Unidad 5
Ordenación interna

- **Algoritmos de Ordenamiento por Intercambio.**
  - **Intercambio Directo (Burbuja).**
  - **ShellSort.**
  - **QuickSort.**
- **Algoritmos de Ordenamiento por Distribución.**
  - **Radix.**

Los algoritmos para el manejo de información se pueden clasificar como:

a) De Ordenamiento.

b) De Búsqueda.

Ambos procesos pueden clasificarse como externos o internos dependiendo del lugar en que se encuentre almacenada la información.

Los internos se llevan a cabo en memoria principal, los externos se realizan en memoria secundaria (diskettes, disco duro, cintas, memorias flash –USB-, etc.).

El proceso de ordenar representa la organización o reorganización de un conjunto de datos u objetos de acuerdo a una secuencia específica. Este proceso es importante cuando se requiere optimizar un proceso de búsqueda.

La operación de búsqueda es la que permite recuperar datos almacenados previamente.

Formalmente se define un ordenamiento de la siguiente manera:

Sea Vec un arreglo o lista de n elementos: Vec1, Vec2, Vec3, Vec4. . . VecN. Vec estará ordenado si depués de aplicarle un proceso logramos que:

a) Vec1 <= Vec2 <= Vec3 <= . . . . . .<= VecN (Ascendente)

b) Vec1 >= Vec2 >= Vec3 >= . . . . . .>= VecN (Descendente).

Un método de ordenación es estable si el orden relativo de elementos iguales permanece inalterado durante el proceso. La estabilidad es conveniente si los elementos ya se encontraban ordenados conforme a algún otro campo. Un método de ordenación es inestable si se altera el orden relativo de elementos iguales durante el proceso de ordenamiento.

Existen muchos algoritmos de ordenamiento, donde cada uno tiene sus méritos. Pero el criterio para juzgar un algoritmo se basa en las respuestas a las siguientes preguntas:

1) ¿Qué rapidez tiene el algoritmo en el caso medio ?
2) ¿Qué rapidez tiene en el caso mejor y en el caso peor ?
3) ¿El algoritmo exhibe un comportamiento natural o antinatural?
4) ¿Vuelve a colocar los elementos con claves o llaves iguales?

Algoritmos de Ordenamiento por Intercambio.

Ordenamiento por Intercambio Directo (BURBUJA).

Método muy utilizado por su fácil comprensión y programación pero considerado como el método más ineficiente.

La idea básica de este algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentren ordenados. Se realizan n-1 pasadas, transportando en cada una de las mismas el menor o mayor elemento (según sea el caso) a su posición real. Al final de las n-1 pasadas los elementos del arreglo estarán ordenados.

Algoritmo:

Burbuja(Vec, n) {Vec es un arreglo de **n** elementos}

{ i , j, aux son variables de tipo entero }

```
Repetir con i desde 2 hasta n
|   Repetir con j desde n hasta  i
|   |   Si Vec[j – 1]  >  Vec[j]  entonces
|   |   |           Hacer  aux← Vec[j – 1]
|   |   |                 Vec[j-1] ← Vec[j]
|   |   |                 Vec[j] ← aux
|   |   Fin_Si
|   Fin_Repetir
Fin_Repetir
```

Método de Ordenamiento Shell (ShellSort).

Este método es una versión mejorada del método de inserción directa, también se le conoce con el nombre de inserción con incrementos decrecientes.

Este método propone que las comparaciones entre los elementos se efectúen con saltos de mayor tamaño pero con incrementos decrecientes, así los elementos quedarán ordenados más rápidamente.

Pasos:

1. Los elementos del arreglo se dividirán en grupos de 8 elementos, teniendo en cuenta los elementos que se encuentran a 8 posiciones de distancia entre sí y los ordenará por separado.
2. Después del primer paso, se dividirán los elementos del arreglo en grupos de 4, teniendo en cuenta los elementos que se encuentran a 4 posiciones de distancia y los ordenará por separado.
3. Se dividen los elementos del arreglo en grupos de 2, tomando en cuenta los elementos que se encuentran a 2 posiciones de distancia entre sí y los ordenará por separado.
4. Finalmente se agruparán y ordenarán los elementos de manera normal, uno a uno.

Algoritmo:

```
Shell(Vec, n)    {Vec es un arreglo de n elementos}

{ i , j, aux son variables de tipo entero
  band es variable de tipo bool          }

   Hacer inc ← n + 1
   Repetir mientras(inc > 1)
      Hacer  inc ← parte entera (inc entre 2)
             band ← verdadero
      Repetir mientras ( band = verdadero)
             Hacer band = falso
                    i ← 1
             Repetir mientras ( (i + inc) <= n )
                    Si Vec[i] > Vec[ i + inc ] entonces
                          Hacer
                                 aux     ← Vec[i]
                                 Vec[i] ← Vec[i + inc]
                                                Vec[i + inc] ← aux
                                                band ← verdadero
                    Fin_Si
                   Hacer  i ← i + 1
             Fin_Repetir
      Fin_Repetir
   Fin_Repetir
```

Método de Ordenamiento QuickSort.

Inventado y nombrado por C.A. Hoare. Este método se basa en la ordenación por intercambio directo (burbuja).

El quicksort se basa en la idea de las particiones. El procedimiento general selecciona un valor llamado “comparando” (o pivote), y entonces se divide el arreglo en 2 partes, una con todos los elementos mayores o iguales del pivote y todos los elementos mayores que el valor de la partición en el otro.

La idea central del algoritmo consiste en :

1. Tomar un elemento X (pivote) de una posición del arreglo:

- Aleatorio
- Seleccionar el elemento medio del arreglo.

2. Se trata de ubicar a X en la posición correcta dentro del arreglo, de tal forma que los elementos a la izquierda sean menores o iguales a X, y todos los que se encuentran a la derecha sean mayores o iguales a X.
3. Se repiten los pasos pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de X en el arreglo.
4. El proceso termina cuando todos los elementos están ordenados

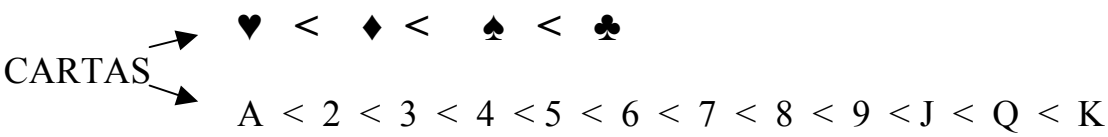
Versión del método QuickSort en Lenguaje C:

```
void quicksort(int *vec, int izq,int der)
{
 int i,j,pivote,aux;
  i = izq;
  j = der;
  pivote = vec[(izq+der)/2] ;
  do
  {
     while(vec[i] < pivote && i < der)
            i++;

     while(vec[j] > pivote && j > izq)
            j--;
     if(i <= j)
     {
            aux = vec[i];
            vec[i] = vec[j] ;
            vec[j] = aux;
            i++;
            j--;
     }
  }while (i <= j);
 if(izq < j)
    quicksort(vec,izq,j);
 if(i < der)
    quicksort(vec,i,der);

}
```

Algoritmos de Ordenamiento por Distribución.



Métodos de Ordenamiento por Distribución:

- Distribución Simple.
- Ordenamiento Radix.

Distribución Simple.

Este método se base en la técnica que utilizan los jugadores de cartas para ordenar mazos de cartas.

Algorítmicamente lo que se puede ordenar son conjuntos de datos. Para aplicar ésta técnica , se deben realizar las siguientes consideraciones:

1. Los datos que se utilicen deben ser enteros cuyos valores se encuentren en un rango específico:

   L.I. <= Vec[i] <= L.S.

2. Se utilizará un arreglo auxiliar (conjunto ó urnas)

   C[j] donde L.I. <= j <= L.S.

3. El resultado quedará ordenado en un arreglo auxiliar S, del mismo tamaño del arreglo Vec.

Los pasos para ordenar por el método de Distribución Simple, serán vistos en clases.

Método de Ordenamiento Radix.

Este método esta basado en los valores reales de los dígitos de acuerdo a la posición cardinal que ocupan los números que son ordenados.

123 →	1	2	3
	centena	decena	unidad

El ordenamiento Radix consiste en aplicar el método de ordenamiento por distribución simple, sobre porciones fijas de la llave, empezando por el dígito menos significativo. En clase veremos algunas alternativas de éste método.