

TEMA1: BÚSQUEDA, ORDENACIÓN Y MEZCLA

1. Introducción

Una tarea muy común es la búsqueda de elementos.

La búsqueda supone pasar por todos los elementos en el caso de que no está el que busco.

Ahora bien, si los elementos están ordenados (por ejemplo de menor a mayor) la búsqueda se cortará cuando encuentre al elemento buscado o bien cuando encuentre uno mayor que el que busco.

Ordenación. Operación de organizar un conjunto de datos en algún orden establecido, orden numérico -creciente o decreciente- para los datos numéricos o bien alfabético para los caracteres.

- Ordenación en memoria interna: cuando ésta se produce en la memoria principal del ordenador.
- Ordenación en memoria externa: necesaria cuando el número de elementos a ordenar es demasiado grande para poder albergarlo en memoria principal.

2. Búsqueda

```
#define MAXIMO 20
class vector {
    int celda[MAXIMO];
    int nelem;
public:
    ...
    int BusquedaBasica (int buscado, int inicio, int fin);
//Devolverá el índice del elemento de la tabla celda que contiene
//buscado, entre los índices inicio y fin. Si no está devolverá -1.
//Tiene que ser  $0 \leq inicio \leq fin < nelem$ 
    int BusquedaLineal (int buscado, int inicio, int fin);
    // igual que antes.
    int BusquedaBinaria(int buscado, int inicio, int fin);
    // igual que antes.
};
```

```
int vector::BusquedaBasica(int buscado, int inicio, int fin) {
    int i,pos;

    i=inicio;
    while ( (i<fin) && (celda[i] != buscado) )
        i++;

    if (celda[i]==buscado)    pos=i;
    else                      pos=-1;

    return(pos);
}
```

**Precondición adicional .- Conjuntos de elementos ordenados.
(OJO)**

```
int vector::BusquedaLineal(int buscado, int inicio, int fin){  
    int i,pos;  
  
    i=inicio;  
    while ( (i<fin) && (celda[i]<buscado) )  
        i++;  
  
    if (celda[i]==buscado)    pos=i;  
    else                      pos=-1;  
    return(pos);  
}
```

```
int vector::BusquedaBinaria(int buscado, int inicio, int fin){  
    // También recibe el nombre de búsqueda dicotómica.  
    int i,j,centro;  
    int encontrado;  
  
    i=inicio;  
    j=fin;  
    encontrado=0;  
  
    while ( (encontrado==0) && (i<=j) ) {  
        centro = (i+j) /2 ;  
        if (buscado==celda[centro])    encontrado=1;  
        else if (buscado>celda[centro])    i=centro+1;  
        else                            j=centro-1;  
    }  
  
    if (encontrado==0)    centro=-1;  
    return(centro);  
}
```

Sea a un objeto del tipo vector en el que tabla de 20 elementos (de enteros), están ocupadas por 3, 5, 6, 7, 15, 23, 54, 70, 81, 82, 90, 93, 100, ... y nos dicen que apliquemos la siguiente operación:
a.BusquedaBinaria (5,0,5). //buscado, inicio, fin

En este caso tendríamos la siguiente situación:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	19
3	5	6	7	15	23	54	70	81	82	90	93	100	...	

Si hacemos seguimiento del método para el objeto a
a.BusquedaBinaria (5,0,5) :

i	j	centro	encontrado
0	5	2	0
1	1	1	1

El algoritmo termina porque lo ha encontrado y devuelve 1, que es el índice del elemento que contiene un 5.

Supongamos ahora **a.BusquedaBinaria(40,4,10)**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	19
3	5	6	7	15	23	54	70	81	82	90	93	100	...	

Y si hacemos un seguimiento del algoritmo,

i	j	centro	encontrado
4	10	7	0
4	6	5	0
6	6	6	0
6	5	-1	

Observamos que cuando $i > j$, el programa termina porque no lo ha encontrado, devolviendo -1.

3. Ordenación

```
#define MAXIMO 20
class vector {
    int celda[MAXIMO];
    int nelem;
    void intercambiar(int &x,int &y);
    // El valor que tiene x lo pondrá en y, y el valor de y en x
public:
    ...
    void Burbuja (int inicio, int fin);
    //Ordenará crecientemente la tabla celda entre los elementos inicio y
    //fin, ambos incluidos.  0 <= inicio <= fin < nelem.
    void Seleccion(int inicio, int fin);
    //Igual que antes
    void Insercion(int inicio, int fin);
    //Igual que antes
    void Shell(int inicio, int fin);
    //Igual que antes
};
```

```
void vector::intercambiar (int &x, int &y) {
    int aux;
    aux=x; x=y; y=aux;
}
```

ORDENACIÓN BURBUJA

```
void vector::Burbuja(int inicio, int fin){
    int pos,ele;

    for (pos=inicio; pos<fin; pos++)
        for (ele=fin; ele>=pos+1; ele--)
            if(celda[ele]<celda[ele-1])
                intercambiar(celda[ele],celda[ele-1]);
}
```

ORDENACIÓN POR SELECCIÓN

```
void vector::Seleccion(int inicio, int fin){
    int pos,pos_menor, ele;

    for (pos=inicio; pos<fin; pos++)
    {
        pos_menor=pos;
        for (ele=pos+1; ele<=fin; ele++)
            if (celda[ele]<celda[pos_menor])
                pos_menor=ele;

        intercambiar(celda[pos],celda[pos_menor]);
    }
}
```

ORDENACIÓN POR INSERCIÓN

```
void vector::Insercion(int inicio, int fin) {
    int i,j;
    int lugar;
    for (i= inicio +1; i <= fin; i++) {
        j=i;
        lugar=0;
        while ((j>inicio) && (lugar==0))
            {
                if (celda[j]<celda[j-1])
                    intercambiar(celda[j],celda[j-1]);
                else lugar=1;
                j--;
            }
    }
}
```

ORDENACIÓN SHELL.

```
void vector::Shell(int inicio, int fin) {
    int inc,i,j;
    int encontrado;

    inc = (fin - inicio + 1) /2;
    while (inc>0)
        {
            for (i=inc+inicio; i<=fin; i++)
                {
                    encontrado=0;
                    j=i;
                    while ( (j>=inicio+inc) && (encontrado==0) ) {
                        if (celda[j]<celda[j-inc])
                            intercambiar(celda[j],celda[j-inc]);
                        else encontrado = 1;
                        j=j-inc;
                    }
                }
            inc = inc / 2;
        }
}
```

4. Mezcla

```
#define MAXIMO 20

class vector {
    int celda[MAXIMO];
    int nelem;

public:
    vector (int n) { nelem = n}      ...
    vector mezcla(vector v);
// Partiendo de que los vectores v y el propio al que se le aplica el
//método están ordenados crecientemente devolverá un vector
//resultante de fusionar estos dos vectores con sus elementos ordenados
//de forma creciente.
};
```

```
vector vector::mezcla(vector v) {
    int i, j, k;
    if (nelem + v.nelem > MAXIMO)
        cout << "\nError: N° de elementos mayor de " << MAXIMO;
    else { vector aux(nelem+v.nelem);
        i=0; j=0; k=0;
        while ( (i<nelem) && (j<v.nelem) ) {
            if (celda[i]<v.celda[j]) {
                aux.celda[k]=celda[i];      i++;
            }
            else {
                aux.celda[k]=v.celda[j];    j++;
            }
            k++;
        }
        while (i<nelem) {
            aux.celda[k]=celda[i];          i++;      k++;
        }
        while (j<v.nelem) {
            aux.celda[k]=v.celda[j];        j++;      k++;
        }
        return(aux);
    }
}
```

Realice un seguimiento a los 4 procedimientos de ordenación vistos en clase con los siguientes datos:

0	1	2	3	4	5	6	7
10	5	2	18	23	14	11	9

SOLUCION.-

BURBUJA.-

10	5	2	18	23	14	11	9
10	5	2	18	23	14	9	11
10	5	2	18	23	9	14	11
10	5	2	18	9	23	14	11
10	5	2	9	18	23	14	11
10	2	5	9	18	23	14	11
2	10	5	9	18	23	14	11
2	10	5	9	18	23	11	14
2	10	5	9	18	11	23	14
2	10	5	9	11	18	23	14
2	5	10	9	11	18	23	14
2	5	10	9	11	18	14	23
2	5	10	9	11	14	18	23
2	5	9	10	11	14	18	23

SELECCIÓN.-

10	5	2	18	23	14	11	9
2	5	10	18	23	14	11	9
2	5	9	18	23	14	11	10
2	5	9	10	23	14	11	18
2	5	9	10	11	14	23	18
2	5	9	10	11	14	18	23

INSERCIÓN.-

10	5	2	18	23	14	11	9
5	10	2	18	23	14	11	9
2	5	10	18	23	14	11	9
2	5	10	18	23	14	11	9
2	5	10	18	23	14	11	9
2	5	10	14	18	23	11	9
2	5	10	11	14	18	23	9
2	5	9	10	11	14	18	23

SHELL.-

10	5	2	18	23	14	11	9
10	5	2	9	23	14	11	18
2	5	10	9	11	14	23	18
2	5	9	10	11	14	18	23

Ordenación con elementos de tipo string

```
#include <string.h> // por usar strcpy y strcmp
#define MAXIMO 20
typedef char string[25]
class vector {
    string celda[MAXIMO];
    int nelem;
    void intercambiar(string x, string y);
public:
    void Burbuja (int inicio, int fin);
};
```

```
void vector::Burbuja(int inicio, int fin){
    int pos,ele;
    for (pos=inicio; pos<fin; pos++)
        for (ele=fin; ele>=pos+1; ele--)
            if ( strcmp(celda[ele], celda[ele-1]) < 0)
                intercambiar(celda[ele],celda[ele-1]);
}
```

```
void vector::intercambiar (string x, string y) {
    string aux;
    strcpy(aux,x);
    strcpy(x,y);
    strcpy(y,aux);
}
```

```
// ORDENACIÓN CON TABLAS DE REGISTROS
```

```
#define MAXIMO 20
```

```
typedef char string[40];
```

```
struct persona {
```

```
    string    nombre;
```

```
    string    apellido1;
```

```
    string    apellido2;
```

```
    long     dni;
```

```
    ...
```

```
};
```

```
class vector {
```

```
    persona celda[MAXIMO];
```

```
    int nelem;
```

```
    void intercambiar(persona &x, persona &y);
```

```
public:
```

```
    ...
```

```
    void Burbuja (int inicio, int fin);
```

```
};
```

```
void vector::Burbuja(int inicio, int fin){
```

```
    int pos,ele;
```

```
    for (pos=inicio; pos<fin; pos++)
```

```
        for (ele=fin; ele>=pos+1; ele--)
```

```
            if(celda[ele].dni <celda[ele-1].dni)
```

```
                intercambiar(celda[ele],celda[ele-1]);
```

```
    }
```

```
void vector::intercambiar (persona &x, persona &y) {
```

```
    persona aux;
```

```
    aux=x; x=y; y=aux;
```

```
}
```

```
// BÚSQUEDA EN TABLAS DE REGISTROS
```

```
#define MAXIMO 20
```

```
typedef char string[40];
```

```
struct persona {
```

```
    string    nombre;
```

```
    string    apellido1;
```

```
    string    apellido2;
```

```
    long     dni; ...
```

```
};
```

```
class vector {
```

```
    persona celda[MAXIMO];
```

```
    int nelem;
```

```
    void intercambiar(persona &x, persona &y);
```

```
public:
```

```
    ...
```

```
    int BusquedaBinaria(int buscado, int inicio, int fin);
```

```
};
```

```
int vector::BusquedaBinaria(int buscado, int inicio, int fin){
```

```
    int i,j,centro;
```

```
    int encontrado;
```

```
    i=inicio;
```

```
    j=fin;
```

```
    encontrado=0;
```

```
    while ( (encontrado==0) && (i<=j) ) {
```

```
        centro = (i+j) /2 ;
```

```
        if (buscado==celda[centro].dni)    encontrado=1;
```

```
        else if (buscado>celda[centro].dni) i=centro+1;
```

```
        else                                j=centro-1;
```

```
        }
```

```
    if (encontrado==0) centro=-1;
```

```
    return(centro);
```

```
}
```

```

void vector::desconocido1 (int inicio, int fin) {
int k, izq, dcha, j, el_en_curso;
for (j= inicio+1; j <= fin; j++) {
    el_en_curso = celda[j];
    izq = inicio;
    dcha = j-1;
    while (izq <= dcha) {
        k = (izq + dcha) / 2 ;
        if (el_en_curso < celda[k] )    dcha = k-1;
        else                            izq = k+1;
    }
    for (k = j-1; k >= izq; k--)    celda[k+1] = celda[k];
    celda[izq] = el_en_curso;
}
}

```

```

void vector::desconocido2 (int inicio, int fin) {
int izq, dcha, ultimo_int, k;
izq = inicio +1;
dcha = fin;
ultimo_int = fin;
do {
    for (k=dcha; k >= izq; k--) {
        if (celda[k-1] > celda[k] ) {
            intercambiar (celda[k-1], celda[k] );
            ultimo_int = k;
        }
    };
    izq = ultimo_int + 1;
    for (k=izq; k <= dcha; k++) {
        if (celda[k-1] > celda[k] ) {
            intercambiar (celda[k-1], celda[k] );
            ultimo_int = k;
        }
    };
    dcha = ultimo_int -1;
} while (izq <= dcha);
}

```