2. CLASIFICACIÓN

2.1 CLASIFICACIÓN es el proceso de rearreglar un conjunto de objetos en un orden específico. Su finalidad es facilitar la búsqueda posterior.

Un método de clasificación se llama **estable** si el orden relativo de elementos con iguales llaves permanece inalterado en el proceso de clasificación

2.2 CLASIFICACIÓN DE ARREGLOS

2.2.1 POR INSERCIÓN DIRECTA

- Método con centinela: consiste en recorrer el arreglo, en cada paso, comenzando con la posición i = 2 e ir aumentando la posición *i* en una unidad, el i-ésimo elemento de la secuencia desordenada se toma y se transfiere a la secuencia destino (ya ordenada), insertándolo en el lugar correspondiente. Ejem:

- inicio: se clasifica de menor a mayor, la posición ***0 es el centinela***

  cadena= ***??*** 12 42 44 55 94 18 6 67

  posicion= *0* 1 2 3 4 5 6 7 8

- recorrido del elemento 18: la cadena está dividida en dos partes cuyo eje es el elemento 18 (negrita), a su izquierda los elementos ordenados (subrayados) y el centinela, a su derecha los desordenados (incluido el 18).

  cadena= *??* 12 42 44 55 94 **18** 6 67 ElementoInsertado=18

  - se carga el elemento (18) en el centinela

    cadena= ***18*** 12 42 44 55 94 **18** 6 67 ElementoInsertado=18

  - se va comparando el elemento (18) con el anterior hasta encontrar su lugar correspondiente en orden:

    cadena= *18* 12 42 44 55 **94** **18** 6 67 .

    ¿ 94 < 18 ? NO, intercambia valores

    cadena= *18* 12 42 44 **55** **18** 94 6 67 .

    ¿ 55 < 18 ? NO, intercambia valores

    cadena= *18* 12 42 **44** **18** 55 94 6 67 .

    ¿ 44 < 18 ? NO, intercambia valores

    cadena= *18* 12 **42** **18** 44 55 94 6 67 .

    ¿ 42 < 18 ? NO, intercambia valores

    cadena= *18* **12** **18** 42 44 55 94 6 67 .

    ¿ 12 < 18 ? SÍ, sigue recorriendo el vector con la llave 6.....

- Programa:

```
MODULE ClasificacionDirecta;
   FROM InOut IMPORT WriteLn,WriteString,Read,WriteCard,Write;
CONST N=8; (* longitud de la cadena *)
      TYPE vector=ARRAY [0..N] OF CARDINAL;
      (* vector[0]= centinela *)
VAR    cadena:vector; (* cadena para la b£squeda *)
      i,j,f,ElementoInsertado:CARDINAL;
BEGIN
(* inicio *)
cadena[1]:= 44;cadena[2]:= 55;cadena[3]:= 12;cadena[4]:= 42;cadena[5]:= 94;
cadena[6]:= 18;cadena[7]:= 06;cadena[8]:= 67;
(* clasificaci¢n directa *)
FOR i:= 2 TO N DO
      ElementoInsertado:= cadena[i];
      cadena[0]:=ElementoInsertado;j:=i;
      WHILE ElementoInsertado < cadena[j-1] DO
            cadena[j]:=cadena[j-1];
            j:=j-1;
      END;
      cadena[j]:= ElementoInsertado;
END;
```

```
END ClasificacionDirecta._
```


- **Análisis**: caso mínimo de comparaciones: si están todos ordenados será 3*(n-1)
caso promedio de comparaciones: $(n^2 +11n -12)/4$
caso máximo de comparaciones: si están en orden inverso al ordenado será de $(n^2 +5n -6)/3$.

POR INSERCIÓN BINARIA

- **Método**: trata de insertar el i-ésimo (negrita) elemento en un subarreglo $a_j..a_{i-1}$ ya ordenado mediante el método similar a la búsqueda binaria. Ejem:

- inicio, el nuevo elemento 2 se quiere insertar en la secuencia 0...9 ya ordenada (subrayados):

```
cadena =0346789 2 51 ElementoInsertado=8
pos i^ =          i
pos L^ =L
pos m^ =     m
pos R^ =          R
```

- se va comparando el elemento apuntado por *m* hasta cumplir la condición de ordenación:

```
cadena =0346789 2 51 ElementoInsertado=8
pos i^ =          i
pos L^ =L
pos m^ = m
pos R^ =     R

cadena =0346789 2 51 ElementoInsertado=8
pos i^   =          i
pos L^  =L
pos m^ =m
pos R^ =   R
```

- cuando el valor apuntado por *m* cumple la condición, lo inserta en esta posición (negrita) y desplaza los valores (cursiva) del subarreglo ordenado:

```
cadena =02345678951
```

- continúa con la siguiente llave (5).....

- **Programa**:

```
MODULE ClasificacionBinaria;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read;
CONST N=10; (* longitud de la cadena *)
   TYPE tira= ARRAY [0..N-1] OF CHAR;
(* cadena para la b£squeda y la SUBselecci¢n *)
VAR cadena:tira;
VAR ElementoInsertado:CHAR;
VAR L,R,m,i,j:CARDINAL;
BEGIN
(* inicio *)
cadena:="3746908251";
ElementoInsertado:="0";
(* INSERCIàN binaria *)
FOR i:= 1 TO N-1 DO
      ElementoInsertado:= cadena[i];L:= 0; R:=i;
      WHILE L < R DO
            m:= (L+R) DIV 2;
            IF cadena[m] <= ElementoInsertado THEN
                  L:= m+1;
            ELSE
                  R:= m
            END;
```

```
        END;
        FOR j:= i TO R+1 BY -1 DO
            cadena[j]:= cadena[j-1];
        END;
        cadena[R]:=ElementoInsertado;
    END;
END ClasificacionBinaria.
```


- **Análisis**: comportamiento NO natural de un algoritmo de clasificación porque,

    - caso mínimo de comparaciones, si están en orden inverso al ordenado

    - caso promedio de comparaciones: n * (log n - log e ± 0.5)

    - caso máximo de comparaciones, si están ya ordenados. Tarda más que algoritmo de inserción directa

    Otro coste añadido es la inserción, puesto que al insertarlo habrá que mover todo la información individualmente y esto se encarece en distancias largas

2.2.2 POR SELECIÓN DIRECTA

- **Método**: considera todos los elementos del subarreglo fuente (desordenados) para encontrar el que tenga la llave más pequeña que debe depositarse como siguiente elemento en la secuencia destino (ordenado). Ejem:

    - inicio: funciona con tres índices, elementos ordenados *1..i-1* (subrayados), *j* para buscar elementos o llaves (negrita) y comparar con el índice *k* que "memoriza" el elemento (cursiva) más pequeños de *i+1..j*

```
cadena= 44 55 12 42 94 18  6 67  ElementoInsertado=12
pos i^=    iii
pos j^=       jjj
pos k^= kkk
```

    - va recorriendo el arreglo y "memorizando el elemento más pequeño:

```
cadena= 44 55 12 42 94 18  6 67
pos i^=    iii
pos j^=          jjj
pos k^=        kkk

cadena= 44 55 12 42 94 18  6 67
pos i^=   iii
pos j^=             jjj
pos k^=        kkk

cadena= 44 55 12 42 94 18  6 67
pos i^=    iii
pos j^=                jjj
pos k^=        kkk

cadena= 44 55 12 42 94 18  6 67
pos i^=    iii
pos j^=                   jjj
pos k^=        kkk

cadena= 44 55 12 42 94 18  6 67
pos i^=    iii
pos j^=                      jjj
pos k^=                   kkk

cadena= 44 55 12 42 94 18  6 67
pos i^=    iii
pos j^=                         jjj
pos k^=                   kkk
```

- una vez localizado el elemento inferior (6) se produce un intercambio entre las posiciones que indican los índices *k* (44) e *i* (6)

```
cadena=  6 55 12 42 94 18 44 67
pos i^=     iii
pos j^=        jjj
pos k^=     kkk
```

- continúa localizando los menores del arreglo a partir de la siguiente posición "*i*" (55)....

- **Programa**:

```
MODULE ClasificacionSeleccionDirecta;
   FROM InOut IMPORT WriteLn,WriteString,Read,WriteCard,Write;
CONST N=8; (* longitud de la cadena *)
      TYPE vector=ARRAY [1..N] OF CARDINAL;
VAR    cadena:vector; (* cadena para la b£squeda *)
      i,j,k,ElementoInsertado:CARDINAL;
BEGIN
(* inicio *)
cadena[1]:= 44;cadena[2]:= 55;cadena[3]:= 12;cadena[4]:= 42;cadena[5]:= 94;
cadena[6]:= 18;cadena[7]:= 06;cadena[8]:= 67;
(* clasificaci¢n directa *)
FOR i:= 1 TO N-1 DO
   k:= i; ElementoInsertado:=cadena[i];
   FOR j:= i+1 TO N DO
      IF cadena[j]<ElementoInsertado THEN
         k:=j; ElementoInsertado:= cadena[k];
      END;
   END;
   cadena[k]:=cadena[i]; cadena[i]:= ElementoInsertado;
END;
END ClasificacionSeleccionDirecta.
```

- **Análisis**: el número de comparaciones es independiente del orden inicial de las llaves o elementos (a diferencia de los algoritmos anteriores).

El nº de movimientos:

- mínimo es 3 * (n-1) en llaves iniciales ordenadas
- máximo es $n^2/4 + 3 * (n-1)$ en llaves iniciales en orden inverso
- promedio es n * ((ln n) + g) ; g= 0.577216... cte Euler

En consecuencia, este algoritmo ha de preferirse a la selección directa, pese a que ésta última es todavía un poco más rápida en los casos en que las llaves se clasifican o casi clasificadas al principio

2.2.3 POR INTERCAMBIO DIRECTO (BURBÚJA)

- **Método**, compara e intercambia (negrita) pares de elementos contiguos hasta clasificar todos los elementos. Hace varios pases sobre el arreglo, y en cada recorrido desplaza el elemento más pequeño (cursiva) del conjunto restante hacia el extremo final del subarreglo ya ordenado (subrayado). Ejem:

- ¿18 < 67?; sí, entonces no intercambia

```
cadena   = 6 44 55 12 42 94 18 67
pos i^   =        iii
pos j^   =              jjj
pos (j^-1)=          j-1
```

- ¿94 < 18?; NO, entonces intercambia

```
cadena   = 6 44 55 12 42 94 18 67
pos i^   =        iii
pos j^   =              jjj
pos (j^-1)=          j-1
```

- ¿42 < 18?; NO, entonces intercambia

```
cadena   = 6 44 55 12 42 18 98 67
pos i^   =         iii
pos j^   =                jjj
pos (j^-1)=             j-1
```

- ¿12 < 18?; si, entonces NO intercambia

```
cadena   = 6 44 55 12 18 42 98 67
pos i^   =         iii
pos j^   =             jjj
pos (j^-1)=          j-1
```

- ¿55 < 12?; NO, entonces intercambia

```
cadena   = 6 44 55 12 18 42 98 67
pos i^   =         iii
pos j^   =           jjj
pos (j^-1)=        j-1
```


- ¿44 < 12?; NO, entonces intercambia

```
cadena   = 6 44 12 55 18 42 98 67
pos i^   =         iii
pos j^   =         jjj
pos (j^-1)=    j-1
```

- ¿44 < 12?; NO, entonces intercambia

```
cadena   = 6 12 44  55 18 42 98 67
pos i^   =         iii
pos j^   =       jjj
pos (j^-1)= j-1
```

- y continúa....

- **Programa**:

```
MODULE ClasificacionBurbuja;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read;
CONST N=8; (* longitud de la cadena *)
   TYPE tira= ARRAY [0..N-1] OF INTEGER;
(* cadena para la b£squeda y la SUBselecci¢n *)
VAR cadena:tira;
VAR ElementoInsertado:INTEGER;
VAR i,j,Scroll:CARDINAL;
BEGIN
(* inicio *)
cadena[0]:= 44;cadena[1]:= 55;cadena[2]:= 12;cadena[3]:= 42;cadena[4]:= 94;
cadena[5]:= 18;cadena[6]:= 06;cadena[7]:= 67;
FOR i:= 1 TO N-1 DO
     FOR j:= N-1 TO i BY -1 DO
          IF cadena[j-1] > cadena[j] THEN
               ElementoInsertado:= cadena[j-1];
               cadena[j-1]:=cadena[j];
               cadena[j]:=ElementoInsertado;
          END;
     END;
END;
END ClasificacionBurbuja.
```

- **Análisis**: el orden es n^2 es poco eficaz

VIBRACIÓN ("Mejoramiento del Burbuja")

- **Método**, mejora el algoritmo anterior (burbuja) pues hace un doble pase (de derecha a izquierda y de izquierda a derecha) por el arreglo "memorizando" (índices L y R) la posición de los últimos intercambios, **el mejoramiento consiste en comparar las posiciones de intercambio para comprobar si el arreglo ya está ordenado, si es así acabará .** Ejem:

- inicio:

```
cadena   = 44 55 12 42 94 18  6 67
pos L^   =   LLL
```

pos R^ = RRR

- doble pasada:
  - fin primera parte (como el burbuja):

cadena = 6 **44** 55 12 42 94 18 **67**
pos L^ = LLL
pos R^ = RRR

  - fin segunda parte (como el burbuja pero al revés):

cadena = 6 44 **12** 42 55 18 67 **94**
pos L^ = LLL
pos R^ = RRR

- ¿arreglo ordenado?, NO porque L < R

cadena = 6 44 12 42 55 18 67 94
pos L^ = **LLL**
pos R^ = **RRR**

- va haciendo dobles pasadas hasta encontrar:

cadena = 6 12 18 42 44 55 67 94
pos L^ = **LLL**
pos R^ = **RRR**

- ¿arreglo ordenado?, SÍ porque L < R

- **Programa**:

```
MODULE ClasificacionVibracion;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read;
CONST N=8; (* longitud de la cadena *)
   TYPE tira= ARRAY [0..N-1] OF INTEGER;
(* cadena para la b£squeda y la SUBselecci¢n *)
VAR cadena:tira;
VAR ElementoInsertado:INTEGER;
VAR L,R,j,k:CARDINAL;
BEGIN
(* inicio *)
cadena[0]:= 44;cadena[1]:= 55;cadena[2]:= 12;cadena[3]:= 42;cadena[4]:= 94;
cadena[5]:= 18;cadena[6]:= 06;cadena[7]:= 67;
L:= 1; R:=N-1; k:=N-1;
REPEAT
     FOR j:= R TO L BY -1 DO
          IF cadena[j-1] > cadena[j] THEN
               ElementoInsertado:= cadena[j-1];
               cadena[j-1]:=cadena[j];
               cadena[j]:=ElementoInsertado;
               k:=j;
          END;
     END;
     L:=k+1;
     FOR j:= L TO R BY +1 DO
          IF cadena[j-1] > cadena[j] THEN
               ElementoInsertado:= cadena[j-1];
               cadena[j-1]:=cadena[j];
               cadena[j]:=ElementoInsertado;
               k:=j;
          END;
     END;
     R:=k-1;
UNTIL L > R;
END ClasificacionVibracion.
```

- **Análisis**: el orden es n^2 es poco eficaz, pero mejora su rendimiento si el arreglo está casi en orden

2.3 MÉTODOS DE CLASIFICACIÓN AVANZADOS

2.3.1 INSERCIÓN POR INCREMENTO DECRECIENTE ó SHELL

- **Método** es un refinamiento de la inserción directa consiste, en transferir a la secuencia ordenada (subrayado) un elemento (negrita) en el lugar correspondiente según un orden preestablecido y usa un centinela (cursiva) en cada búsqueda pero situado estratégicamente. Ejemplo:

- inicio (****** = posición del centinela):

** ****** ** ** 44 55 12 42 94 18 06 67

- **clasificación-4 (k = 4)**, similar a la inserción directa: la subcadena ordenada a la izquierda, un centinela y un elemento a transferir a la subcadena ordenada, pero los elementos **distan 4 posiciones**:

- cadena ordenada [44], elemento a insertar (94) y centinela (94):

94 ** ** ** 44 55 12 42 **94** 18 06 67

cadena resultante ordenada [44,94], continúa con otra

- cadena ordenada [55], elemento a insertar (18) y centinela (18):

** *18* ** ** 44 55 12 42 94 **18** 06 67

¿55 < 18?. NO, intercambia valores de la cadena ordenada hasta encontrar su lugar correspondiente en orden:

** *18* ** ** 44 55 12 42 94 **55** 06 67.

** **>>** ** ** 44 **18** 12 42 94 55 06 67.

cadena resultante ordenada [44,94], continúa con otra

- cadena ordenada [12], elemento a insertar (06) y centinela (06):

** ** *06* ** 44 18 12 42 94 55 **06** 67.

¿12 < 06? NO, intercambia valores de la cadena ordenada hasta encontrar su lugar correspondiente en orden:

** ** ***06*** ** 44 18 12 42 94 55 12 67

** ** >> ** 44 18 **06** 42 94 55 12 67.

cadena resultante ordenada [06,12], continúa con otra

- cadena ordenada [42], elemento a insertar (67) y centinela (67):

** ** ** *67* 44 18 06 42 94 55 12 **67**

¿42 < 67?, Sí, cadena ordenada [42,67], continúa con otra

- NO hay más elementos a la derecha, entonces siguiente clasificación

Resumen de la clasificación - 4:

subcadenas ordenadas [44,94],[18,55], [06,12], [42,67]

- **clasificación-2 (k = 2),** similar a la inserción directa: la subcadena ordenada a la izquierda, un centinela y un elemento a transferir a la subcadena ordenada, pero los elementos **distan 2 posiciones:**

- cadena ordenada [44], elemento a insertar (06) y centinela (06):

** ** *06* ** 44 18 **06** 42 94 55 12 67

¿44 < 06? NO, intercambia valores de la cadena ordenada hasta encontrar su lugar correspondiente en orden:

** ** ***06*** ** 44 18 44 42 94 55 12 67

** ** **>>** ** **06** 18 44 42 94 55 12 67

cadena resultante ordenada [06,44], continúa con otra

- cadena ordenada [18], elemento a insertar (42) y centinela (42):

** ** ** *42* 06 18 44 **42** 94 55 12 67

cadena resultante ordenada [18,42], continúa con otra

- cadena ordenada [06,44], elemento a insertar (94) y centinela (94):

** ** *94* ** 06 18 44 42 94 55 12 67

cadena resultante ordenada [06,44,94], continúa con otra

- cadena ordenada [18,42], elemento a insertar (55) y centinela (55):

** ** ***55*** 06 18 44 42 94 **55** 12 67

cadena resultante ordenada [18,42,55], continúa con otra

- cadena ordenada [06,44,94], elemento a insertar (12)
y centinela (12):

** ** *12* ** 06 18 44 42 94 55 **12** 67

¿94 < 12? NO, intercambia valores de la cadena ordenada
hasta encontrar su lugar correspondiente en orden:

** ** ***12*** ** 06 18 44 42 94 55 94 67.

** ** ***12*** ** 06 18 44 42 44 55 94 67.

** ** *12* ** 06 18 **12** 42 44 55 94 67.

cadena resultante ordenada [06,12,44,94], continúa con otra

- cadena ordenada [18,42,44,55] elemento a insertar (67) y centi. (67):

** ** ** *67* 06 18 12 42 44 55 94 **67**

cadena resultante ordenada [18,42,44,55,67], continúa con otra

Resumen de la clasificación - 2:

subcadenas ordenadas [06,12,44,94], [18,42,55,67]

- **clasificación-1 (k = 1), idéntico** a la inserción directa pues ahora la distancia entre elementos **es 1:**

- cadena ordenada [06], elemento a insertar (18) y centinela (18):

** ** ** *18* 06 **18** 12 42 44 55 94 67

cadena resultante ordenada [06,18], continúa con otra

- cadena ordenada [06,18], elemento a insertar (12) y centinela (12):

** ** ** *12* *06 18* ***12*** 42 44 55 94 67

¿18 < 12? NO, intercambia valores de la cadena ordenada
hasta encontrar su lugar correspondiente en orden:

** ** ** ***12*** 06 18 18 42 44 55 94 67.

** ** ** ***12*** 06 12 18 42 44 55 94 67.

- cadena ordenada [06,12,18], elemento a insertar (42)
y centinela (42):

** ** ** *42* 06 12 18 **42** 44 55 94 67

- cadena ordenada [06,12,18,42], elemento a insertar (44)
y centinela (44):

** ** ** *44* 06 12 18 42 44 55 94 67

cadena resultante ordenada [06,18,18,42,44], continúa con otra

- cadena ordenada [06,12,18,42,44], elemento a insertar (55)
y centinela (55):

** ** ** *55* 06 12 18 42 44 **55** 94 67

cadena resultante ordenada [06,18,18,42,44,55], continúa con otra

- cadena ordenada [06,12,18,42,44,55], elemento a insertar (94)

y centinela (94):

** ** ** *94* 06 12 18 42 44 55 **94** 67

cadena resultante ordenada [06,18,18,42,44,55,94]continúa con otra

- cadena ordenada [06,12,18,42,44,55,94], elemento a insertar (97) y centinela (97):

** ** ** *67* 06 12 18 42 44 55 94 **67**

¿67 < 94? NO, intercambia valores de la cadena ordenada hasta encontrar su lugar correspondiente en orden:

** ** ** ***67*** 06 12 18 42 44 55 94 94

** ** ** >> 06 12 18 42 44 55 **67** 94

Resumen de la clasificación - 1:

cadena ordenada[06,12,18,42,44,55,67,97]

- fin: ** ** ** ** **06 12 18 42 44 55 67 94.**

- Programa

```
MODULE ClasificacionIncDecreciente;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read,WriteInt;
CONST N=8; (* longitud de la cadena *)
CONST t=4; (* longitud de la cadena *)
   TYPE tipoM         = [1..t];
   TYPE tipoH         = ARRAY tipoM OF INTEGER;
   TYPE tipoAindice    = [-9..N];
   TYPE tipoA         = ARRAY tipoAindice  OF INTEGER;(* -9 viene de h[1]:=9 *)
   VAR a:tipoA;
      m:tipoM;
     h:tipoH;
      i,j,k,s,x:INTEGER;
BEGIN
(* inicio *)
a[1]:= 44;a[2]:= 55;a[3]:= 12;a[4]:= 42;a[5]:= 94; a[6]:= 18;a[7]:= 06;a[8]:= 67;
h[1]:= 8; h[2]:=4; h[3]:=2; h[4]:=1;
```



```
FOR m:= 1 TO t DO
   k:= h[m]; s:=-1 * k;
   FOR i:=k+1 TO N DO
      x:= a[i];
      j:= i-k;
      IF s = 0 THEN s:=-k END;
      s:= s + 1;
      a[s]:= x;(* centinela *)
WHILE x < a[j] DO
         a[j+k]:= a[j];
         j:=j-k;
      END;
      a[j+k]:= x;
   END;
END;
END ClasificacionIncDecreciente.
```

- Análisis: plantea algunos problemas matemáticos no resueltos porque no se sabe la elección de incrementos con resultados más eficientes, aunque se recomiendan las secuencias: 1,4,13,40,121,.... ó 1,3,7,15,31,... (colocadas en orden inverso); produciendo un rendimiento próximo a $n^{1.2}$

2.3.2 CLASIFICACIÓN POR MONTÓN es un mejoramiento de la clasificación por árbol pues este último presenta muchas comparaciones superfluas y para *n* unidades hay que crear *2*n -1* unidades. Se define **montón** a la secuencia de llaves $[h_L, h_{L+1,...}, h_R]$ tales que

$h_i <= h_{2*1}$ y $h_i <= h_{2*i+1}$ para L...R/2, esto es, dado el arreglo: [h_1,h_2,h_3,h_4,h_5,h_6,h_7,h_8] estará ordenado parcialmente y en montón si:

h_1 (la llave mínima porque se opera con "<=")

$h_1 <= h_2$ y $h_1 <= h_3$

$h_2 <= h_4$ y $h_2 <= h_5$ $h_3 <= h_6$ y $h_3 <= h_7$

$h_4 <= h_8$

- **Método**, se realiza de dos formas bien diferenciadas, éstas son:

- crear **montones** u ordenamiento **parcial** del arreglo. Ejemplo:

[h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8]

[44 55 12 42 94 18 6 67]

- el arreglo inicialmente no está ordenado parcialmente o en montón puesto que NO cumple la definición anterior; se procede a ordenarlo (1º bucle WHILE):

- inicialmente 44 55 12 42 94 18 6 67

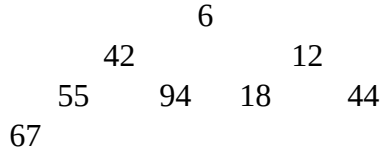
- ordena por montones de abajo hacia arriba según definición, esto es, ordena el montón $h_4 <= h_8$ 44 55 12 **42** 94 18 6 **67**
¿está ordenado?. Sí, continuaría en h_8 si existiera en el arreglo h_{16} pues no puede comparar $h_8 <= h_{16}$

- luego, el montón $h_3 <= h_6$ y $h_3 <= h_7$ 44 55 **12** 42 94 **18** **6** 67
¿está ordenado?. NO, intercambia la llave h_3 con la menor llave (subrayado) entre h_6 y h_7: 44 55 **6** 42 94 **18** **12** 67
¿está ordenado?.Sí, continuaría en h_7 (elemento intercambiado) si h_{14} existiera pues no puede comparar $h_7 <= h_{14}$

- luego, el montón $h_2 <= h_4$ y $h_2 <= h_5$ 44 **55** 6 **42** **94** 18 12 67
¿está ordenado?. NO, intercambia la llave h_2 con la menor llave (subrayado) entre h_4 y h_5: 44 **42** 6 **55** **94** 18 12 67
¿está ordenado?. Sí, continúa pues existe h_4 (elemento intercambiado) con el montón $h_4 <= h_8$ 44 42 6 **55** 94 18 12 **67**
¿está ordenado? Sí, continuaría en h_8 si existiera h_{16} en el arreglo pues no puede comparar $h_8 <= h_{16}$

- luego, ordena el montón $h_1 <= h_2$ y $h_1 <= h_3$ **44** **42** **6** 55 94 18 12 67
¿está ordenado?. NO, intercambia la llave h_1 con la menor llave (subrayado) entre h_2 y h_3: **6** **42** **44** 55 94 18 12 67
¿está ordenado?. Sí, continúa pues existe h_3 (elemento intercambiado), con $h_3 <= h_6$ y $h_3 <= h_7$ 6 42 **44** 55 94 **18** **12** 67
¿está ordenado?. NO, intercambia la llave h_3 con la menor llave (subrayado) con la menor llave entre h_6 y h_7: 6 42 **12** 55 94 **18** **44** 67
¿está ordenado?. Sí, continuaría en h_7 si h_{14} existiera en el arreglo pues no puede comparar $h_7 <= h_{14}$

- resultando: 6 42 12 55 94 18 44 67



- ordenamiento **completo** (2º bucle WHILE): en cada paso se intercambian el último componente con el primero (mínima llave del arreglo por definición, en negrita) del arreglo y luego se reordena el nuevo montón (subrayado) hasta alcanzar su posición correspondiente en . Ejemplo:

- resultado del anterior bucle: 6 42 12 55 94 18 44 67

- intercambio de valores: **67** 42 12 55 94 18 44 **6**
- resultado del reordenamiento del nuevo montón (subrayado):
12 42 18 55 94 67 44 6

- intercambio de valores: **44** 42 18 55 94 67 **12** 6
- resultado del reordenamiento del nuevo montón (subrayado):
18 42 44 55 94 67 12 6

- Programa:

```
MODULE ClasificacionMonton;
  FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read,WriteInt;
  CONST N=8; (* longitud de la cadena *)
  TYPE tira= ARRAY [1..N] OF INTEGER;
  VAR cadena:tira;L,R:CARDINAL; x:INTEGER;
  PROCEDURE sift(Ls,Rs:CARDINAL);
    VAR i,j:CARDINAL; x:INTEGER;
    BEGIN
    i:=Ls; j:= 2*Ls; x:= cadena[Ls];
    IF (j < Rs) & (cadena[j+1] < cadena[j]) THEN j:= j+1; END;
    WHILE (j <= Rs) & (x > cadena[j]) DO
      cadena[i]:= cadena[j];
      cadena[j]:=x;
      i:=j; j:=2*j;
      IF (j < Rs) & (cadena[j+1] < cadena[j]) THEN j:= j+1; END;
    END;
  END sift;
```



```
  BEGIN
  cadena[1]:= 44;cadena[2]:= 55;cadena[3]:= 12;cadena[4]:= 42;cadena[5]:= 94;
  cadena[6]:= 18;cadena[7]:= 06; cadena[8]:= 67;
  (* INSERCIàN por montones o  rbol *)
  L:= (N DIV 2)+1; R:= N;
  WHILE  L > 1 DO
    L:= L-1;
    sift(L,R);
  END;
  WHILE  R > 1 DO
    x:= cadena[1]; cadena[1]:= cadena[R]; cadena[R]:= x;
    R:= R-1;
    sift(L,R);
  END;
  END ClasificacionMonton.
```

- Análisis: - en el peor de los casos, su rendimiento es excelente n * log n
- en arreglos clasificados en orden inverso (más o menos) el promedio es n/2 * log(n)

2.3.3 CLASIFICACIÓN POR PARTICIÓN (ó CLASIFICACIÓN RÁPIDA ó QUICKSORT)

- **Método**, se selecciona un elemento al azar (aunque mejora su rendimiento si se produce en la mitad del arreglo), partiéndose en dos mitades. Se quiere conseguir que en la mitad superior estén las llaves mayores que el elemento seleccionado (negrita) al azar y en la parte inferior los menores. Se rastrea el arreglo mediante dos punteros desde afuera hacia dentro y se intercambian (subrayados) pares de elementos que NO cumplan la condición indicada. Produciéndose el intercambio de llaves (aunque no estarán ordenados totalmente). Luego, cada uno de los subarreglos semiordenados resultantes (superior e inferior), se consideraran como un arreglo, iniciando el algoritmo hasta que no se puedan subdividir.

Ejemplo:

- Inicialmente:	44 55 12 42 94 6 18 67	
- selección elemento:	44 55 12 **42** 94 6 18 67	
- intercambios:	44 55 12 **42** 94 6 18 67	
	18 55 12 **42** 94 6 44 67	
	18 6 12 **42** 94 55 44 67	
- partición del arreglo:	18 6 12	94 55 44 67
- selección elemento:	18 **6** 12	94 **55** 44 67
- intercambios en subarreglos:	6 **18** 12	44 **55** 94 67
- partición del arreglo:	18 12	94 67
- selección elemento:	**18** 12	**94** 67
- intercambios en subarreglos:	12 18	67 94

- **Programa**:

```
MODULE ClasificacionRapida;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read,WriteInt;
CONST N=8; (* longitud de la cadena *)
   TYPE tira= ARRAY [1..N] OF INTEGER;
(* cadena para la b£squeda y la SUBselecci¢n *)
VAR cadena:tira;
```



```
   PROCEDURE sort(L,R:CARDINAL);
   VAR    i,j:CARDINAL;
          w,x:INTEGER;
   BEGIN
   i:= L;j:=R;
   x:= cadena[(L+R) DIV 2];
   REPEAT
      WHILE cadena[i] < x DO
         i:= i+1;
      END;
      WHILE x < cadena[j] DO
         j:= j-1;
      END;
      IF i <= j THEN
         w:= cadena[i];
         cadena[i]:= cadena[j];
         cadena[j]:= w;
```

```
        i:= i+1;
        j:= j-1;
      END;
  UNTIL i > j;
  IF L < j THEN
     sort(L,j);
  END;
  IF i < R THEN
     sort(i,R);
  END;
  END sort;

BEGIN
cadena[1]:= 44;cadena[2]:= 55;cadena[3]:= 12;cadena[4]:= 42;cadena[5]:= 94;
cadena[6]:= 6;cadena[7]:= 18; cadena[8]:= 67;
sort(1,N);
END ClasificacionRapida.
```

- **Análisis** probabilístico: el número de comparaciones será *log n*
el número de intercambios será 2 * ln2

Su rendimiento es moderado para valores pequeños de *n*

El caso peor (se selecciona la llave más grande de una partición) será de n^2

2.3.4 OBTENCIÓN DE LA MEDIANA. La mediana es el elemento que es menor que la mitad de *n* elementos y que es mayor que la otra mitad

- **Método** similar a la clasificación rápida pues no llega a clasificar totalmente el arreglo:
  - se elige la llave a_k, dividiendo al arreglo en dos partes iguales. Ejem:

    16 12 99 **95** 18 87 10 16 95 99 **12** 18 87 10

  - produce una partición similar a la clasificación rápida, presentándose que:

    - el valor a_k es demasiado PEQUEÑO, en consecuencia, el índice *k* no está entre los índices *i,j*. Repitiéndose otro proceso de partición entre los elementos $a_i..a_R$. Ejem:

      95 12 99 **16** 18 87 10
      10 12 16 ***95*** *18 87 99*

    - el valor a_k es demasiado GRANDE, en consecuencia, el índice *k* no está entre los índices *i,j*. Repitiéndose otro proceso de partición entre los elementos $a_L..a_j$. Ejem:

      16 12 99 **95** 18 87 10
      16 12 10 **87** 18 95 99

    - el valor a_k divide al arreglo en dos partes y está el índice *k* entre los índices *i,j*. Ejem:

      16 12 10 **18** 87 95 99 10 12 16 **18** *95 87 99*

- **Programa**:

```
MODULE ObtencionMediana;
   FROM InOut IMPORT WriteLn,WriteString,WriteCard,Write,Read,WriteInt;
CONST N=7; (* longitud de la cadena *)
   TYPE tira= ARRAY [1..N] OF INTEGER;
(* cadena para la b£squeda y la SUBselecci¢n *)
VAR cadena:tira; posMediana:CARDINAL;
```

```
    PROCEDURE Find(k:CARDINAL);
    VAR     L,R,i,j:CARDINAL; w,x:INTEGER;
    BEGIN
    L:= 1;R:= N;
    WHILE L < R DO
       x:= cadena[k]; i:= L; j:=R;
       REPEAT
          WHILE cadena[i] < x DO i:= i+1; END;
          WHILE x < cadena[j] DO j:= j-1; END;
          IF i <= j THEN
             w:= cadena[i];
             cadena[i]:= cadena[j];
             cadena[j]:= w;
             i:= i+1;
             j:= j-1;
          END;
       UNTIL i > j;
       IF j < k THEN L:= i END;
       IF k < i THEN R:= j END;
    END;
    END Find;
BEGIN
(* inicio *)
cadena[1]:= 16;cadena[2]:= 12;cadena[3]:= 99;cadena[4]:= 95;cadena[5]:= 18;
cadena[6]:= 87;cadena[7]:= 10;
posMediana:= (1+N) DIV 2;
Find(posMediana);
END ObtencionMediana.
```

- **Análisis** en el mejor caso tiene un rendimiento de $n * log\ n$
en el peor caso tiene un rendimiento de n^2

2.3.5 UNA COMPARACIÓN DE LOS MÉTODOS DE CLASIFICACIÓN DE ARREGLOS, con una permutación aleatoria de mayor rendimiento a menor, será:

Clasificación Rápida
Mezcla directa (ver 2.4.1Secuencias de Clasificación)
Clasificación por montón
Shell (Inserción por incremento decreciente)
Inserción Binaria
Inserción directa
Seleción directa
Vibración
Burbuja

2.4 SECUENCIAS DE CLASIFICACIÓN. Esta vez se trata de clasificar un archivo dado mediante técnicas que usen poca memoria principal y MUCHO la memoria secundaria (DISCO);
su uso viene determinado por el tamaño del fichero a tratar en comparación con la memoria principal que posea el computador. Pero los ejemplos se realizan en arreglos para evitar complejidad.

2.4.1 MEZCLA DIRECTA.

Mezclar significa combinar dos o más secuencias en una sola ordenada por medio de una selección repetida entre los componentes accesibles.

Fase se le llama a cada operación que trata todo el conjunto de datos una sola vez.

- **Método** consiste en dividir la secuencia en dos mitades y clasificar inicialmente por parejas que luego se irán aumentando las longitudes de las mezclas por 2, ésto es:

- inicio:

a : 44 55 12 42 94 18 06 67

- 1ª Fase:

- Distribuye: divide la secuencia *a* en **dos mitades** (*b*,*c*)

b: 44 55 12 42
c: 94 18 06 67

- Mezcla cada elemento de *b*,*c* combinando en **pares ordenados** y los resultados los devuelve a la secuencia *a*:

b: 44 55 12 42 ⎫
⎬ → a: **44 94'** 18 55' 06 12' 42 67'
c: 94 18 06 67 ⎭

- 2ª Fase:

- Distribuye: divide la secuencia *a* en **dos mitades** (*b*,*c*)

b: 44 55 12 42
c: 94 18 06 67

- Mezcla cada elemento de *b*,*c* combinando **dos cuáduplos ordenados** y los resultados los devuelve a la secuencia *a*:

b: 44 55 12 42 ⎫
⎬ → a: **18 44 55 94'** 06 12 42 67'
c: 94 18 06 67 ⎭

- 3ª Fase:

- Distribuye: divide la secuencia *a* en **dos mitades** (*b*,*c*)

b: 18 44 55 94'
c: 06 12 42 67'

- Mezcla cada elemento de *b*,*c* combinando **octetos ordenados** y los resultados los devuelve a la secuencia *a*:

b: 18 44 55 94 ⎫
⎬ → a: **06 12 18 42 44 55 67 97 '**
c: 06 12 42 67 ⎭

- **Análisis**: realiza log n pases, siendo el total de modificaciones de elementos n * log (2*n); la principal desventaja es el tener que guardar 2 * n elementos, ésto hace que no sea utilizada en la clasificación de arreglos

2.4.2 MEZCLA NATURAL.

Se llama **corrida** a una subsecuencia ordenada $a_i..a_j$ (subrayada), esto es:

$(a_{i-1} > \mathbf{a_i})$ & (Ak: i<= k < j : $a_k <= a_{k+1}$) & $(\mathbf{a_j} > a_{j+1})$

Ejem: 12' 06 18' **02** 04 08 **08**' 01

- Método, similar a la mezcla directa pero **combina corridas**, en lugar de secuencias de longitud fija previamente determinada

En cada PASE : **C** ⇒distribuye { → **A**, → **B** } mezcla ⇒ **C**

Ejemplo: 12 06 18 02 04 08 08 01⇒ { → 12 02 04 08 08, → 06 18 01 } ⇒ **06** 12 **18** **01** 02 04 08 08⇒

⇒**06** 12 **18** **01** 02 04 08 08 ⇒{ → 06 12 18, → 01 02 04 08 08 } ⇒ 01 02 04 **06** 08 08 **12 18**

El problema empieza cuando inicialmente C tiene las corridas IMpares, y en consecuencia, habrá 1 corrida más en A ó B. La solución será que durante la mezcla si se llega al final de una secuencia, habrá que **copiar la corrida restante** (no mezclada) en C para no perder la información.

Ejemplo: 4 5 2 8 6 7 1 3 ⇒{ → 4 5 6 7, → 2 8 1 3 } ⇒

⇒06 12 18 **02 04 08 08** ⇒{ → 06 12 18, → 02 04 08 08 } ⇒ 02 04 06 08 08 12 18

Pero el problema se agrava cuando una de la secuencias distribuidas (A ó B) hayan muchas más corridas que en la otra. Ejem:

Ejemplo: 17 19 13 57 23 29 11 59 31 37 07 61 ⇒{ → 17 19 23 29 31 37, → 13 57 11 59 07 61 } ⇒

⇒13 17 19 23 29 31 37 57 11 59 ~~**07 61**~~ (se perdería la información 07 61)

La solución estará en copiar en C **la secuencia** restante (NO mezclada) y no la corrida como antes se afirmaba. Ejem:

Ejemplo: 17 19 13 57 23 29 11 59 31 37 07 61 ⇒{ → 17 19 23 29 31 37, → 13 57 11 59 07 61 } ⇒

⇒13 17 19 23 29 31 37 57 11 59 **07 61**⇒{ → 13 17 19 23 29 31 37 57 07 61, → 11 59 } ⇒

⇒11 13 17 19 23 29 31 37 57 59 **07 61**⇒{ → 11 13 17 19 23 29 31 37 57 59, → 07 61 } ⇒

⇒ **07** 11 13 17 19 23 29 31 37 57 59 **61**

- Programa:

```
MODULE SecuenciaNatural;
  FROM Sequences IMPORT  item, Sequence, OpenSeq, OpenRandomSeq,
       StartRead,StartWrite,copy,CloseSeq,ListSeq,Traza;
  FROM InOut IMPORT Read,WriteString,WriteInt,WriteLn;
  VAR L:INTEGER; a,b,c:Sequence;
  PROCEDURE copyrun(VAR x,y:Sequence);
  BEGIN
  REPEAT copy(x,y) UNTIL x.eor
  END copyrun;
```



```
  BEGIN
  OpenSeq(a);OpenSeq(b); OpenRandomSeq(c);
```

```
REPEAT (* separa la secuencia en corridas *)
   StartWrite(a);StartWrite(b);StartRead(c);
   REPEAT
      copyrun(c,a);
      IF NOT(c.eof) THEN copyrun(c,b);  END;
   UNTIL c.eof;
   StartRead(a);StartRead(b);StartWrite(c);
   L:= 0;
   REPEAT
      LOOP
         IF a.first < b.first THEN
            copy(a,c);
            IF a.eor THEN copyrun(b,c);  EXIT; END;
         ELSE
             copy(b,c);
            IF b.eor THEN copyrun(a,c); EXIT; END;
         END;
      END;
      L:= L + 1;
   UNTIL a.eof OR b.eof;
   WHILE ~a.eof DO
      copyrun(a,c);
      L:= L +1;
   END;
   WHILE ~b.eof DO
      copyrun(b,c);
      L:= L +1;
   END;
UNTIL L = 1;
ListSeq(c);CloseSeq(a);CloseSeq(b);CloseSeq(c);
END SecuenciaNatural.
```

- **Análisis**: en el peor caso produce n * log n comparaciones

2.4.3 MEZCLA BALANCEADA MÚLTIPLE

- **Método**: consiste en distribuir las corridas en más de dos secuencias. Mezclando *r* corridas que están distribuidas en *N* secuencias origina una secuencia de r/N corridas en el primer pase, tras el segundo pase será de r/N^2.

- **Análisis** comparaciones en el pero caso será de $n * \log_N n$
  total de pases será de $\log_N n$

2.4.4 CLASIFICACIÓN POLIFÁSICA

- **Método**:

1 - **Se distribuyen** las corridas en ***N-1* secuencias** y se deja otra vacía (*N*)

2 - **Se mezclan** las corridas de cada secuencia en otra secuencia vacía (*N*).
Si las secuencias fuentes quedan:

- LLENAS, significará que no está complemente ordenado y habrá que volver al punto **2**, pues siempre quedará una secuencia vacía.

- VACÍAS completamente, significará que en la secuencia destino de la mezcla sólo habrá una corrida y estará ordenada

Ejem (destino en negrita y subrayado la secuencia con menos corridas):

1 distribución en 6 (=*N*)secuencias y se distribuye en 5 (=*N-1*) :

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	16	15	14	12	8	**0**

2 mezcla, resultando las secuencias LLENAS

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	8	7	6	4	**0**	8

2 mezcla, resultando las secuencias LLENAS

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	4	3	2	**0**	4	4

2 mezcla, resultando las secuencias LLENAS

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	2	1	**0**	2	2	2

2 mezcla, resultando las secuencias LLENAS

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	1	0	**1**	1	1	1

2 mezcla, resultando las secuencias **VACÍAS** y f2 ordenada

secuencias:	f1	f2	f3	f4	f5	f6
corridas:	0	1	0	0	0	0

El problema está en **optimizar** la primera distribución en *N*-1 secuencias iniciales para que haya un mínimo número de pases o mezclas. Para ello, se recurre a los **Números de Fibonacci**:

cada número f_{i+1} es la suma de sus *p* predecesores y si no tiene ó son menores que 0 serán 0 los sumandos). Esto es:

$f_{i+1}(p) = f_i(p) + f_{i-1}(p) + ...+ f_{i-p}(p)$ para i >= p

$f_p(p) = 1$

$f_i(p) = 0$ para 0 < = i < p

Ejem: - elemento: $\mathbf{f_3(2)} = f_{2+1}(2) = f_2(2) + f_1(2) = 1 + 0 = \mathbf{1}$

- serie: f (2)= 0(= 0+0),1=(0+1),**1(=0+1)**, 2(=1+1),3(=1+2),(=3+2),....

f (6) = 0(=1<6),0(= 2<6),0(= 3<6),0(=4<6),0(= 5<6),

1(=(6=6)),1(= 0+0+0+0+0+1),2, 4,8,16(= 8+4+2+1+1),....

Distribución con números de Fabonacci por niveles (L):

1- se selecciona en la tabla 2.15 (pag129) el número más aproximado de corridas al real por exceso.

Niveles/secuencias +2	**1**	**2**	**3**	**4**	**5**	**6**
2	3	5	7	9	11	13
3	5	9	13	17	21	25
4	8	17	25	33	41	49

5	13	31	49	65	81	97
6	21	57	94	129	161	193

2- se rellena la diferencia con corridas ficticias

3 - sabiendo el número de secuencias a aplicar, realizamos la tabla