Práctica 1: *Algoritmos de ordenación y búsqueda en vectores*

.::1 Sesión::.

1. Objetivos de la práctica

El objetivo de esta práctica es 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.

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 $n\log_2 n$.

2.1.2 Algoritmo de selección directa

Imaginemos el siguiente vector:

4	2	1

La forma de proceder para ordenarlo mediante este algoritmo, es la siguiente:

1) Siempre comenzamos con el primer elemento de la izquierda.
2) Guardamos ese primer elemento del vector en una variable auxiliar. Comenzaremos por 0.

   menor = 0

3) Comparamos el segundo elemento con el primero y si es menor, le asignamos a la variable auxiliar la posición de ese elemento menor (menor=1). De lo contrario pasaríamos a comparar el tercer elemento con el segundo.

   ¿2 < 4? Sí
   menor = 1

4) Pasamos al siguiente elemento y lo comparamos con el anterior, es decir, ahora comparamos el tercero con el segundo y si es menor, le asignamos a la variable auxiliar la posicion de ese elemento menor (menor = 2)

¿1<2? Sí
menor = 2

5) Asi recorremos el vector completo hasta dar con el valor menor de él, y lo intercambiamos con el primer elemento. Es decir, pasaríamos de:

4	2	1

a este nuevo vector:

1	2	4

6) Seguimos ahora con el segundo elemento, es decir, buscaremos el elemento menor a partir de él a su derecha y lo intercambiaremos por él si éste fuera menor.

¿4<2? No. Por tanto ya hemos acabado

2.1.3 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

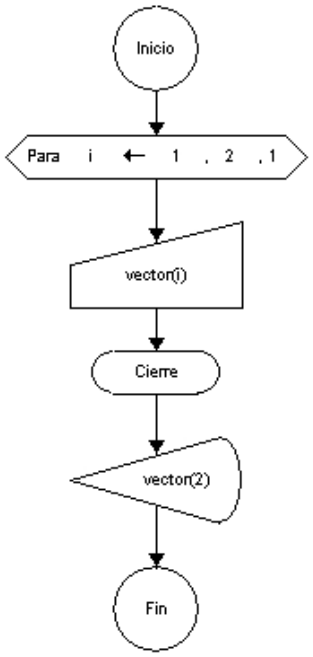
Existen dos métodos básicos de búsqueda de un elemento en un vector: *lineal* y *binaria*. La búsqueda lineal ya se ha estudiado en Fundamentos de Informática, y en ella no hace falta que el vector esté ordenado. El algoritmo de búsqueda binaria requiere que el vector esté ordenado. Consiste en situarse 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;
}
```

4. Implementación de Vectores mediante el software DFD

El software DFD soporta trabajar con vectores y matrices. Para la realización de nuestra práctica, es necesario conocer como se implementan vectores en dicho software. Para ello, en la figura siguiente, se puede observar un pequeño ejemplo de almacenamiento de datos en un vector, y la posterior muestra por pantalla de un elemento del mismo:



Podemos observar que la asignación de valores a un vector o matriz debe realizarse componente a componente, no pudiendo manejarse vectores completos. Para ello, haremos uso de bucles.

Para el ejemplo que nos ocupa de la figura, podemos observar que mediante un bucle `"for"` vamos insertando hasta 2 elementos en un vector llamado `"vector()"`, a través de la variable contador `"i"`, que como vemos en dicho bucle, insertará un valor para i=1 y otro para i=2.

Cabe destacar que si deseamos mostrar el valor de uno de los elementos de dicho vector, deberemos hacer referencia a él mediante el nombre del vector, seguido de la posición de dicho elemento entre paréntesis.

5. Paso de Vectores a funciones mediante el software DFD

Mediante el software DFD, es posible realizar el paso de vectores y matrices a funciones, y que estas a su vez devuelvan los devuelvan.

A continuación se muestra un ejemplo de paso de un vector de 2 elementos a una función llamada "`ModificoVector`", y como ésta modifica el vector que le pasamos asignándole un valor nuevo a cada uno de los 2 elementos del vector:

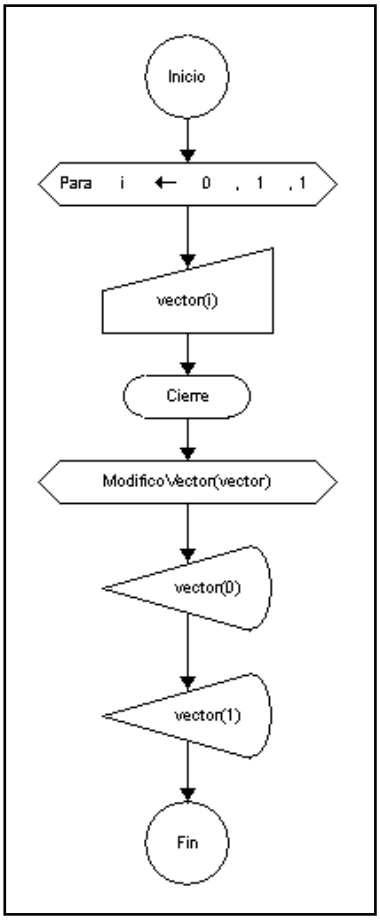


Figura 1. *Función main*

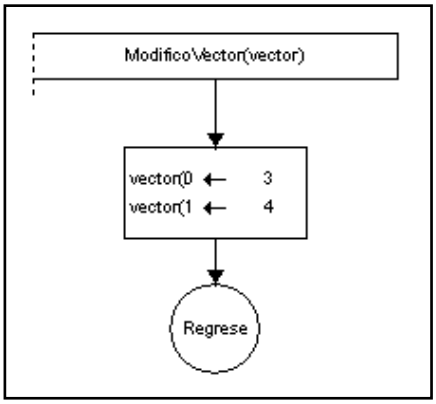


Figura 2. *Función ModificoVector*

6. Tareas a realizar

La práctica consiste en realizar las siguientes tareas

1.- DIAGRAMA DE FLUJO

Mediante el software DFD, realizar el diagrama de flujo que contenga todas las funciones descritas a continuación. Dicho diagrama de flujo se deberá testear y probar su correcto funcionamiento antes de pasar al apartado 2. Se deberá enviar al profesor el archivo practica1.dfd

a) Función `LeerVector`: Leerá un vector de números reales por teclado. Se puede solicitar previamente el número de elementos que se desean introducir.

Se definirá el siguiente prototipo para la función:

```
LeerVector(v,n)
```

donde *v* es un vector y *n* es el número de elementos del vector que se van a leer.

b) Función `OrdenarBurbuja`: Se ordenará el vector utilizando el método de la *burbuja*. Al ordenar el vector por dicho método, debe indicarse cuántos intercambios entre elementos se han realizado.

Se definirá la función con el siguiente prototipo:

```
OrdenarBurbuja(v,n)
```

c) Función `ImprimirVector` para imprimir vectores, con el siguiente prototipo:

```
ImprimirVector(v,n)
```

d) Función `OrdenarSeleccionDirecta` de ordenación utilizando el algoritmo de inserción directa, con el siguiente prototipo:

```
OrdenarSeleccionDirecta(v,n)
```

e) Función `BusquedaBinaria` para buscar elementos utilizando el algoritmo de búsqueda binaria, con el siguiente prototipo:

```
BusquedaBinaria(v, n, dato)
```

donde *dato* es el elemento a buscar.

f) Función `main` que llame a todas las funciones anteriores

2.- PROGRAMACIÓN EN C

Convertir el diagrama de flujo desarrollado a un programa en C. Es decir, dicho programa, deberá de contener:

a) Función `LeerVector`: Leerá un vector de números reales por teclado. Se puede solicitar previamente el número de elementos que se desean introducir.

Se definirá el siguiente prototipo para la función:

```
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.

b) Función `OrdenarBurbuja`: Se ordenará el vector utilizando el método de la *burbuja*. Al ordenar el vector por dicho método, debe indicarse si ya estaba ordenado, y en caso de no estarlo, cuántos intercambios entre elementos se han realizado.

Se definirá la función con el siguiente prototipo:

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

c) Función `ImprimirVector` para imprimir vectores, con el siguiente prototipo

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

d) Función `OrdenarSeleccionDirecta` de ordenación utilizando el algoritmo de inserción directa, con el siguiente prototipo:

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

e) Una nueva función llamada `OrdenarInsercionDirecta` de ordenación utilizando el algoritmo de inserción directa, con el siguiente prototipo:

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

f) Además, deberá contener un menú en el main que llame a todas las funciones mediante el siguiente menú:

```
Practica 1

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

Elegir opcion:
```

El programa se deberá guardar en el archivo practica1.c y dicho fichero se deberá enviar al profesor para la corrección de la práctica junto con el diagrama de flujo.

Notas:

- El programa debe estar estructurado dividiendo cada tarea en funciones.
- Dicho programa deberá corresponderse con el diagrama de flujo realizado anteriormente para la correcta puntuación del ejercicio.
- *Se recomienda diseñar y comprobar cada función independientemente del resto del programa.*