valor buscado ocupa la posicion ',i,' del vector'
   SINO
     escribir 'Valor no encontrado'
   FIN_SI
FIN
```

Búsqueda Binaria

Si bien el método de búsqueda secuencial resulta sencillo, su aplicación a grandes volúmenes de datos ralentiza excesivamente el tiempo de procesamiento de un programa. Una alternativa más eficiente la constituye el método de búsqueda binaria. Este algoritmo parte de la premisa de un vector ordenado y utiliza el método "divide y vencerás" para encontrar el valor buscado. Básicamente, se examina primero el elemento central del arreglo, si éste es el buscado, entonces finaliza allí; de lo contrario se determina si el elemento buscado está en la primera o la segunda mitad del arreglo y a continuación se repite el proceso, utilizando el elemento central del subarreglo.

El algoritmo de *búsqueda binaria* se presenta a continuación:

```
{El procedimiento Busqueda_Bin debe aplicarse sobre un vector ORDENADO. La búsqueda
binaria consiste en dividir el vector original, sucesivamente, en vectores más pequeños
hasta encontrar el valor indicado (si es que éste existe en el vector). Puede observarse que
central permite determinar si la búsqueda prosigue por izquierda (valores menores que el
central) o por derecha (valores mayores que el central). En cada iteración del bucle
mientras se reduce el tamaño de la sublista (cuyos límites están definidos por alto y bajo)
de elementos analizados.}
PROCEDIMIENTO busqueda_bin(var num:vector;buscado:entero)
VARIABLES
  alto,bajo,central:entero
  encontrado:lógico
INICIO
   bajo←1
   alto←MAX
   encontrado←FALSO
   MIENTRAS NO encontrado Y (bajo<=alto) HACER
      central←(bajo+alto) div 2;
      SI buscado=num[central] ENTONCES
        encontrado←VERDADERO
      SINO
        SI buscado<num[central] ENTONCES
          alto←central-1
        SINO
          bajo←central+1
        FIN_SI
      FIN_SI
    FIN_MIENTRAS
   SI encontrado=VERDADERO ENTONCES
     ESCRIBIR 'El valor ocupa la posicion ',central,' del vector'
   SINO
     ESCRIBIR Valor no encontrado'
   FIN_SI
FIN
```

Ordenación de Arreglos

Los métodos más usuales de ordenación de arreglos son:

- Burbuja o Intercambio
- Selección
- Inserción
- Shell
- Rápido (Quicksort)

A continuación se describen los métodos estos métodos.

Método de Burbuja o Intercambio

El algoritmo de intercambio o burbuja se basa en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que estén todos ordenados. Por ejemplo, si se considera el siguiente vector

2.30	5.90	1.44	3.11	0.32
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5

Los pasos a dar son:

1. Comparar *elemento[1]* y *elemento[2]*; si están en orden, se mantienen como están; en caso contrario, se intercambian entre sí.
2. A continuación se comparan los elementos 2 y 3; de nuevo se intercambian si es necesario.
3. El proceso continúa hasta que cada elemento del vector ha sido comparado con sus adyacentes y se han realizado los intercambios necesarios.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación burbuja (en cada paso se destacan los valores que se van comparando), la última línea de la tabla presenta los valores del vector ordenados.

Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
2.30	**5.90**	1.44	3.11	0.32
2.30	**5.90**	**1.44**	3.11	0.32
2.30	1.44	**5.90**	**3.11**	0.32
2.30	1.44	3.11	**5.90**	**0.32**
2.30	**1.44**	3.11	0.32	5.90
1.44	**2.30**	**3.11**	0.32	5.90
1.44	2.30	**3.11**	**0.32**	5.90
1.44	2.30	0.32	**3.11**	**5.90**
1.44	**2.30**	0.32	3.11	5.90
1.44	**2.30**	**0.32**	3.11	5.90
1.44	0.32	**2.30**	**3.11**	5.90
1.44	0.32	2.30	**3.11**	**5.90**
1.44	**0.32**	2.30	3.11	5.90
0.32	**1.44**	**2.30**	3.11	5.90
0.32	1.44	**2.30**	**3.11**	5.90
0.32	1.44	2.30	**3.11**	**5.90**
0.32	**1.44**	**2.30**	**3.11**	**5.90**

El algoritmo de ordenación burbuja se presenta a continuación (el procedimiento cambio realiza el intercambio de valores entre los parámetros x, y que recibe):

```
PROCEDIMIENTO cambio(E/S x:entero, E/S y:entero)
VARIABLES
        aux:entero
INICIO
        aux←x
        x←y
        y←aux
FIN
PROCEDIMIENTO burbuja(E/S a:vector)
VARIABLES
  j:entero
  bandera:lógico
INICIO
   bandera←VERDADERO
   MIENTRAS bandera HACER
    bandera←FALSO
    PARA j DESDE 1 HASTA MAX-1 HASTA
     SI a[j]>a[j+1] ENTONCES
       cambio(a[j],a[j+1])
       bandera←VERDADERO
     FIN_SI
    FIN_PARA
   FIN_MIENTRAS
FIN
```

Método de Ordenación por Selección

El algoritmo de ordenación por selección se basa en buscar el elemento menor del arreglo y colocarlo en primera posición. Luego se busca el segundo elemento más pequeño y se coloca en la segunda posición, y así sucesivamente. Por ejemplo, si se considera el siguiente vector

2.30	**5.90**	**1.44**	**3.11**	**0.32**
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5

Los pasos sucesivos a dar son:

1. Seleccionar el elemento menor del arreglo de n elementos.
2. Intercambiar dicho elemento con el primero.
3. Repetir estas operaciones con los *n-1* elementos restantes, seleccionando el segundo elemento; continuar con los *n-2* elementos restantes hasta que sólo quede el mayor.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación por selección (en cada paso se destacan los valores que se van comparando), la última línea de la tabla presenta los valores del vector ordenados.

Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
2.30	**5.90**	1.44	3.11	0.32
2.30	5.90	**1.44**	3.11	0.32
1.44	5.90	2.30	**3.11**	0.32
1.44	5.90	2.30	3.11	**0.32**
0.32	**5.90**	**2.30**	3.11	1.44
0.32	**2.30**	5.90	**3.11**	1.44
0.32	**2.30**	5.90	3.11	**1.44**
0.32	1.44	**5.90**	**3.11**	2.30
0.32	1.44	**3.11**	5.90	**2.30**
0.32	1.44	**2.30**	**5.90**	**3.11**
0.32	**1.44**	**2.30**	**3.11**	**5.90**

El algoritmo de ordenación por selección se presenta a continuación (el procedimiento cambio realiza el intercambio de valores entre los parámetros x, y que recibe):

```
PROCEDIMIENTO seleccion(E/S a:vector)
VARIABLES
        i,j:entero
INICIO
    PARA i DESDE 1 HASTA MAX-1 HACER
      PARA j DESDE i HASTA MAX-1 HACER
          SI a[i]>a[j+1] ENTONCES
            cambio(a[i],a[j+1])
          FIN_SI
      FIN_PARA
    FIN_PARA
FIN
```

Método de Ordenación por Inserción

El algoritmo de ordenación por inserción ordena un vector insertando cada *elemento[i]* entre los *i-1* anteriores que ya están ordenados. Para realizar esto comienza a partir del segundo elemento, suponiendo que el primero ya está ordenado. Si los 2 primeros elementos están desordenados, los intercambia. Luego, toma el tercer elemento y busca su posición correcta con respecto a los dos primeros. En general, para el elemento I, busca su posición con respecto a los i-1 elementos anteriores y de ser necesario lo inserta adecuadamente.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación por inserción para el vector:

2.30	**5.90**	**1.44**	**3.11**	**0.32**
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5

i	J	aux	a[j]	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
				2.30	**5.90**	**1.44**	**3.11**	**0.32**
2	1	5.90	2.30	2.30	5.90	1.44	3.11	0.32
				2.30	**5.90**	**1.44**	**3.11**	**0.32**
3	2	1.44	5.90			5.90		
	1		2.30		2.30			
	0			1.44				
				1.44	**2.30**	**5.90**	**3.11**	**0.32**
4	3	3.11	5.90				5.90	
	2		2.30			3.11		
				1.44	**2.30**	**3.11**	**5.90**	**0.32**
5	4	0.32	5.90					5.90
	3		3.11				3.11	
	2		2.30			2.30		
	1		1.44		1.44			
	0			0.32				
				0.32	**1.44**	**2.30**	**3.11**	**5.90**

El algoritmo de ordenación por inserción se presenta a continuación:

```
PROCEDIMENTO insercion(E/S a:vector)
VARIABLES
   i,j,aux:entero
INICIO
   PARA i DESDE 2 HASTA MAX HACER
      aux←a[i]
      j←i-1
      MIENTRAS (j>=1) Y (a[j]>aux) HACER
          a[j+1]←a[j];
          j←j-1
      FIN_MIENTRAS
      a[j+1]←aux;
   FIN_PARA
FIN
```

Método de Ordenación Shell

El algoritmo de ordenación Shell (o también llamado de inserción con incrementos decrecientes) es una mejora del método de inserción. En este método se realizan comparaciones por saltos constantes (mayores a 1) con lo que se consigue una ordenación más rápida. Es decir, se aplica la ordenación por inserción a subgrupos de elementos que están separados por una distancia de *salto*.

De forma general, la ordenación Shell se basa en tomar como salto *N/2* (*N* elementos del arreglo) y luego el salto se va reduciendo a la mitad en cada repetición hasta que vale 1.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación Shell para el vector:

18	**11**	**27**	**13**	**9**	**4**	**16**
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7

Salto	i	j	aux	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7
3	4	1	13	**18**	11	27	**13**	9	4	16
		-3		13	11	27	18	9	4	16
	5	2	9	13	**11**	27	18	**9**	4	16
		-1		13	9	27	18	11	4	16
	6	3	4	13	9	**27**	18	11	**4**	16
		0		13	9	4	18	11	27	16
	7	4	16	13	9	4	**18**	11	27	**16**
		1		**13**	9	4	**16**	11	27	18
				13	9	4	16	11	27	18
1	2	1	9	**13**	**9**	4	16	11	27	18
		0		9	13	4	16	11	27	18
	3	2	4	9	**13**	**4**	16	11	27	18
		1		**9**	**4**	13	16	11	27	18

Salto	i	j	aux	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7
				4	9	13	16	11	27	18
	4	3	16	4	9	**13**	**16**	11	27	18
	5	4	11	4	9	13	**16**	**11**	27	18
		3		4	9	**13**	**11**	16	27	18
		2		4	9	11	13	16	27	18
	6	5	27	4	9	11	13	**16**	**27**	18
	7	6	18	4	9	11	13	16	**27**	**18**
		5		4	9	11	13	16	18	27
0	-	-	-	4	9	11	13	16	18	27

El algoritmo de ordenación Shell se presenta a continuación:

```
PROCEDIMIENTO shell(E/S a:vector)
VARIABLES
  i, j, aux, salto:entero
INICIO
  salto←MAX div 2
  MIENTRAS salto > 0 HACER
      PARA i DESDE (salto+1) HASTA MAX HACER
        aux←a[i]
        j←i-salto
        MIENTRAS j>=1 Y a[j]>aux HACER
          a[j+salto]←a[j]
          j←j-salto
        FIN_MIENTRAS
        a[j+salto]←aux
      FIN_PARA
      salto←salto div 2
  FIN_MIENTRAS
FIN
```

Método de Ordenación Rápido (Quicksort)

El algoritmo de ordenación Rápido permite ordenar una lista basándose en el hecho de que es más fácil ordenar 2 listas pequeñas que una lista grande. El método divide al arreglo en 2 sublistas, una con todos los valores menores o iguales a un cierto valor específico y otra con todos los valores mayores que ese valor. El valor elegido puede ser cualquier valor arbitrario del vector. A este valor se lo denomina pivote.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación Rápido para el vector:

18	**11**	**27**	**13**	**9**	**4**	**16**
Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7

Pivote	izq	der	i	J	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7
13	1	7	1	6	**18**	11	27	**13**	9	**4**	16
			3	5	4	11	**27**	**13**	**9**	18	16
			4	4	4	11	9	**13**	27	18	16
			5	3							
11/18	1/5	3/7	1/5	3/7	4	**11**	9	13	27	**18**	16
			2/6	/6	4	9	11	13	16	18	27
			3/7	2/5							
-	-	-	-	-	**4**	**9**	**11**	**13**	**16**	**18**	**27**

El algoritmo de ordenación Rápido se presenta a continuación:

```
PROCEDIMIENTO rapido(E/S a:vector,E izq:integer,E der:integer)
VARIABLES
  i,j,pivote:entero
INICIO
   i←izq
   j←der
   pivote←a[(izq+der) div 2]
   MIENTRAS i<= j HACER
     MIENTRAS a[i]<pivote HACER
        i←i+1;
     FIN_MIENTRAS
     MIENTRAS a[j]>pivote HACER
        j←j-1
     FIN_MIENTRAS
     SI i<= j ENTONCES
       cambio(a[i],a[j]);
       i←i+1;
       j←j-1;
     FIN_SI
   FIN_MIENTRAS
   SI izq < j ENTONCES
    rapido(a,izq,j)
   FIN_si
   SI i < der ENTONCES
    rapido(a,i,der)
   FIN_si
FIN
```

Intercalación de Arreglos

La intercalación es el proceso de mezclar (intercalar) dos **vectores ordenados** y producir un nuevo vector también **ordenado**. En general pueden presentarse 2 casos:

- los vectores a intercalar tienen igual longitud,
- los vectores a intercalar tienen longitudes diferentes

En ambos casos, el tamaño del vector de intercalación resulta de sumar las longitudes de los vectores a mezclar.

El algoritmo de intercalación se presenta a continuación (caso 1: vectores de igual longitud):

```
PROCEDIMIENTO intercalar (E uno:vector1;E dos:vector1;E/S tres:vector2);
VARIABLES
  i,j,k:entero
INICIO
   i←1;j←1;k←1;
   MIENTRAS (i<=MAX1) Y (j<=MAX1) HACER
      SI uno[i]<dos[j] ENTONCES
         tres[k] ←uno[i];
         k←k+1;
         i←i+1;
      SINO
         tres[k]←dos[j];
         k←k+1;
         j←j+1;
       FIN_SI
   FIN_MIENTRAS
   MIENTRAS i<=MAX1 HACER
      tres[k]←uno[i];
      k←k+1;
      i←i+1;
   FIN_MIENTRAS
   MIENTRAS j<=MAX1 HACER
      tres[k]←dos[j];
      k←k+1;
      j←j+1;
    FIN_MIENTRAS
FIN
```

En las siguientes tablas se muestra el comportamiento del algoritmo de intercalación, obsérvese que se han destacado en rojo y azul los elementos de los vector que se utilizan en cada paso para generar el vector de intercalación.

Vector 1	Vector 2	Vector 3
14	**2**	2
16	6	-
21	44	-
42	63	-
		-
		-
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	**6**	6
21	44	-
42	63	-
		-
		-
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	**44**	14
42	63	-
		-
		-
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	**44**	14
42	63	16
		-
		-
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	**44**	14
42	63	16
		21
		-
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	**44**	14
42	63	16
		21
		42
		-
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	**44**	14
42	63	16
		21
		42
		44
		-

Vector 1	Vector 2	Vector 3
14	2	2
16	6	6
21	44	14
42	**63**	16
		21
		42
		44
		63

Vector 1	Vector 2	Vector 3
14	2	**2**
16	6	**6**
21	44	**14**
42	63	**16**
		21
		42
		44
		63