1. Métodos de ordenación

La ordenación o clasificación de datos consiste en la disposición de los mismos de acuerdo con algún valor o característica. Por ejemplo, cada elemento de una agenda telefónica tiene un campo nombre, un campo dirección y un campo número telefónico. Por lo regular los datos en la agenda se encuentran organizados en un orden de la A la Z. De la misma forma un lista ó vector de datos se dice que esta ordenado de manera ascendente, si *X [i] <= X [i +1]* y, por otro lado, se dice que esta ordenado de manera descendente sí *X [i] >= X [i +1]*.

Los métodos de ordenamiento son muy variados y la elección del algoritmo adecuado debe realizarse con criterios de eficiencia (tiempo y ejecución) y en función de la memoria disponible. Dividiremos los métodos en dos grandes grupos:

- Directos (burbuja, selección e inserción).
- Avanzados (Shell sort, Merge sort, Quick sort, entre otros).

En el caso de listas pequeñas, los métodos directos se demuestran eficientes y relativamente eficientes, ya que la codificación del algoritmo correspondiente no es compleja. Su uso es muy frecuente. Sin embargo, en arreglos grandes las ordenaciones directas resultan ineficientes y se necesitara un método avanzado. Nosotros únicamente cubriremos los métodos de ordenación directos.

Método de la burbuja

Es uno de los métodos relativamente más sencillo e intuitivo, pero también resulta ser uno de los más ineficientes. Se basa en la ordenación por cambio, y recibe su nombre de la semejanza con las burbujas de un depósito de agua donde cada burbuja busca su propio nivel.

Los pasos a efectuar en el caso de una ordenación ascendente (en el caso de la ordenación descenderte solo habría que cambiar el signo de comparación) son:

1. Comparar el primer y segundo elemento, intercambiarlos si el primero es mayor que el segundo; luego se compara el primero con el tercero, intercambiándose en caso necesario, y el proceso se repite hasta llegar al último elemento. De este modo en la primera casilla quedara el elemento más pequeño de esa iteración.

2. Se repite el paso anterior, pero ahora con el segundo y tercero, en caso de ser necesario se intercambian, y así hasta llegar a comparar el segundo con el ultimo.

Consideremos el siguiente ejemplo. Se cuenta con un vector de 6 posiciones donde se inicia una lista de números { 7, 2, 8, 3, 5, 1 }, la cual será ordenada en forma ascendente { 1, 2, 3, 5, 7, 8 }, observe como se declara una variable constante entera llamada ***n*** la cual tiene un valor de 6, enseguida se declara un vector de tipo entero que contendrá una cantidad ***n*** de casillas, en este caso 6, declaramos también las variables ***i*** y ***j*** que nos ayudaran a desplazarnos entre casilla y casilla para hacer las comparaciones. Y finalmente la variable ***tem***, almacenara temporalmente el valor a intercambiar entre las casillas que lo necesiten.

El proceso sería de la siguiente manera:

1ra iteración, ***i*** permanece fijo en la casilla 0 y ***j*** se incrementa hasta llegar al último elemento:
{ **7**, **2**, 8, 3, 5, 1 } j = 1, intercambio generando { **2**, **7**, 8, 3, 5, 1 }, enseguida
{ **2**, 7, **8**, 3, 5, 1 } j = 2, no genera intercambio
{ **2**, 7, 8, **3**, 5, 1 } j = 3, no genera intercambio
{ **2**, 7, 8, 3, **5**, 1 } j = 4, no genera intercambio
{ **2**, 7, 8, 3, 5, **1** } j = 5, intercambio generando { **1**, 7, 8, 3, 5, **2** }, termina iteración

2da iteración, ***i*** permanece fijo en la casilla 1 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, **7**, **8**, 3, 5, 2 } j = 2, no genera intercambio
{ 1, **7**, 8, **3**, 5, 2 } j = 3, intercambio generando { 1, **3**, 8, **7**, 5, 2 }, enseguida
{ 1, **3**, 8, 7, **5**, 2 } j = 4, no genera intercambio
{ 1, **3**, 8, 7, 5, **2** } j = 5, intercambio generando { 1, **2**, 8, 7, 5, **3** }, termina iteración

3ra iteración, ***i*** permanece fijo en la casilla 2 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, **8**, **7**, 5, 3 } j = 3, intercambio generando { 1, 2, **7**, **8**, 5, 3 }
{ 1, 2, **7**, 8, **5**, 3 } j = 4, intercambio generando { 1, 2, **5**, 8, **7**, 3 }
{ 1, 2, **5**, 8, 7, **3** } j = 5, intercambio generando { 1, 2, **3**, 8, 7, **5** }, termina iteración

4ta iteración, ***i*** permanece fijo en la casilla 3 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, 3, **8**, **7**, 5 } j = 4, intercambio generando { 1, 2, 3, **7**, **8**, 5 }
{ 1, 2, 3, **7**, 8, **5** } j = 5, intercambio generando { 1, 2, 3, **5**, 8, **7** }, termina iteración

5ta iteración, ***i*** permanece fijo en la casilla 4 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, 3, 5, **8**, **7** } j = 5, intercambio generando { 1, 2, 3, 5, **7**, **8** }, termina proceso

```
#include <stdio.h>
#define n 6

int main() {
   int vector[n]={7,2,8,3,5,1};
   int i, j, tem;

   for (i = 0; i < n-1; i++) /* ordenamiento de burbuja */
      for (j = i+1; j < n; j++)
      {
         if ( vector[i] > vector[j] ) //realiza la comparación
         {  //si la comparación es cierta intercambia los datos
            tem = vector[i];
            vector[i] = vector[j];
            vector[j] = tem;
         }
      }
   /* veamos los elementos ordenados */
   for (i = 0; i < n; i++)
      printf(" %d ", vector[i]);
   getchar();
}
```

Método de selección

La idea básica es encontrar el elemento más pequeño (grande), en orden ascendente de la lista, e intercambiarlo con el elemento que ocupa la primer posición en la lista, a continuación se busca el siguiente elemento más pequeño y se transfiere a la segunda posición. Se repite el proceso hasta que el último elemento ha sido transferido a su posición correcta.

El algoritmo de ordenación depende a su vez del algoritmo necesario para localizar el componente mayor (menor) de un *array*. Es un proceso muy similar al método de la burbuja pero haciendo más eficiente la búsqueda y evitando intercambios innecesarios.

Consideremos el mismo arreglo del ejemplo anterior { 7, 2, 8, 3, 5, 1 }. El proceso sería de la siguiente manera:

1ra iteración, ***i*** permanece fijo en la casilla 0 y ***j*** se incrementa hasta llegar al último elemento:
{ **7**, **2**, 8, 3, 5, 1 } k = 0, j = 1, cambio k = j ya que en j esta el menor
{ 7, **2**, **8**, 3, 5, 1 } k = 1, j = 2, no genera cambio
{ 7, **2**, 8, **3**, 5, 1 } k = 1, j = 3, no genera cambio
{ 7, **2**, 8, 3, **5**, 1 } k = 1, j = 4, no genera cambio
{ 7, **2**, 8, 3, 5, **1** } k = 1, j = 5, cambio k = j ya que en j esta el menor, genera { **1**, 2, 8, 3, 5, **7** }

2da iteración, ***i*** permanece fijo en la casilla 1 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, **2**, **8**, **3**, **5**, **7** } k = 1, j = 2 - 5, no genera cambios

3ra iteración, ***i*** permanece fijo en la casilla 2 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, **8**, **3**, 5, 7 } k = 2, j = 3, cambio k = j ya que en j esta el menor
{ 1, 2, 8, **3**, **5**, 7 } k = 3, j = 4, no genera cambio
{ 1, 2, 8, **3**, 5, **7** } k = 3, j = 5, no genera cambio, termina el ciclo, genera { 1, 2, **3**, **8**, 5, 7 }

4ta iteración, ***i*** permanece fijo en la casilla 3 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, 3, **8**, **5**, 7 } k = 3, j = 4, cambio k = j ya que en j esta el menor
{ 1, 2, 3, 8, **5**, **7** } k = 4, j = 5, no genera cambio, termina el ciclo, genera { 1, 2, 3, **5**, **8**, 7 }

5ta iteración, ***i*** permanece fijo en la casilla 4 y ***j*** se incrementa hasta llegar al último elemento:
{ 1, 2, 3, 5, **8**, **7** } k = 4, j = 5, cambio k = j ya que en j esta el menor, genera { 1, 2, 3, 5, **7**, **8** }

```
#include <stdio.h>
#define n 6

int main(){
   int vector[n]={7,2,8,3,5,1};
   int i, j, k, tem;

   for (i = 0; i < n-1; i++) /* ordenamiento de selección */
   {
      k = i;
      for (j = i+1; j < n; j++)
         if ( vector[k] > vector[j] )
            k = j;
      tem = vector[i];
      vector[i] = vector[k];
      vector[k] = tem;
   }
   /* veamos los números ordenados */
   for (i = 0; i < n; i++)
```

```
      printf(" %d ", vector[i]);
   getchar();
}
```

Método de Inserción

Este método también se denomina "*método del jugador de cartas*", por la semejanza con la forma de clasificar las cartas de una baraja, insertando cada carta en el lugar adecuado.

El algoritmo ordena los dos primeros elementos de la lista, a continuación el tercer elemento se inserta en la posición que corresponda, el cuarto se inserta en la lista de tres elementos, y así sucesivamente. Este proceso continua hasta que la lista este totalmente ordenada.

Sea una lista A[1], A[2], ... A[n]. Los pasos a dar para una ordenación ascendente son:

- Ordenar A[1] y A[2].
- Comparar A[3] con A[2], si A[3] es mayor o igual a que A[2], sigue con el siguiente elemento si no se compara A[3] con A[1]; si A[3] es mayor o igual que A[1], insertar A[3] entre A[1] yA[2]. Si A[3] es menor que A[1], entonces transferir A[3] a A[1], A[1] a A[2] y A[2] a A[3].
- Se suponen ordenados los n-1 primeros elementos y corresponde insertar el n-ésimo elemento. Si A[m] es mayor que A[k] (con K = 1, 2, ..., m-1), se debe correr una posición A[k+1], ... A[m-1] y almacenar A[m] en la posición k+1.

Consideremos el mismo arreglo del ejemplo anterior { 7, 2, 8, 3, 5, 1 }. El proceso sería de la siguiente manera:

1ra iteración, ***i*** permanece fijo en la casilla 1 y ***j*** se decrementa mientras el elemento es menor a ***j***:
{ 7, **2**, 8, 3, 5, 1 } tem = **2**, j = 0, mientras j >= 0 y tem < **7**, { 7, **7**, 8, 3, 5, 1 }, j se decrementa en 1
{ 7, **7**, 8, 3, 5, 1 } tem = **2**, j = -1, mientras j >=0, rompe mientras, tem ingresa { **2**, **7**, 8, 3, 5, 1 }

2da iteración, ***i*** permanece fijo en la casilla 2 y ***j*** se decrementa mientras el elemento es menor a ***j***:
{ 2, **7**, 8, 3, 5, 1 } tem = **8**, j = 1, mientras j >= 0 y tem < **7**, rompe mientras { 2, 7, **8**, 3, 5, 1 }

3ra iteración, ***i*** permanece fijo en la casilla 3 y ***j*** se decrementa mientras el elemento es menor a ***j***:
{ 2, 7, 8, **3**, 5, 1 } tem = **3**, j = 2, mientras j >= 0 y tem < **8**, { 2, 7, 8, **8**, 5, 1 }, j se decrementa en 1
{ 2, 7, 8, **8**, 5, 1 } tem = **3**, j = 1, mientras j >= 0 y tem < **7**, { 2, 7, **7**, 8, 5, 1 }, j se decrementa en 1
{ 2, 7, **7**, 8, 5, 1 } tem = **3**, j = 0, mientras j >= 0 y tem < **2**, rompe, tem ingresa { 2, **3**, 7, 8, 5, 1 }

4ta iteración, ***i*** permanece fijo en la casilla 4 y ***j*** se decrementa mientras el elemento es menor a ***j***:
{ 2, 3, 7, 8, **5**, 1 } tem = **5**, j = 3, mientras j >= 0 y tem < **8**, { 2, 3, 7, 8, **8**, 1 }, j se decrementa en 1
{ 2, 3, 7, 8, **8**, 1 } tem = **5**, j = 2, mientras j >= 0 y tem < **7**, { 2, 3, 7, **7**, 8, 1 }, j se decrementa en 1
{ 2, 3, 7, **7**, 8, 1 } tem = **5**, j = 1, mientras j >= 0 y tem < **3**, rompe, tem ingresa { 2, 3, **5**, 7, 8, 1 }

5ta iteración, ***i*** permanece fijo en la casilla 5 y ***j*** se decrementa mientras el elemento es menor a ***j***:
{ 2, 3, 5, 7, 8, **1** } tem = **1**, j = 4, mientras j >= 0 y tem < **8**, { 2, 3, 5, 7, 8, **8** }, j se decrementa en 1
{ 2, 3, 5, 7, 8, **8** } tem = **1**, j = 3, mientras j >= 0 y tem < **7**, { 2, 3, 5, 7, **7**, 8 }, j se decrementa en 1
{ 2, 3, 5, 7, **7**, 8 } tem = **1**, j = 2, mientras j >= 0 y tem < **5**, { 2, 3, 5, **5**, 7, 8 }, j se decrementa en 1
{ 2, 3, 5, **5**, 7, 8 } tem = **1**, j = 1, mientras j >= 0 y tem < **3**, { 2, 3, **3**, 5, 7, 8 }, j se decrementa en 1
{ 2, 3, **3**, 5, 7, 8 } tem = **1**, j = 0, mientras j >= 0 y tem < **2**, { 2, **2**, 3, 5, 7, 8 }, j se decrementa en 1
{ 2, **2**, 3, 5, 7, 8 } tem = **1**, j = -1, mientras j >= 0, rompe mientras, tem ingresa { **1**, 2, 3, 5, 7, 8 }

```
#include <stdio.h>
#define n 6

int main(){
   int vector[n]={7,2,8,3,5,1};
   int i, j, tem;

   for (i = 1; i < n; i++) /* ordenamiento de inserción */
   {
      tem = vector[i];
      j = i - 1;
      while ( j >= 0 && tem < vector[j] )
      {
         vector[j+1] = vector[j];
         j--;
      }
      vector[j+1] = tem;
   }
   /* veamos los números ordenados */
   for (i = 0; i < n; i++)
      printf(" %d ", vector[i]);
   getchar();
}
```

2. Métodos de búsqueda

Se describirán las técnicas elementales que existen para buscar en una lista de datos un determinado elemento. Por ejemplo, se puede buscar en una lista de clientes, el nombre especifico de un cliente o en una lista de números encontrar alguno en especial. Existen 2 métodos en particular para buscar elementos en una lista:

- ✓ Búsqueda lineal o secuencial
- ✓ Búsqueda binaria

Búsqueda lineal

Es la técnica de búsqueda más simple para encontrar un elemento específico en una lista. Consiste un recorrer la lista elemento a elemento comenzando desde la primera posición de la lista, y así hasta finalizar con la lista. Este método se utiliza con las listas no ordenadas, y esa es su gran virtud, ya que en términos de eficiencia es muy lento. Si se desea conocer la posición que el elemento ocupa será en algunos casos necesario ocupar un variable extra para almacenarlo.

A continuación se presenta el código de la búsqueda lineal implementado con un ciclo while:

```
#include <stdio.h>
int main(){
   int const n = 5;
   int vector[n] = {3,5,8,1,4}, valor, i;
   printf("elemento a buscar ? ");
   scanf("%d", &valor);
   i=0;
   do {
      if ( valor == vector[i] )
            break;
```

```
      else
            i++;
   }while(i < n);
   if ( i == n )
      printf("no se encontró");
   else
      printf("si se encontró en %d", i);
   return 0;
}
```

Ahora veremos una versión un poco más corta o sencilla de entender con el ciclo for:

```
#include <stdio.h>
int main(){
   int const n = 5;
   int vector[n] = {3,5,8,1,4}, valor, i;
   printf("elemento a buscar ? ");
   scanf("%d", &valor);
   for ( i = 0; i < n; i++ )
      if ( valor == vector[i] )
            break;

   if ( i == n )
      printf("no se encontró");
   else
      printf("si se encontró en %d", i);
   return 0;
}
```

Búsqueda binaria

Dentro de los métodos de búsqueda de elementos, la búsqueda binaria es uno de los mas apropiados cuando hablamos de datos que ya se encuentran ordenados. Se basa en el fundamento de divide y vencerás para localizar el elemento deseado. Es un proceso muy similar al que una persona utiliza para buscar información en un entorno ordenado, por ejemplo en una agenda, si uno abre la agenda en la parte central, por ejemplo en la M, y la letra a buscar es la J, pues únicamente se desplaza al inicio de la agenda que es la dirección correcta.

El algoritmo de búsqueda binaria comienza en el elemento central de la lista. Si el elemento central no es buscado, entonces se repite la búsqueda, centrándose en la primera o segunda mitad, dependiendo de que el elemento buscado sea más pequeño o más grande que el valor central.

Supongamos una lista de 20 enteros ordenados en orden ascendente. Dos variables, primero y ultimo, contienen los valores de los índices inferior y superior del arreglo de la lista. Inicialmente, a primero se le asignará el valor 1 y al ultimo el valor 20. El recorrido del arreglo A se realizará desde A [primero] a A [ultimo] (A[1] a A[20]).

```
#include <stdio.h>
int main(){
   int const n = 20;
   int vector[n] = {1,2,3,4,5,6,7,8,9,10,11,
                    12,13,14,15,16,17,18,19,20};
   int encontrado = 0, valor, primero, ultimo, central;
   printf("elemento a buscar ? ");
   scanf("%d", &valor);
   primero = 0;
   ultimo =  n-1 ;
```

```
   while( !encontrado && primero <= ultimo ){
      central = (primero + ultimo) / 2;
      if ( valor == vector[ central ] )
            encontrado = 1;
      else if ( vector[ central ] > valor )
            ultimo = central -1;
      else
            primero = central +1;
   }
   if ( encontrado != 1 )
      printf("no se encontró");
   else
      printf("si se encontró en %d", central);
   system("pause");
   return 0;
}
```

Referencias

Si te interesa aprender más sobre esta unidad te recomiendo el libro: Cairó, Osvaldo. "*Estructura de Datos*". Editorial Mc.GrawHill, 3ª. Edición. ISBN: 9701059085. Año 2006.