INFORMÁTICA APLICADA curso 2006-2007
PRÁCTICA 2: Algoritmos de ordenación y búsqueda en vectores.

1. Objetivos de la práctica

Los objetivos de esta práctica son:

1. Aprender a utilizar el depurador del compilador utilizado en las prácticas.

2. Implementar los algoritmos de ordenación y búsqueda utilizados en vectores y comprobar su funcionamiento.

No es necesario conocer conceptos de programación en C que no se hayan estudiado previamente en la asignatura Fundamentos de Informática, aunque sí se deben estudiar detenidamente los algoritmos de ordenación y búsqueda que se exponen en la sección siguiente. El profesor de prácticas expondrá qué es y cómo se usa el depurador de programas disponible en Dev C++ así como los algoritmos de ordenación y búsqueda.

2. Algoritmos de ordenación y búsqueda en vectores

2.1 Ordenación de vectores

Ordenar un vector consiste en reorganizar sus elementos según alguna relación de orden. El objetivo de la ordenación es que la información se pueda recuperar fácil y rápidamente (por ejemplo, en una base de datos grande). La ordenación es bastante usual en programación, existiendo varios métodos o algoritmos distintos de ordenación. Cuando se selecciona el método a usar, debe tenerse en cuenta la eficacia del mismo en dos aspectos:

- En la memoria que utiliza
- En el tiempo de ejecución

Si N es el número de elementos que se quiere ordenar, los buenos algoritmos de ordenación realizan del orden de $N\log_2 N$ operaciones. En esta práctica vamos a estudiar algunos algoritmos que realizan del orden de N^2 operaciones. Son métodos directos, que son más cortos y fáciles de entender, de tal forma que si N es pequeño, son lo suficientemente eficaces.

2.1.1 Algoritmo de intercambio directo (burbuja)

Básicamente consiste en realizar varias iteraciones que llevan el elemento menor del vector hasta su posición. Se considera el vector dividido en dos partes: la parte ordenada y la parte desordenada. Inicialmente, se considera que todo el vector está desordenado. Se detecta el elemento menor del vector y se lleva a la posición más a la

izquierda posible, quedando por tanto colocado en su posición definitiva. El elemento que estaba en ese extremo de la izquierda se desplaza, junto con los demás, hacia la derecha, hasta llegar donde estaba el elemento menor. Este método también se conoce como "método de la *burbuja*" ya que si imaginamos el vector en posición vertical y considerando los elementos como burbujas con "pesos" en proporción a su valor, en cada pasada sobre el vector ascenderá hasta el nivel superior la "burbuja" de menor peso, desplazando a las demás hacia abajo.

Veamos un ejemplo. Si se quiere ordenar el vector v = (8,4,11,6,5), en la tabla siguiente se muestran las sucesivas iteraciones del algoritmo (en negrita aparecen los elementos que están colocados en su posición definitiva).

		ITERACIONES									
índice	**v. original**	**1ª (i=1)**				**2ª (i=2)**			**3ª (i=3)**		**4ª(i=4)**
0	8	8	8	8	**4**	**4**	**4**	**4**	**4**	**4**	**4**
1	4	4	4	4	8	8	8	**5**	**5**	**5**	**5**
2	11	11	5	5	5	5	5	8	8	**6**	**6**
3	6	5	11	11	11	6	6	6	6	8	**8**
4	5	6	6	6	6	11	11	11	11	11	**11**

Este algoritmo es bastante ineficiente (en el peor caso se deben hacer del orden de n^2 intercambios). Es posible realizar algunas mejoras, como por ejemplo detectar en una iteración intermedia que el vector ya está ordenado y parar el algoritmo. Sin embargo, el algoritmo más eficaz de ordenación, llamado *Quick Sort*, deriva del algoritmo de la burbuja, y es del orden $n\log_2 n$.

El alumno deberá escribir la función OrdenarBurbuja, que ordena el vector de *n* elementos pasados por dirección. El prototipo de la función es el siguiente:

```
void OrdenarBurbuja(float *v, int n);
```

La función OrdenarBurbuja deberá comprobar si el vector estaba ya previamente ordenado, informar por pantalla de esa situación y salir en ese caso.

2.1.2 Algoritmo de inserción directa

A continuación se describe el algoritmo de inserción directa. El alumno deberá codificarlo en lenguaje C y comprobar su funcionamiento.

Vamos a elaborar un método para ordenar. Partimos de este vector

5	7	3	2	4

Y queremos obtener este otro:

2	3	4	5	7

Por ejemplo, podemos pensar en obtener el máximo del vector original y colocarlo al final, intercambiándolo por el número que se encuentre en esa posición.

5	4	3	2	7

El número más a la derecha ya se encuentra ordenado, se encuentra en su posición final ¿Qué hacemos ahora? Volvemos a obtener el máximo del vector, excluyendo el elemento más a la derecha y volvemos a intercambiar.

2	4	3	5	7

El alumno deberá escribir la función OrdenarInsercionDirecta, que ordena el vector de *n* elementos pasados por dirección. El prototipo de la función es el siguiente:

```
void OrdenarInsercionDirecta(float *v, int n);
```

2.2 Algoritmos de búsqueda

Existen dos métodos básicos de búsqueda de un elemento en un vector: *lineal* y *binaria*. La búsqueda lineal ya se ha estudiado en Fundamentos de Informática, y en ella no hace falta que el vector esté ordenado. El algoritmo de búsqueda binaria requiere que el vector esté ordenado. Consiste en situarse en el elemento central del vector, comparando el elemento buscado con el elemento central. Si coincide, se ha terminado la búsqueda con éxito. Si el elemento a buscar es menor, rechazamos los elementos de la derecha, si es mayor, rechazaremos los elementos de la izquierda, y así sucesivamente.

El algoritmo puede expresarse en C como sigue, donde se supone que este código forma parte de una función que devuelve –1 si la variable *dato* no se encuentra en el vector, y el índice correspondiente en caso contrario.

```
// Devuelve el índice del vector si se encuentra
// Devuelve -1 en caso contrario
int BusquedaBinaria(float *v, float dato, int n)
{
inicio = 0;
final = n-1;
while (inicio <= final)
{
  medio = (inicio+final)/2;
  if (dato == v[medio])
    return (medio);
  else
    if (dato > v[medio])
      inicio = medio+1;
    else
      final = medio-1;
}
return -1;
}
```

3. Tareas a realizar

La práctica consiste en escribir un programa que realice las siguientes tareas:

a) Leer un vector de números reales por teclado. Se puede solicitar previamente el número de elementos que se desean introducir.
b) Ordenar el vector utilizando el método de la *burbuja*. Al ordenar el vector por el método de la burbuja debe indicarse si ya estaba ordenado, y caso de no estarlo, cuántos intercambios entre elementos se han realizado.
c) Ordenar el vector utilizando el algoritmo de inserción directa.
d) Buscar un elemento en el vector utilizando el algoritmo de *búsqueda binaria (*optativo*).*
e) El programa deberá mostrar un menú para elegir cada una de las opciones que se pueden ejecutar en el programa:

```
Practica 2

1.- Introducir vector
2.- Ordenar por el metodo de la burbuja
3.- Imprimir vector
4.- Busqueda binaria
5.- Ordenar por insercion directa
6.- Salir

Elegir opcion:
```

Siguiendo las indicaciones del profesor de prácticas, y en el orden que éste indique, deberán realizarse los siguientes pasos:

1) Crear y configurar un proyecto en Visual C++ que se llamará **practica1**. El fichero fuente se llamará *practica1.c*.

2) Escribir la función para leer el vector con el siguiente prototipo:

```
void LeerVector(float *v, int n)
void LeerVector(float v[], int n)
```

donde *v* es un vector de tamaño MAXVECTOR (50), y *n* es el número de elementos del vector que se van a leer.

3) Escribir la función de ordenación por el método de la burbuja, con el siguiente prototipo:

```
void Burbuja(float v[], int n)
```

4) Escribir la función para imprimir vectores, con el siguiente prototipo

```
void ImprimirVector(float v[], int n)
```

5) Escribir la función para buscar elementos utilizando el algoritmo de búsqueda binaria, con el siguiente prototipo (optativo):

```
int BusquedaBinaria(float v[], int n, float dato)
```

donde *dato* es el elemento a buscar.

6) Escribir la función de ordenación utilizando el algoritmo de selección directa, con el siguiente prototipo:

```
void OrdenarInsercionDirecta(float v[], int n)
```

7) Escribir la función main que llame a todas las funciones anteriores.

Notas:

- El programa debe estar estructurado dividiendo cada tarea en funciones.
- *Se recomienda diseñar y comprobar cada función independientemente del resto del programa.*