Tema 1. ORDENACIÓN, BÚSQUEDA E INTERCALACIÓN INTERNA (Algoritmos)

1. Declaraciones previas

Para los algoritmos que aparecen a continuación se supone que se han realizado las siguientes declaraciones globales:

```
const
   numElementos = ... //Número de elementos del array a ordenar
tipos
   //tipoElemento podrá ser cualquier tipo de dato simple, por ejemplo
   entero = tipoElemento
   array[0..numElementos] de tipoElemento = vector
```

También se considera que está presente el siguiente procedimiento:

```
procedimiento intercambia(ref tipoElemento : a,b)
var
   tipoElemento : aux
inicio
   aux ← a
   a ← b
   b ← aux
fin_procedimiento
```

2. Ordenación interna

2.1. Ordenación por intercambio directo ("burbuja")

```
procedimiento OrdenaciónIntercambioDirectoBásico(ref vector:v;valor
entero : n)
var
   entero : i,j
inicio
   desde i ← 1 hasta n-1 hacer
      desde j ← n hasta i+1 incremento -1 hacer
         si v[j-1] > v[j]entonces
            intercambiar(v[j],v[j-1])
         fin_si
      fin_desde
```

```
   fin_desde
fin_procedimiento

procedimiento OrdenaciónIntercambioDirecto(ref vector:v;valor entero : n)
var
   entero : i,j
   lógico : ordenado
inicio
   i ← 0
   repetir
      i ← i + 1
      ordenado ← verdad
      desde j ← n hasta i+1 incremento -1 hacer
         si v[j-1] > v[j]entonces
            intercambiar(v[j],v[j-1])
            ordenado ← falso
         fin_si
      fin_desde
   hasta_que ordenado
fin_procedimiento
```

2.2. Ordenación por selección directa

```
procedimiento OrdenaciónSelecciónDirecta(ref vector:v;valor entero : n)
var
   entero : i,j,min
inicio
   desde i ← 1 hasta n-1 hacer
      min ← i
      desde j ← i+1 hasta n hacer
         si v[j] < v[min] entonces
           min ← j
         fin_si
      fin_desde
      intercambiar(v[i],v[min])
   fin_desde
fin_procedimiento
```

2.3. Ordenación por inserción

Ordenación por inserción directa

En este algoritmo se utiliza el elemento 0 del array como centinela de forma que sale del bucle si el índice `j` llega a ese elemento. Se ordenan los elementos entre 1 y n.

```
procedimiento OrdenaciónInserciónDirecta(ref vector:v;valor entero : n)
//El vector v tiene elementos entre 1 y n
var
   entero : i,j
inicio
```

```
   desde i ← 2 hasta n hacer
      //Se almacena el elemento a insertar (v[i]) en la posición 0
      //del array para que actúe como centinela
      v[0] ← v[i]
      j ← i - 1
      //Se desplazan todos los elementos mayores que v[0]
      //y situados a su izquierda una posición a la derecha
      mientras v[j] > v[0] hacer
          v[j+1] ← v[j]
          j ← j - 1
      fin_mientras
      //En la posición siguiente al primer elemento menor o igual
      //se inserta el elemento v[0]
      v[j+1] ← v[0]
   fin_desde
fin_procedimiento
```

En esta otra versión no se utiliza el elemento 0 del array como centinela y puede utilizarse de forma que se ordenan todos los elementos entre 0 y n.

```
procedimiento OrdenaciónInserciónDirecta(ref vector:v;valor entero : n)
//El vector v tiene elementos entre 0 y n
var
   entero : i,j
   tipoElemento : aux
inicio
   //Puesto que el primer elemento (elemento 0)ya está colocado resepcto
   //a él mismo, se comienza a colocar elementos a partir del segundo
   //elemento, es decir, el elemento 1 del array
   desde i ← 1 hasta n hacer
      //Se almacena el elemento a insertar (v[i]) en la variable
      //aux para guardar su valor
      aux ← v[i]
      j ← i - 1
      //Se desplazan todos los elementos mayores que aux
      //y situados a su izquierda una posición a la derecha
      //Se sale del bucle cuando se encuentra un elemento menor o igual
      //o cuando el índice j llega a 0
      mientras j>= 0 y v[j] > aux hacer
          v[j+1] ← v[j]
          j ← j - 1
      fin_mientras
      //En la posición siguiente al primer elemento menor o igual
      //se inserta el elemento aux
      v[j+1] ← aux
   fin_desde
fin_procedimiento
```

Ordenación por inserción binaria

```
procedimiento OrdenaciónInserciónBinaria(ref vector:v ;valor entero : n)
var
   entero : i,j,iz,de,ce
   tipoElemento : aux
inicio
   desde i ← 2 hasta n hacer
      aux ← v[i]
      iz ← 1
      de ← i-1
      mientras iz <= de hacer
         ce ← (iz + de) div 2
         si aux < v[ce] entonces
            de ← ce - 1
         si_no
            iz ← ce + 1
         fin_si
      fin_mientras
      desde j ← i - 1 hasta iz incremento -1 hacer
          v[j+1] ← v[j]
      fin_desde
      v[iz] ← aux
   fin_desde
fin_procedimiento
```

2.4. Ordenación por incrementos decrecientes (Shell)

```
procedimiento OrdenaciónShell(ref vector:v; valor entero : n)
var
   //incr es la separación entre elementos a comparar
   entero : i,j,incr
   tipoElemento : aux
inicio
   //Inicialmente la separación entre elementos a comparar es n/2
   incr ← n div 2
   //Se repite el método de ordenación por inserción mientras
   //que la separación sea mayor que 0
   mientras incr > 0 hacer
     desde i ← incr + 1 hasta n hacer
       aux ← v[i]
       j ← i - incr
      //Se comparan el elemento auxiliar con el situado
      //incr posiciones más a la derecha (elemento v[j])
      mientras j>=1 y v[j]>aux hacer
        v[j+incr] ← v[j]
        j ← j-incr
      fin_mientras
      v[j+incr] ← aux
```

```
      fin_desde
      //Una vez que la lista está ordenada entre los elementos
      //situados a incr posiciones, el incremento decrece
      incr ← incr div 2
   fin_mientras
fin_procedimiento
```

3. Búsqueda

3.1. Búsqueda secuencial o lineal

```
//La función recibe el vector donde se va a buscar,
//un dato de tipoElemento que será el elemento a buscar y
//el número de elementos del array.
//Se supone que tipoElemento es un tipo de dato
//que admite los operadores de relación.
entero función Buscar(valor vector:v;valor tipoElemento:el;valor
entero:n)
var
   entero: i
inicio
   i ← 1
   mientras (el <> v[i]) y (i < n) hacer
      i ← i + 1
   fin_mientras
   si el = v[i] entonces
      devolver(i)
   si_no
      devolver(0)
   fin_si
fin_función
```

Búsqueda secuencial con centinela

```
entero función Buscar(valor vector:v; valor tipoElemento:el; valor
entero:n)
var
   entero: i
inicio
   i ← n
   v[0] ← el
   mientras el <> v[i] hacer //El elemento siempre está
      i ← i - 1
   fin_mientras
   //La función siempre devuelve i
   devolver(i)
fin_función
```

Búsqueda secuencial en un array ordenado

```
//La búsqueda se realiza sobre un array ordenado
entero función Buscar(valor vector:v; valor tipoElemento:el; valor
entero:n)
var
   entero: i
inicio
   i ← 1
   mientras (el > v[i]) y (i < n) hacer
      i ← i + 1
   fin_mientras
   si el = v[i] entonces
      devolver(i)
   si_no
      devolver(0)
   fin_si
fin_función
```

3.2. Búsqueda binaria o dicotómica

```
entero función Buscar(valor vector:v; valor tipoElemento:el; valor
entero:n)
var
   entero: izq, der, cen
inicio
   izq ← 1
   der ← n
   repetir
      cen ← (izq + der) div 2
      si v[cen] > el entonces
         der ← cen – 1
      si_no
         izq ← cen + 1
      fin_si
   hasta_que (v[cen] = el) o (izq > der)
   si v[cen] = el entonces
       devolver(cen)
   si_no
       devolver(0)
   fin_si
fin_función
```

3.3. Búsqueda por transformación de claves (hash)

```
//Se desea buscar el registro con clave claveBuscada
dirección ← hash(claveBuscada)
si v[dirección].clave < 0 entonces
   //El elemento no se encuentra
si_no
```

```
   si v[dirección].clave <> claveBuscada entonces
      //Puede que se trate de un sinónimo
      //Se busca entre las posiciones siguientes
      //hasta que se encuentra o hasta hallar un hueco vacío
      repetir
         dirección ← dirección mod 120 + 1
      hasta_que (v[dirección].clave=claveBuscada) o v[dirección].clave<0
   fin_si
fin_si

si v[dirección].clave = claveBuscada entonces
   //El elemento se encuentra en la posición dirección
si_no
   //El elemento no se encuentra
fin_si

entero función hash(valor entero: clave)
inicio
   devolver(clave mod numElementos +1)
fin_función
```

4. Intercalación

```
procedimiento Intercalación(valor vector:A,B;
                            ref vector:C; valor entero: m,n)
var
   //i será el índice de A, j el de B y k el de C
   entero : i,j,k
inicio
   i ← 1
   j ← 1
   k ← 1
   //Mientras no se acabe alguno de los arrays de entrada
   mientras (i <= M) y (j <= N) hacer
      si A[i] < B[j] entonces
         //Se selecciona un elemento de A y se inserta en C
         C[k] ← A[i]
         //Se desplaza el índice del array A
         i ← i + 1
      si_no
         //Se selecciona un elemento de B y se inserta en C
         C[k] ← B[j]
         //Se desplaza el índice del array B
         j ← j + 1
      fin_si
      //Se desplaza el índice del array de salida
      k ← k + 1
   fin_mientras
```

```
   //Si se ha llegado al final del array B se vuelca todo
   //el contenido que queda de A en el array de salida
   mientras i <= M hacer
      C[k] ← A[i]
      i ← i + 1
      k ← k + 1
   fin_mientras
   //Si se ha llegado al final del array A se vuelca todo
   //el contenido que queda de B en el array de salida
   mientras j <= N hacer
      C[k] ← B[j]
      j ← j + 1
      k ← k + 1
   fin_mientras
fin_procedimiento
```