INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA y ELÉCTRICA

UNIDAD CULHUACAN

INGENIERÍA EN COMPUTACIÓN

ASIGNATURA: ESTRUCTURA DE DATOS

Ordenamiento
Actividad 2

Suarez Vega Edgar Alan

Profesora: Clarissa Janeth González Acatitla

Semestre 2-2021

Febrero - Junio 2021

Indice

Introducción

En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación o reordenamiento de la entrada que satisfaga la relación de orden dada.

Ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren listas ordenadas para una ejecución rápida. También es útil para poner datos en forma canónica y para generar resultados legibles por humanos.

Desde los comienzos de la computación, el problema del ordenamiento ha atraído gran cantidad de investigación, tal vez debido a la complejidad de resolverlo eficientemente a pesar de su planteamiento simple y familiar. Los algoritmos de ordenamiento son comunes en las clases introductorias a la computación, donde la abundancia de algoritmos para el problema proporciona una gentil introducción a la variedad de conceptos núcleo de los algoritmos.

Los métodos de ordenación se clasifican en dos categorías

- **Ordenación interna (de arreglos) y**
- **Ordenación externa (de archivos).**

La ordenación interna o de arreglos, recibe este nombre ya que los elementos o componentes del arreglo se encuentran en la memoria principal de la computadora.

Los métodos de ordenación interna a su vez se clasifican en:

- **Métodos directos (n2) y**
- **Métodos logarítmicos (n * log n).**

Los métodos directos, son los más simples y fáciles de entender, son eficientes cuando se trata de una cantidad de datos pequeña. Los métodos logarítmicos, son más complejos, difíciles de entender y son eficientes en grandes cantidades de datos.

Los métodos directos más conocidos son:

- **Ordenación por intercambio.**
- **Ordenación por inserción.**
- **Ordenación por selección.**

Algoritmos de ordenamiento por intercambio.

La ordenación por intercambio consiste en comparar dos elementos del arreglo y determinar si existe un intercambio entre ellos, para esto debe definirse el tipo de ordenamiento que se quiere ya sea ascendente o descendente.

Los algoritmos de ordenación directa por intercambio que se analizaran son :

- **El método de la burbuja.**
- **El método quicksort.**
- **El método shaker sort o burbuja bidireccional.**

Método de Burbuja

Funcionamiento

El método de ordenación por intercambio directo o método de la burbuja, es el más simple y consiste en comparar dos elementos adyacentes para determinar si se realiza un intercambio entre los mismos, esto en caso de que el primero sea mayor que el segundo (forma ascendente) o el caso de que el primero sea menor que el segundo (forma descendente).

El primer procedimiento del método de la burbuja es:

1. Generar un ciclo que inicie desde uno hasta el número de elementos del arreglo.
2. Generar un segundo ciclo dentro del anterior que inicie desde cero hasta el número de elementos del arreglo menos dos.
3. Dentro del segundo ciclo debe existir una comparación que determina el tipo de ordenamiento (ascendente o descendente) entre el primer elemento (posición generado por el segundo ciclo) y el segundo elemento (el que le sigue), si la respuesta a la condición es verdadera se realiza un intercambio entre los dos elementos.
4. Para realizar el intercambio se genera un almacenamiento temporal, el cual guarda el dato del primer elemento, el segundo elemento toma el lugar del primero y en el lugar del segundo se coloca lo que contiene el almacenamiento temporal.

Una vez que los ciclos terminan la estructura debe quedar ordenada de forma ascendente o descendente, pero este procedimiento es considerado como el pero de los casos ya que si el número de elementos de la estructura es de 100, se tienen que realizar 9900 comparaciones entes de terminar la ejecución del método.

Un segundo procedimiento del método de la burbuja es:

1. Generar un ciclo que inicie desde cero hasta el número de elementos menos dos.
2. Generar un segundo ciclo desde el valor del ciclo anterior mas uno hasta el número de elementos menos uno;
3. Dentro del segundo ciclo debe existir una comparación que determina el tipo de ordenamiento (ascendente o descendente) entre el primer elemento (posición generada por el primer ciclo) y el segundo elemento (posición generada por el segundo ciclo), si la respuesta a la condición es verdadera se realiza un intercambio entre los dos elementos.
4. Para realizar el intercambio se genera un almacenamiento temporal, el cual guarda el dato del primer elemento, el segundo elemento toma el lugar del primero y en el lugar del segundo se coloca lo que contiene el almacenamiento temporal.

Una vez que los ciclos terminan la estructura debe quedar ordenada, la diferencia con el procedimiento anterior radica en el número de comparaciones y posibles intercambios que se presentan, en este segundo procedimiento, es menor ya que cada pasada que se le da al arreglo se realiza una comparación menos que en la pasada anterior.

Un tercer procedimiento del método de la burbuja es el siguiente:

1. Generar un ciclo que inicie desde uno hasta el número de elementos menos uno.
2. Generar un segundo ciclo que inicie desde el número de elementos menos uno y mientras que ese valor sea mayor o igual al del ciclo anterior (con decrementos).
3. Dentro del segundo ciclo debe existir una comparación que determina el tipo de ordenamiento (ascendente o descendente) entre el primer elemento (posición generada por el segundo ciclo) y el segundo elemento (posición generada por el segundo ciclo menos uno), si la respuesta a la condición es verdadera se realiza un intercambio entre los dos elementos.
4. Para realizar el intercambio se genera un almacenamiento temporal, el cual guarda el dato del primer elemento, el segundo elemento toma el lugar del primero y en el lugar del segundo se coloca lo que contiene el almacenamiento temporal

Este tercer procedimiento es muy similar al anterior con la diferencia que el elemento que va quedando es su lugar el primero no el último como en el caso anterior.

Ejemplo

Tomemos como ejemplo los números: "9 6 5 8 2 1", que serán ordenados de menor a mayor valor usando el método burbuja. Los elementos siguientes resaltados están siendo comparados.

Primera vuelta: (9 6 5 8 2 1) (6 9 5 8 2 1), el algoritmo compara los primeros dos elementos y los cambia porque 9 > 6 (6 9 5 8 2 1) (6 5 9 8 2 1) (6 5 9 8 2 1) (6 5 8 9 2 1) (6 5 8 9 2 1) (6 5 8 2 9 1) (6 5 8 2 9 1) (6 5 8 2 1 9)

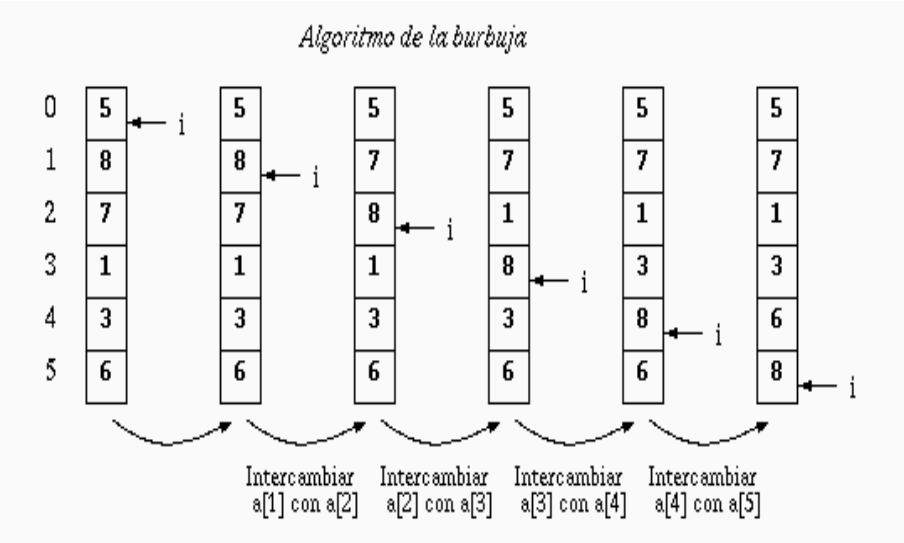
Segunda vuelta: (6 5 8 2 1 9) (5 6 8 2 1 9) (5 6 8 2 1 9) (5 6 8 2 1 9), como estos elementos ya están en orden, el algoritmo no hace cambios. (5 6 8 2 1 9) (5 6 2 8 1 9) (5 6 2 8 1 9) (5 6 2 1 8 9) (5 6 2 1 8 9) (5 6 2 1 8 9)

Tercera vuelta: (5 6 2 1 8 9) (5 6 2 1 8 9) (5 6 2 1 8 9) (5 2 6 1 8 9) (5 2 6 1 8 9) (5 2 1 6 8 9) (5 2 1 6 8 9) (5 2 1 6 8 9) (5 2 1 6 8 9) (5 2 1 6 8 9)

Cuarta vuelta: (5 2 1 6 8 9) (2 5 1 6 8 9) (2 5 1 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9) (2 1 5 6 8 9)

Quinta vuelta: (2 1 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9) (1 2 5 6 8 9)

otro ejemplo, este grafico



Código

```
#include<iostream>

using namespace std;

int main() {

   int num, aux;
   int comparaciones = 0;
   int intercambios = 0;
   int* arreglo;
   cout << "Cuantos numeros seran: ";
   cin >> num;
   arreglo = new int[num];
   cout << endl << "***CAPTURA DE NUMEROS***" << endl;

   for(int x = 0; x < num; x++) {
      cout << "Ingresa el numero " << x << " de la serie: ";
      cin >> arreglo[x];
      cout << endl;
   }
   cout << "***MUESTRA DE NUMEROS***" << endl;

   for(int y = 0; y < num; y++) {
      cout << "Numero " << y << ".- " << arreglo[y] << endl;
   }

   for(int z = 1; z < num; ++z) {
      for(int v = 0; v < (num - z); v++) {
         comparaciones++;
         if(arreglo[v] > arreglo[v+1]){
            aux = arreglo[v];
            arreglo[v] = arreglo[v + 1];
            arreglo[v + 1] = aux;
            intercambios++;
         }
      }
    }

   cout << "***NUMEROS ARREGLADOS***" << endl;

   for(int w = 0; w < num; w++) {
      cout << "Numero " << w << ".- " << arreglo[w] << endl;
   }

   cout << "Numero de comparaciones: " << comparaciones << endl;
   cout << "Numero de intercambios: " << intercambios << endl;

   delete[] arreglo;
   return 0;
}
```

Método quicksort

Funcionamiento

El *método de ordenamiento rápido* o *método quicksort*, es una técnica basada en otra conocida con el nombre *divide y vencerás*, que permite ordenar una cantidad de elementos en un tiempo proporcional a n^2 en el peor de los casos o a *n log n* en el mejor de los casos. El algoritmo original es recursivo, como la técnica en la que se basa.

La descripción del algoritmo para el método de ordenamiento quicksort es la siguiente:

1. Debe elegir uno de los elementos del arreglo al que llamaremos pivote.
2. Debe acomodar los elementos del arreglo a cada lado del pivote, de manera que del lado izquierdo queden todos los menores al pivote y del lado derecho los mayores al pivote; considere que en este momento, el pivote ocupa exactamente el lugar que le corresponderá en el arreglo ordenado.
3. Colocado el pivote en su lugar, el arreglo queda separado en dos subarreglos, uno formado por los elementos del lado izquierdo del pivote, y otro por los elementos del lado derecho del pivote.
4. Repetir este proceso de forma recursiva para cada subarreglo mientras éstos contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Para elegir un pivote se puede aplicar cualquiera de las siguientes tres opciones:

1. El pivote será el primer elemento del arreglo,
2. El pivote será el elemento que esta a la mitad del arreglo, o
3. Que el pivote se elija de entre tres elementos del arreglo (cualesquiera), los cuales se deben comparar para seleccionar el valor intermedio de los tres y considerarlo como el pivote.

La forma o técnica de re-acomodo de los elementos del lado izquierdo y derecho del pivote, aplica el siguiente procedimiento que es muy efectivo. Se utilizan dos índices: *izq*, al que llamaremos índice inicial, y *der*, al que llamaremos índice final. Conociendo estos elementos el algoritmo quedaría de la siguiente manera:

1. Recorrer el arreglo simultáneamente con *izq* y *der*: por la izquierda con *izq* (desde el primer elemento), y por la derecha con *der* (desde el último elemento).
2. Mientras el arreglo en su posición izq (arreglo[izq]) sea menor que el pivote, continuamos el movimiento a la derecha.
3. Mientras el arreglo en su posición der (arreglo[der]) sea mayor que el pivote, continuamos el movimiento a la izquierda.
4. Terminando los movimientos se compara los índices y si izq es menor o igual al der, se intercambian los elementos en esas posiciones y las posiciones se cambian izq a la derecha y der a la izquierda.

5. Repetir los pasos anteriores hasta que se crucen los índices (izq sea menor o igual a der).
6. El punto en que se cruzan los índices es la posición adecuada para colocar el pivote, porque sabemos que a un lado los elementos son todos menores y al otro son todos mayores (o habrían sido intercambiados).

Ejemplo

En el siguiente ejemplo se marcan el pivote y los índices *i* y *j* con las letras *p*, *i* y *j* respectivamente.

1. **Comenzamos con la lista completa. El elemento pivote será el 4:**

```
5 - 3 - 7 - 6 - 2 - 1 - 4
                        p
```

2. **Comparamos con el 5 por la izquierda y el 1 por la derecha.**

```
5 - 3 - 7 - 6 - 2 - 1 - 4
i                   j   p
```

3. **5 es mayor que 4 y 1 es menor. Intercambiamos:**

```
1 - 3 - 7 - 6 - 2 - 5 - 4
i                   j   p
```

4. **Avanzamos por la izquierda y la derecha:**

```
1 - 3 - 7 - 6 - 2 - 5 - 4
    i           j       p
```

5. **3 es menor que 4: avanzamos por la izquierda. 2 es menor que 4: nos mantenemos ahí.**

```
1 - 3 - 7 - 6 - 2 - 5 - 4
        i       j       p
```

6. **7 es mayor que 4 y 2 es menor: intercambiamos.**

```
1 - 3 - 2 - 6 - 7 - 5 - 4
        i       j       p
```

7. **Avanzamos por ambos lados:**

```
1 - 3 - 2 - 6 - 7 - 5 - 4
          iyj           p
```

8. **En este momento termina el ciclo principal, porque los índices se cruzaron. Ahora intercambiamos lista[i] con lista[p] (pasos 16-18):**

```
1 - 3 - 2 - 4 - 7 - 5 - 6
          p
```

9. **Aplicamos recursivamente a la sublista de la izquierda (índices 0 - 2). Tenemos lo siguiente:**

```
1 - 3 - 2
```

10. **1 es menor que 2: avanzamos por la izquierda. 3 es mayor: avanzamos por la derecha. Como se intercambiaron los índices termina el ciclo. Se intercambia lista[i] con lista[p]:**

```
1 - 2 - 3
```

11. **El mismo procedimiento se aplicará a la otra sublista. Al finalizar y unir todas las sublistas queda la lista inicial ordenada en forma ascendente.**

```
1 - 2 - 3 - 4 - 5 - 6 - 7
```

Código

```
// Función para dividir el array y hacer los intercambios
int divide(int *array, int start, int end) {
   int left;
   int right;
   int pivot;
   int temp;

   pivot = array[start];
   left = start;
   right = end;

   // Mientras no se cruzen los índices
   while (left < right) {
      while (array[right] > pivot) {
         right--;
      }

      while ((left < right) && (array[left] <= pivot)) {
         left++;
      }

      // Si todavía no se cruzan los indices seguimos intercambiando
      if (left < right) {
         temp = array[left];
         array[left] = array[right];
         array[right] = temp;
      }
   }

   // Los índices ya se han cruzado, ponemos el pivot en el lugar que le corresponde
   temp = array[right];
   array[right] = array[start];
   array[start] = temp;

   // La nueva posición del pivot
   return right;
}

// Función recursiva para hacer el ordenamiento
void quicksort(int *array, int start, int end)
{
   int pivot;

   if (start < end) {
      pivot = divide(array, start, end);

      // Ordeno la lista de los menores
      quicksort(array, start, pivot - 1);

      // Ordeno la lista de los mayores
      quicksort(array, pivot + 1, end);
   }
}
```

Método shaker sort o burbuja bidireccional

Funcionamiento

Surge como una mejora del algoritmo ordenamiento de burbuja.

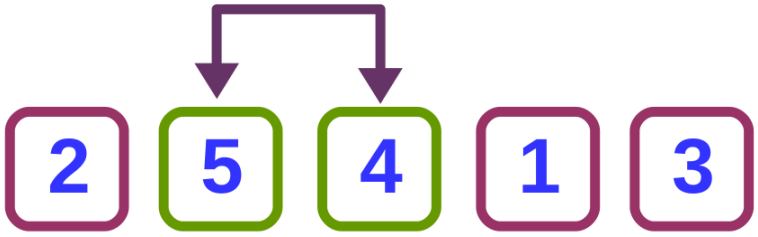
Ejemplo de la operativa paso a paso

La manera de trabajar de este algoritmo es ir ordenando al mismo tiempo por los dos extremos del vector. De manera que tras la primera iteración, tanto el menor como el mayor elemento estarán en sus posiciones finales. De esta manera se reduce el número de comparaciones aunque la complejidad del algoritmo sigue siendo $O(n^2)$

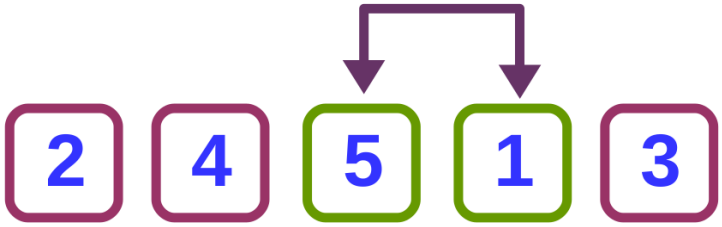
Comenzamos con una lista de elementos no ordenados

Tomamos los primeros dos números y si no están ordenados se intercambian los lugares

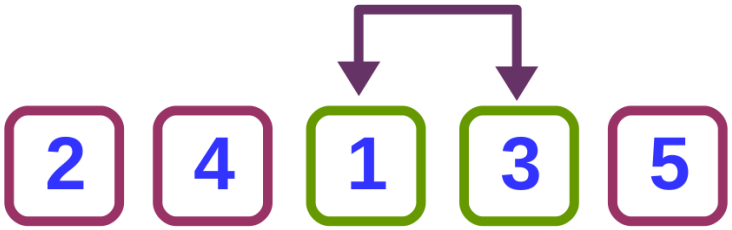
Se mueve un espacio hacia la derecha y se repite el proceso.

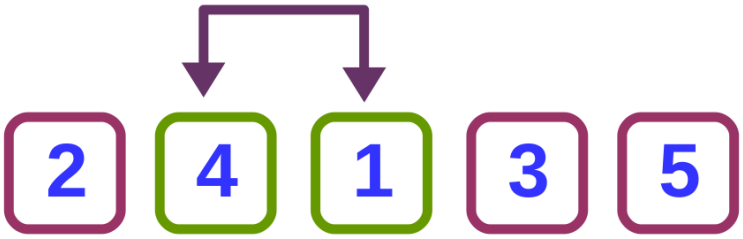


El proceso continua hasta llegar al final de la lista.

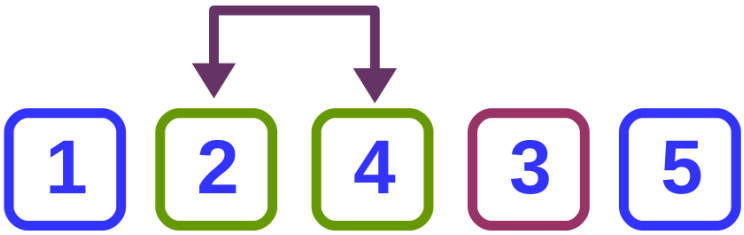


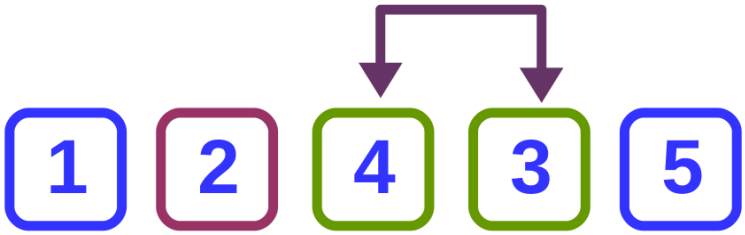
Al llegar al final a diferencia del ordenamiento de burbuja se repite el proceso en sentido inverso comenzando por el final de la lista hasta llegar al inicio.

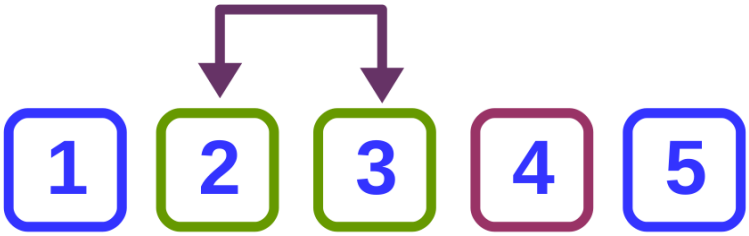




Al terminar el proceso el último número y el primero ya quedan ordenados por lo que en la siguiente iteración ya no se evalúan acortando el proceso.







Al finalizar la segunda iteración se marcan como ordenados el primer y último número comparado.

No es posible realizar más iteraciones ya que no quedan dos números sin ordenar para comparar, por lo tanto el algoritmo termina.

Código

```
#include<stdio.h>

#define N 5

int main(int argc, char** argv){

        int lista[N]={5,2,4,1,3};

        int inicioIndice=0;
        int finIndice=N-1;
        int i,aux,intercambio=0;

        //Recorrer la lista hasta que no haya números para ordenar
        while(inicioIndice<=finIndice){
                int nuevoPrimerIndice=finIndice,nuevoInicioIndice;
                int nuevoFinIndice=inicioIndice;

                //Ordenado hacia adelante
                for(i=inicioIndice;i<finIndice;i++){
                        //Comparando valores
                        if(*(lista+i)>*(lista+i+1)){
                                //Intercambio valores
                                aux=*(lista+i);
                                *(lista+i)=*(lista+i+1);
                                *(lista+i+1)=aux;
                                //Marcar que ha ocurrido un intercambio
                                intercambio=1;
                                //Marcar último número ordenado al final de la *(lista
                                nuevoFinIndice=i;
                        }
                }

                if(!intercambio){
                        //Si no se ha realizado ningún cambio salir de la iteracion
                        break;
                }

//Marcar el último número ordenado al final de la *(lista como final para no evaluarlo
                finIndice=nuevoFinIndice;
                intercambio=0;

                //Ordenado hacia atras
                for(i=finIndice;i>=inicioIndice;i--){
                        //Comparando valores
                        if(*(lista+i)>*(lista+i+1)){
                                //Intercambio valores
                                aux=*(lista+i);
                                *(lista+i)=*(lista+i+1);
                                *(lista+i+1)=aux;
                                //Marcar que ha ocurrido un intercambio
                                intercambio++;
                                //Marcar último número ordenado al principio de la *(lista
                                nuevoInicioIndice=i;
                        }
                }

                if(!intercambio){
                        //Si no se ha realizado ningún cambio salir de la iteracion
                        break;
                }

//Marcar el último número ordenado al principio de la *(lista como final para no evaluarlo la siguiente
iteración
                inicioIndice=nuevoInicioIndice+1;
        }

        return 0;
}
```

otra manera mas simplificada

```
void cocktail_sort(int* vector) {
    bool permutation;
    int actual=0, dirección=1;
    int comienzo=1, fin=19;
    do {
        permutation=false;
        while (((dirección==1) && (actual<fin)) || ((dirección==-1) && (actual>comienzo))) {
            actual += dirección;
            if (vector[actual]<vector[actual-1]) {
                int temp = vector[actual];
                vector[actual]=vector[actual-1];
                vector[actual-1]=temp;
                permutation=true;
            }
        }
        if (dirección==1) fin--; else comienzo++;
        dirección = -dirección;
    } while (permutation);
}
```

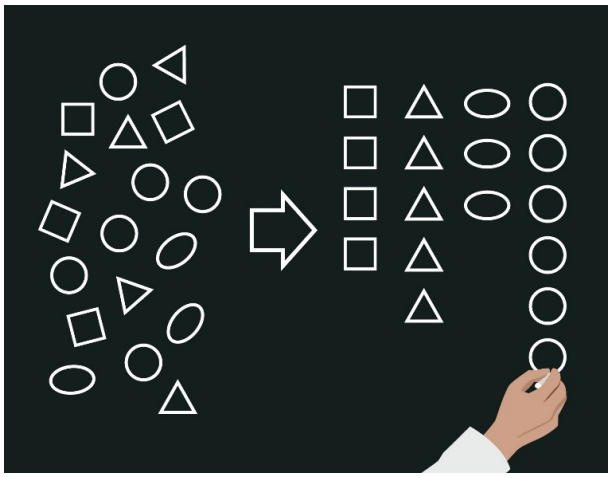
Conclusión

En conclusión desde los comienzos de la computación, el problema del ordenamiento ha atraído gran cantidad de investigación, tal vez debido a la complejidad de resolverlo eficientemente a pesar de su planteamiento simple y familiar. Por ejemplo, Bubble Sort fue analizado desde 1956.1 Aunque muchos puedan considerarlo un problema resuelto, nuevos y útiles algoritmos de ordenamiento se siguen inventado hasta el día de hoy (por ejemplo, el ordenamiento de biblioteca se publicó por primera vez en el 2004).

Hoy en día el ordenamiento esta mas presente que nunca y es de gran importancia como llevar estos métodos acabo de una manera acertada ya que se necesitan resolver problemas, automatizar procesos, etc, que con la ayuda de estos métodos de ordenamiento son realizados con éxito.

Como hoy en día la programación esta en todos lados gracias a el internet los datos y su organización de igual manera están en todos lados y los ocupamos día con día, de ahí su gran importancia en nuestro mundo cada vez mas digital.

Los métodos de ordenamiento siguen actualizándose y los ya existentes siguen vigentes, en la actualidad incluso con técnicas mas sofisticadas de inteligencia artificial se puede hacer ordenamiento, en conclusión estas herramientas o algoritmos, son ya parte del mundo digital y es importante conocerlos.



Bibliografías

https://sites.google.com/site/estructuradedatosmaac/home/metodos-de-ordenacion

https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja

https://es.wikipedia.org/wiki/Quicksort#Transici%C3%B3n_a_otro_algoritmo

https://github.com/ronnyml/C-Tutorial-Series/blob/master/Ordenamiento/quicksort.cpp

http://lwh.free.fr/pages/algo/tri/tri_shaker_es.html

https://juncotic.com/ordenamiento-de-burbuja-bidireccional-algoritmos-de-ordenamiento/

https://www.youtube.com/watch?v=BJhj7KrrPSg