

Apellido y Nombre:

Fecha:/...../.....

CONCEPTOS A TENER EN CUENTA**INTERCALACIÓN DE ARREGLOS**

La intercalación es el proceso que permite 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):

```

constantes
    MAX1=10
    MAX2=20
tipos
    tvector1=ARREGLO [1..MAX1] de caracteres
    tvector2=ARREGLO [1..MAX2] de caracteres
    . . .
procedimiento intercalar (E uno:tvector1, E ocup1: entero,
                        E dos:tvector1, E ocup2: entero,
                        E/S tres:tvector2, ocup3:entero)
variables
    i,j,k:entero
inicio
    i<-1
    j<-1
    k<-1
    mientras (i<=ocup1) Y (j<=ocup2) 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<=ocup1 hacer
        tres[k] <-uno[i]
        k<-k+1
        i<-i+1
    fin_mientras
    mientras j<=ocup2 hacer
        tres[k] <-dos[j]
        k<-k+1
        j<-j+1
    fin_mientras
    ocup3<-k-1
fin
  
```

En las siguientes tablas se muestra el comportamiento del algoritmo de intercalación. Como puede observarse, los elementos considerados en cada paso de la mezcla se han destacado con color.

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

ORDENACIÓN DE ARREGLOS

Una función habitual de los sistemas informáticos es la clasificación u ordenación de datos. En particular, para arreglos se han desarrollado una variedad de algoritmos de ordenación, siendo algunos de los métodos más conocidos:

- Burbuja o Intercambio
- Selección
- Inserción
- Shaker Sort
- 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

Se realizan los siguientes pasos:

1. Comparar *elemento[1]* y *elemento[2]* e intercambiarlos si no estuvieran en orden.
2. Comparar *elemento[2]* y *elemento[3]* e intercambiarlos si no estuvieran en orden.
3. Comparar *elemento[i]* y *elemento [i+1]* e intercambiarlos si no estuvieran en orden.
4. Comparados todos los elementos del arreglo, el proceso se reinicia y se repite hasta que el vector quede completamente ordenado (ya no se realizan intercambios).

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación burbuja. En cada paso se destacan los valores comparados. Además, en la última línea de la tabla se muestran los valores 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

Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
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	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 *Burbuja*, con criterio de ordenación creciente o ascendente, se presenta a continuación (el procedimiento cambio realiza el intercambio de valores):

PROCEDIMIENTO burbuja (E/S a: tvector, E ocup: ENTERO)

VARIABLES

j: ENTERO

bandera: LÓGICO

INICIO

ordenado ← FALSO

MIENTRAS ordenado = FALSO HACER

ordenado ← VERDADERO

PARA j DESDE 1 HASTA ocup-1 HACER

SI a[j] > a[j+1] ENTONCES

cambio(a[j], a[j+1])

bandera ← FALSO

FIN_SI

FIN_PARA

FIN_MIENTRAS

FIN

PROCEDIMIENTO cambio(E/S x: REAL, E/S y: REAL)

VARIABLES

aux: REAL

INICIO

aux ← x

x ← y

y ← aux

FIN

Método de Ordenación por Selección

El algoritmo de ordenación por *Selección* comienza buscando el menor elemento del arreglo para colocarlo en primera posición. Luego, busca el segundo elemento más pequeño y lo coloca en la segunda posición del vector, 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 realizar son:

1. Seleccionar el menor elemento del arreglo considerando n elementos.
2. Intercambiar el elemento encontrado con el que ocupa la primera posición del arreglo.
3. Seleccionar el menor elemento del arreglo considerando $n-1$ elementos
4. Intercambiar el elemento encontrado con el que ocupa la segunda posición del arreglo.
5. Los pasos anteriores deben repetirse para los $n-2$ elementos restantes.
6. Cuando se complete el recorrido del vector, los valores del vector quedarán ordenados.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación por *selección*. En cada paso se destacan, en color, los valores comparados. La última línea de la tabla presenta los valores 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 *Selección*, con criterio de ordenación creciente o ascendente, se presenta a continuación (el procedimiento cambio realiza el intercambio de valores):

```

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

```

Método de Ordenación por Inserción

El algoritmo de *Inserción* ordena un vector insertando cada *elemento[i]* entre los *i-1* elementos anteriores (ya ordenados). El método comienza comparando la segunda posición del arreglo con la primera, reubicando los valores de estas posiciones si fuera necesario. Luego, se toma el tercer elemento del arreglo y se busca la posición que éste debe ocupar respecto a los dos primeros. Esto puede derivar en el desplazamiento de datos para acomodar el valor a insertar. En general, para el elemento *i*, se busca la posición que le corresponde respecto a los *i-1* elementos anteriores, desplazándose datos si fuera necesario para insertarlo adecuadamente.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación por inserción para el vector mostrado a continuación. Como puede observarse el valor considerado en cada paso de ordenación se destaca en color naranja, mientras que los valores desplazados se resaltan en celeste:

				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:

```

PROCEDIMIENTO insercion(E/S a:tvector,E ocup:ENTERO)
VARIABLES
    i,j:ENTERO
    aux:REAL
INICIO
    PARA i DESDE 2 HASTA ocup 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 *Shaker Sort* o Sacudida

El algoritmo *shaker sort* o sacudida es una variante del método Burbuja que funciona de la siguiente forma:

1. Se recorre el arreglo de derecha a izquierda intercambiando pares de valores según el criterio de ordenación elegido.
2. Se recorre el arreglo de izquierda a derecha intercambiando pares de valores según el criterio de ordenación elegido.
3. Cada recorrido ignora las posiciones ya ordenadas lo que reduce las posiciones a comparar y ordenar
4. El algoritmo finaliza cuando todas las posiciones fueron ordenadas o cuando en un recorrido no se realizaron intercambios.

En la siguiente tabla se muestra el comportamiento del algoritmo de ordenación *Shaker Sort*. En cada paso se destacan, en color, los valores comparados (naranja y verde) y las posiciones ya ordenadas (gris). La última línea de la tabla presenta los valores del vector 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

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	0.32	3.11
2.30	5.90	0.32	1.44	3.11
2.30	0.32	5.90	1.44	3.11
0.32	2.30	5.90	1.44	3.11
0.32	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	5.90
0.32	2.30	1.44	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 sacudida se presenta a continuación (el procedimiento cambio realiza el intercambio de valores):

```

PROCEDIMIENTO sacudida (E/S a: tvector, E ocup: ENTERO)
VARIABLES
    ordenado: LÓGICO
    pri, ult, i: ENTERO
INICIO
    ordenado ← FALSO
    pri ← 1
    ult ← ocup
    MIENTRAS (ordenado = FALSO Y pri < ult) HACER
        ordenado ← VERDADERO
        PARA i DESDE ult HASTA pri+1 CON PASO -1 HACER
            SI (a[i] < a[i-1]) ENTONCES
                cambio(a[i], a[i-1])
                ordenado ← FALSO
            FIN_SI
        FIN_PARA
        pri ← pri+1
        SI (ordenado = FALSO) ENTONCES
            ordenado ← -VERDADERO
            PARA i DESDE pri HASTA ult-1 CON PASO 1 HACER
                SI (a[i] > a[i+1]) ENTONCES
                    cambio(a[i], a[i+1])
                    ordenado ← FALSO
                FIN_SI
            FIN_PARA
        FIN_SI
        ult ← ult-1
    FIN_MIENTRAS
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.

La ordenación Shell se basa en tomar como salto $N/2$ (N elementos del arreglo), valor que se reduce (a la mitad) en cada repetición hasta que alcanzar 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	K	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6	Elemento 7
3	4	1	4	18	11	27	13	9	4	16
		-2		13	11	27	18	9	4	16
	5	2	5	13	11	27	18	9	4	16
		-1		13	9	27	18	11	4	16
	6	3	6	13	9	27	18	11	4	16
		0		13	9	4	18	11	27	16
	7	4	7	13	9	4	18	11	27	16
		1	4	13	9	4	16	11	27	18
		0		13	9	4	16	11	27	18
1	2	1	2	13	9	4	16	11	27	18
		0		9	13	4	16	11	27	18
	3	2	3	9	13	4	16	11	27	18
		1	2	9	4	13	16	11	27	18
		0		4	9	13	16	11	27	18
	4	3	4	4	9	13	16	11	27	18
		0		4	9	13	16	11	27	18
	5	4	5	4	9	13	16	11	27	18
		3	4	4	9	13	11	16	27	18
		0		4	9	11	13	16	27	18
	6	5	6	4	9	11	13	16	27	18
		0		4	9	11	13	16	27	18
	7	6	7	4	9	11	13	16	27	18
		5	6	4	9	11	13	16	18	27
		0		4	9	11	13	16	18	27

El algoritmo de ordenación Shell se presenta a continuación (el procedimiento cambio realiza el intercambio de valores):

```

PROCEDIMIENTO shell(E/S a:tvector,E ocup:ENTERO)
VARIABLES
    i,j,k,salto:ENTERO
INICIO
    salto ← ocup div 2
    MIENTRAS salto > 0 HACER
        PARA i DESDE (salto+1) HASTA ocup HACER
            j ← i-salto
            MIENTRAS j > 0 HACER
                k ← j+salto
                SI a[j] <= a[k] ENTONCES
                    j ← 0
                SINO
                    cambio(a[j],a[k])
            FIN_SI
            j ← j-salto
        FIN_PARA
        salto ← salto div 2
    FIN_MIENTRAS
FIN

```

Método de Ordenación Rápido (Quicksort)

El algoritmo de ordenación Rápido se basa en el principio *divide y vencerás*. Aplicando este enfoque, el método divide al arreglo en 2 subarreglos, uno con todos los valores menores o iguales a un valor específico (un valor del arreglo denominado pivote) y otro con todos los valores mayores al pivote. A su vez, cada subarreglo vuelve a dividirse aplicando la misma estrategia y esto continúa hasta que los sucesivos subarreglos se reducen al vacío (ningún elemento).

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	1/6	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:tvector,E izq:ENTERO,E der:ENTERO)
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
  
```

EJERCICIOS A RESOLVER

- Dados los siguientes vectores, muestre gráficamente y paso a paso, cómo se genera el vector de intercalación. Además indique la definición de tipos de datos adecuada considerando que el primer arreglo tiene tamaño máximo 10, mientras que el segundo puede almacenar hasta 16 elementos.

1	5	6	12	16	23	36	55
---	---	---	----	----	----	----	----

9	18	21	22	27	29	33	35
---	----	----	----	----	----	----	----

- Dados los siguientes vectores, muestre gráficamente y paso a paso, cómo se genera el vector de intercalación. Además indique la definición de tipos de datos adecuada considerando que los arreglos pueden almacenar como máximo 20 elementos cada uno.

17	22	31	47	78
----	----	----	----	----

2	8	11	36	41	49	52	66
---	---	----	----	----	----	----	----

3. Dados los siguientes vectores, ¿cómo se modificará el algoritmo de Intercalación para mezclarlos correctamente? Indique la definición de tipos de datos adecuada considerando que el primer arreglo puede almacenar hasta 30 elementos mientras que el segundo tiene capacidad para 20.

76	54	46	32	26	13	6	5
----	----	----	----	----	----	---	---

89	78	41	31	27	17	13	9
----	----	----	----	----	----	----	---

4. Modifique el algoritmo de intercalación de modo el recorrido de los vectores se realice de forma inversa (desde las últimas posiciones hacia las primeras). Verifique el algoritmo modificado realizando la prueba de escritorio con los siguientes vectores.

8	15	26	31	36	43	46	75					
---	----	----	----	----	----	----	----	--	--	--	--	--

2	8	11	43	67	69									
---	---	----	----	----	----	--	--	--	--	--	--	--	--	--

5. Modifique el algoritmo de intercalación de modo que realice la mezcla de 2 vectores con criterios de ordenación opuestos. Suponga que el primer vector almacena datos en orden creciente mientras que el segundo guarda datos en orden decreciente. Verifique el algoritmo modificado realizando la prueba de escritorio con los siguientes vectores.

2	6	16	24	35	69	77	81						
---	---	----	----	----	----	----	----	--	--	--	--	--	--

79	66	45	39	37	15	9	7			
----	----	----	----	----	----	---	---	--	--	--

6. Realice la prueba de escritorio del algoritmo de ordenación *Burbuja* para el siguiente vector (utilice la tabla adjunta):

Bandera	j	j+1	$a[j] > a[j+1]$	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5	Elemento 6
				5	13	1	10	3	8
...

7. Dado los siguientes vectores, muestre gráficamente el comportamiento del algoritmo de ordenación *Burbuja*.

71	53	62	2	60	53	46	25
----	----	----	---	----	----	----	----

79	8	11	21	17	37	13	5
----	---	----	----	----	----	----	---

9	16	81	11	7	25	3	36
---	----	----	----	---	----	---	----

8. Modifique el algoritmo de ordenación *Burbuja* considerando lo siguiente (cada modificación se realiza por separado):

- el criterio de ordenación debe ser decreciente
- el criterio de ordenación será definido por un parámetro del algoritmo: (1) creciente, (2) decreciente
- los bucles del algoritmo deben implementarse con estructuras REPETIR
- las posiciones ya ordenadas no deben ser consideradas en los recorridos sucesivos del vector

9. Realice la prueba de escritorio del algoritmo de ordenación por *Selección* para el siguiente vector (utilice la tabla adjunta):

I	J	$a[i] > a[j+1]$	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
			70	50	30	34	10
...

10. Dado los siguientes vectores, muestre gráficamente el comportamiento del algoritmo de ordenación por *Selección*.

17	33	45	25	40	62	39	14
----	----	----	----	----	----	----	----

80	90	38	18	43	11	63	23
----	----	----	----	----	----	----	----

17	9	31	44	96	10	6	58
----	---	----	----	----	----	---	----

11. Modifique el algoritmo de ordenación por *Selección* considerando lo siguiente (cada modificación se realiza por separado):

- el criterio de ordenación debe ser decreciente
- el criterio de ordenación será definido por un parámetro del algoritmo: (1) creciente, (2) decreciente
- los bucles del algoritmo deben implementarse con estructuras MIENTRAS
- sabiendo que el algoritmo, en cada pasada (recorrido), intercambia el contenido de la posición comparada con cada valor que cumpla la condición de ordenación, modifíquelo de manera que por recorrido sólo se realice un intercambio (correspondiente al valor mínimo o máximo valor detectado para esa pasada)

12. Realice la prueba de escritorio del algoritmo de ordenación por *Inserción* para el siguiente vector (utilice la tabla adjunta):

I	aux	J	$j \geq 1 \text{ Y } a[j] > \text{aux}$	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
				35	88	44	50	24
...

13. Dado los siguientes vectores, muestre gráficamente el comportamiento del algoritmo de ordenación por *Inserción*.

54	76	58	98	20	60	17	13
----	----	----	----	----	----	----	----

31	70	45	65	67	35	18	33
----	----	----	----	----	----	----	----

54	14	89	23	12	76	11	30
----	----	----	----	----	----	----	----

14. Modifique el algoritmo de ordenación por *Inserción* considerando lo siguiente (cada modificación se realiza por separado):

- el criterio de ordenación debe ser decreciente
- el criterio de ordenación será definido por un parámetro del algoritmo: (1) creciente, (2) decreciente
- el recorrido de ordenación del vector se realizará desde las últimas posiciones hacia las primeras (inverso al recorrido normal). Tenga en cuenta que "últimas posiciones" se refiere a datos válidos del arreglo.
- los bucles del algoritmo deben implementarse con estructuras REPETIR.

15. Realice la prueba de escritorio del algoritmo de ordenación Sacudida o *Shaker Sort* para el siguiente vector (utilice la tabla adjunta):

ordenado	pri	ult	i	$a[i] > a[i+1]$	$a[i] < a[i-1]$	Elemento 1	Elemento 2	Elemento 3	Elemento 4	Elemento 5
						35	33	30	34	43
...

16. Dado los siguientes vectores, muestre gráficamente el comportamiento del algoritmo de ordenación *Shaker Sort*.

7	3	6	2	10	5	4	12
---	---	---	---	----	---	---	----

9	8	31	11	14	17	28	15
---	---	----	----	----	----	----	----

56	13	87	36	41	76	15	22
----	----	----	----	----	----	----	----

17. Modifique el algoritmo de ordenación *Shaker Sort* considerando lo siguiente (cada modificación se realiza por separado):

- el criterio de ordenación debe ser decreciente
- los bucles del algoritmo deben implementarse con estructuras REPETIR

18. Codifique los algoritmos *Shell* y *Rápido* vistos en teoría y compruebe su funcionamiento con vectores de 100 o más elementos.

