Ordenamiento de vectores, matrices y vectores de struct

Ordenar un vector permite, entre otras cosas, simplificar los procedimientos de búsqueda que deban realizarse dentro de él. A partir de un vector ordenado, se puede aplicar el algoritmo de búsqueda binaria –mucho más rápido que otros algoritmos- , o finalizar una búsqueda secuencial cuando el elemento del vector que se está comparando es mayor (o menor) que el valor que se busca. También es útil a la hora de mostrar un conjunto de valores hacerlo con algún criterio de ordenamiento que facilite la tarea del usuario.

Por las razones mencionadas, y otras que se verán en los distintos cursos de programación, el ordenamieno cumple un papel muy importante. Máxime cuando se trabaja con grandes volúmenes de información. Existen varios algoritmos de ordenamiento: selección, inserción, burbuja, quicksort, etc. Cada uno de ellos ha sido estudiado y analizada su eficiencia con detalle, ya que, como se dijo, si es necesario hacer repetidos ordenamientos, cuanto más eficiente sea el algoritmo que se utilice, mejor será el rendimiento general del programa.

En Programación I se vio como ordenar vectores, usando el algoritmo de ordenamiento por Selección. Este consiste en analizar cada elemento del vector, empezando por el que ocupa la posición 0 -al que se lo considera como el mínimo (o máximo) relativo-, contra el resto de los elementos; para el caso de que el ordenamiento sea ascendente, si existe algún valor menor dentro del vector se lo toma como el valor mínimo relativo, y se lo compara con los elementos restantes. Luego de recorrer todo el vector se obtiene el valor mínimo entre todos los existentes, y se intercambia ese valor con el que había en la posición 0. Se continúa el proceso, sin considerar el elemento ya examinado, tantas veces como elementos tenga el vector. Una función para ordenar un vector de enteros de 10 elementos podría ser como la siguiente:

```
void ordenar(int *v, int tam)
	{
	int i, j, posmin, aux;
	for(i=0;i,<tam-1;i++)
		{
		posmin=i;
		for(j=i+1;j<tam;j++)
			if(v[j]<v[posmin])
				posmin=j;
		aux=v[i];
```

```
            v[i]=v[posmin];
            v[posmin]=aux;
            }
    }
```

El for externo va hasta la posición anterior a la última, ya que ese elemento no tendría contra quien compararse. El for interno empieza desde una posición más adelante que la del for externo, ya que es innecesario comparar un valor contra sí mismo. Es decir, la primera comparación es entre el elemento 0 (i) y el elemento 1(j).

Para ordenar el vector de forma descendente (de mayor a menor) sólo hay que cambiar el condición del if (pasar de menor que a mayor que).

Matrices:
Las matrices pueden ser ordenadas –en caso que así lo requiera el programa- por filas o por columnas. Podemos entonces pensar que lo que debemos ordenar es un "conjunto de vectores" con lo cual el procedimiento sería, para cada uno de esos vectores, idéntico al anterior. Para poder pasar de un vector a otro, necesitamos otro for; luego para ordenar cada vector, o dicho correctamente cada fila o columna de la matriz, una de las dimensiones de la matriz debe permanecer fija:

- para ordenar las filas de una matriz, el for más externo debe ir desde 0 hasta el número de filas que tenga la matriz.
- cada fila se ordenará fijando el número de subíndice de la fila al valor del for más externo, y variando luego el número de subíndice de las columnas

- para ordenar las columnas de una matriz, el for más externo debe ir desde 0 hasta el número de columnas que tenga la matriz.
- cada columna se ordenará fijando el número de subíndice de la columna al valor del for más externo, y variando luego el número de subíndice de las filas.

Veamos las funciones
Ordenamiento por filas:

```
void ordenar_m_fila(int (*m)[5], int fila, int col)
        {
      int i,j,k,nc,aux;
      for(k=0;k<fila;k++)
            for(i=0;i<col-1;i++)
                  {
                  nc=i;
                  for(j=i+1;j<col;j++)
                        if(m[k][j]>m[k][nc])
                              nc=j;
                  aux=m[k][i];
                  m[k][i]=m[k][nc];
                  m[k][nc]=aux;
                  }
      }
```

Ordenamiento por columnas:

```
void ordenar_m_col(int (*m)[5], int fila, int col)
	{
	int i,j,k,nf,aux;
	for(k=0;k<col;k++)
		for(i=0;i<fila-1;i++)
			{
			nf=i;
			for(j=i+1;j<fila;j++)
				if(m[j][k]>m[nf][k])
					nf=j;
			aux=m[i][k];
			m[i][k]=m[nf][k];
			m[nf][k]=aux;
			}
	}
```

Ordenamiento de vectores de struct

Para el caso de los vectores de struct, la única diferencia con los vectores de tipos primitivos es que hay que elegir por cuál de los campos del vector de struct hacer el ordenamiento. Veamos un ejemplo:

```
struct articulo
	{
	char codart[5];
	float pu;
	};
```

Supongamos que se ha declarado el siguiente vector vart[10];
Como el vector tiene dos campos, puede ordenarse o por el string codart, o por el float pu.
Para ordenar por el precio unitario, ascendente, la función es:

```
void ordenar_pu(struct articulo *vart)
	{
	int i,j,posmin;
	struct articulo aux;
	for(i=0;i<9;i++)
		{
		posmin=i;
		for(j=i+1;j<10;j++)
			if(vart[j].pu<vart[posmin].pu)
				posmin=j;
		aux=vart[i];
		vart[i]=vart[posmin];
		vart[posmin]=aux;
		}
	}
```

Como se ve en el ejemplo, no es necesario hacer la asignación campo a campo. La variable auxiliar utilizada aux es de tipo struct articulo, y por lo tanto se le puede asignar un registro completo.

Para ordenar por codart

```
void ordenar_cod(struct articulo *vart)
	{
	int i,j,posmin;
	struct articulo aux;
	for(i=0;i<9;i++)
		{
		posmin=i;
		for(j=i+1;j<10;j++)
			if(strcmp(vart[j].codart,vart[posmin].codart)<0)
				posmin=j;
		aux=vart[i];
		vart[i]=vart[posmin];
		vart[posmin]=aux;
		}
	}
```

En este caso, como el campo de ordenamiento es un string, la comparación debe hacerse utilizando la función strcmp().