

# Algoritmos de Ordenamiento

Yadira Yaneth Marroquín de la Peña

Matricula: 1701504

yaz\_marroquin@outlook.com

<https://github.com/yadira02/1701504MatComp>

1 de septiembre de 2017

En el siguiente documento observaremos los beneficios y las características de cada uno de los algoritmos de ordenación.

## 1. Selection Sort

El método de ordenación por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo.

### **SdeuCodigo**

```
para i=1 hasta n-1
    mínimo = i;
    para j=i+1 hasta n
        si lista[j] < lista[mínimo] entonces
            mínimo = j /* (!) */
    fin si
fin para
intercambiar(lista[i], lista[mínimo])
fin para
```

Al principio se tiene un elemento, que se considera un conjunto ordenado. Después, al haber k elementos ordenados de menor a mayor se coge el elemento k+1 y se va comparando con los elementos ya ordenados deteniéndose hasta que se encuentra un elemento mayor.

Como se puede ver en el código de la función, para buscar el máximo elemento en un segmento de lista se debe recorrer todo ese segmento, por lo que en nuestro caso debemos recorrer en el primer paso  $N$  elementos, en el segundo paso  $N - 1$  elementos, en el tercer paso  $N - 2$  elementos, etc. Cada visita a un elemento implica una cantidad constante y pequeña de comparaciones (que no depende de  $N$ ). Por lo tanto, tenemos que

$$T(N) = c * (2 + 3 + \dots + N) = c * N * (N + 1)/2 = N^2$$

### **Ventajas:**

- Fácil implementación.
- No requiere memoria adicional.
- Realiza pocos intercambios.
- Rendimiento constante: poca diferencia entre el peor y el mejor caso.
- No hay almacenamiento temporal

### **Desventajas:**

- Lento.
- Realiza numerosas comparaciones.
- Este método requiere  $n$  al cuadrado de número de pasos para ordenar  $n$  elementos.
- Además, su rendimiento es fácilmente influenciado por el orden inicial de los elementos antes del proceso de ordenamiento.
- El ordenamiento por selección sólo es apto para una lista de pocos elementos que estén en orden aleatorio.

## **2. Bubble Sort**

El método de la burbuja es uno de lo más simples, es tan fácil comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición.

Se le llama burbuja debido a que los valores más pequeños “burbujean” gradualmente(suben) hacia la cima o parte superior del array de modo similar a como suben las burbujas en el agua, los valores mayores se hunden en la parte inferior del array. El ordenamiento es uno de los procesos más comunes y útiles en el procesamiento de datos, es la clasificación u ordenación de los mismos.

**Características:** funciona comprando elementos de dos en dos en un ciclo, intercambiándolo según sea el caso ascendente o descendente.

Es necesario revisar varias veces toda lista hasta que no se necesiten más intercambios.

### SdeuCodigo

```
public void ordenamientoBurbuja(int a[], int n) {  
    EntradaSalida obj1= new EntradaSalida();  
  
    int aux=0;  
    for(int i=0;i<n-1;i++){  
        for(int j=0;j<n-1;j++){  
            if(a[j]>a[j+1]){  
                aux=a[j];  
                a[j]=a[j+1];  
                a[j+1]=aux;  
            }  
        }  
        System.out.println("\n Pasada : "+ i);  
        obj1.imprimirLista(a);  
    }  
}
```

#### Ventajas:

- Es bastante sencillo
- Es un código reducido.
- Realiza el ordenamiento.
- Eficaz

#### Desventajas:

- Consume bastante tiempo de computadora.
- Requiere muchas lecturas, escrituras en memoria.
- Su desventaja principal, es uno de los menos eficientes y por ello, normalmente, se aprende su técnica, pero no se utiliza.

## 3. Insertion Sort

El ordenamiento por inserción es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria.

La idea de este algoritmo de ordenación consiste en ir insertando un elemento de la lista o un arreglo en la parte ordenada de la misma, asumiendo que el primer elemento es la parte ordenada, el algoritmo ira comparando un elemento de la parte desordenada de la lista con los elementos de la parte ordenada.

Por ejemplo, otra forma de pensar acerca del ordenamiento. Imagina que estás jugando un juego de cartas. Tienes las cartas en tu mano y las cartas están ordenadas. Tomas exactamente una nueva carta del mazo. La tienes que colocar en el sitio correcto de manera que las cartas en tu mano sigan estando ordenadas.

### PSEUDOCÓDIGO

ALGORITMO INSERCIÓN

INICIO

ENTERO X, Z, AUX, ARREGLO[N]

LOGICO B

PARA(X ← 1, HASTA N, X ← X + 1)

{

AUX ← ARRAY[X]

Z ← X - 1

B ← FALSO

MIENTRAS(B = FALSO Y Z >= 0)

{

SI (AUX < ARREGLO[Z])

{

ARREGLO[Z + 1] ← ARREGLO[Z]

Z ← Z - 1

}

SI NO

{

B ← VERDAD

}

}

ARREGLO[Z + 1] ← AUX

}

FIN

### Ventajas:

- Fácil de implementar
- Requiere mínimo de memoria
- Rendimiento constante: poca diferencia entre el peor y el mejor caso.

### Desventajas:

- Lento
- Realiza numerosas comparaciones

### Características:

Requerimientos de Memoria: Al igual que el ordenamiento burbuja, este algoritmo sólo necesita una variable adicional para realizar los intercambios. Tiempo de Ejecución: El ciclo externo se ejecuta n veces para una lista de n elementos. Cada búsqueda requiere comparar todos los elementos no clasificados.

## 4. Quick Sort

El ordenamiento por partición (Quick Sort) se puede definir en una forma más conveniente como un procedimiento recursivo. Tiene aparentemente la propiedad de trabajar mejor para elementos de entrada desordenados completamente. Esta situación es precisamente la opuesta al ordenamiento de burbuja.

Este tipo de algoritmos se basa en la técnica "divide y vencerás", o sea es más rápido y fácil ordenar dos arreglos o listas de datos pequeños, que un arreglo o lista grande. El orden de complejidad del algoritmo es entonces de

$O(n^2)$ . El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. En el caso promedio, el orden es  $O(n \cdot \log n)$ .

### SdeuCodigo

```
procedure quicksort(iz, de: indice);
  var i, j : indice; x, w: item;
begin
  i = iz;
  j = de;
  x = a[(iz + de) div 2];
  repeat
    while (a[i].clave < x.clave) do i = i + 1;
    while (x.clave < a[j].clave) do j = j - 1;
    if (i <= j) then
      begin
        w = a[i];
        a[i] = a[j];
        a[j] = w;
        i = i + 1;
        j = j - 1;
      end
    until i > j;
    if iz < j then quicksort(iz, j);
    if i < de then quicksort(i, de);
  end;
```

### Características:

- Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores.
- En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sablistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.

## Conclusión

Observamos que cada uno de los algoritmos de ordenación es más útil, sencillo y rápido. El algoritmo que es el más fácil de usar es el de burbuja además de ser muy rápido. Porque su código es muy pequeño y puedes visualizar su procedimiento. Este algoritmo de ordenación nos sirve mucho porque puedes implementar ejercicios en búsqueda de ordenar una gran cantidad números. Para así conseguir que cada uno de los algoritmos lo forme más rápida que haciéndolo a mano.