

# Estructuras de Datos

## Clase 21 – Ordenamiento



Dr. Sergio A. Gómez  
<http://cs.uns.edu.ar/~sag>



Departamento de Ciencias e Ingeniería de la Computación  
Universidad Nacional del Sur  
Bahía Blanca, Argentina

## Definición del problema

Dado un arreglo  $a$  de  $n$  componentes enteras  $a[0], a[1], \dots, a[n-1]$  se desea obtener una permutación de  $a$  tal que  $a[0] \leq a[1] \leq \dots \leq a[n-2] \leq a[n-1]$ .

## Resumen de métodos de ordenamiento

- Multipasada:  $O(n^2)$ 
  - Selección (selection sort)
  - Burbuja (bubble sort) o intercambio (exchange sort)
  - Inserción (insertion sort)
- Merge sort:  $O(n \cdot \log_2(n))$
- Heap sort:  $O(n \cdot \log_2(n))$
- Quick sort:  $O(n^2)$  en peor caso pero se espera  $O(n \cdot \log_2(n))$  en promedio

Estructuras de datos - Dr. Sergio A. Gómez

3

## Ordenamiento por selección

- Algoritmo:

Haremos  $n-1$  pasadas sobre el arreglo  $a$

En la pasada  $i$ -ésima, encontramos el  $i$ -ésimo menor elemento del arreglo y lo intercambiamos con  $a[i]$

Pasada 1: intercambiar  $a[0]$  con  $a[3]$

Pasada 2: intercambiar  $a[1]$  con  $a[2]$

Pasada 3: intercambiar  $a[2]$  con  $a[3]$

Pasada 4: intercambiar  $a[3]$  con  $a[4]$

Pasada 4: 1 2 3 4 5

Estructuras de datos - Dr. Sergio A. Gómez

4

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019

```

public static void selectionsort( int [] a, int n ) {
    for( int i=0; i<n-1; i++ ) {
        // Hallar el mínimo a[p] de a[i], ..., a[n-1]
        p = i;
        for( int j=i+1; j<n; j++ )
            if( a[j] < a[p] ) p = j;

        // Intercambiar a[i] y a[p]
        int item = a[p];
        a[p] = a[i];
        a[i] = item;

        // Ahora a[0] <= ... <= a[i] y
        // a[i] <= a[j] cuando i < j < n
    }
}

```

$$T_{\text{selectionsort}}(n) = O(n^2)$$

## Ordenamiento por intercambio (burbuja)

- En cada pasada el burbuja mira dos elementos adyacentes y los intercambia si están fuera de orden.
- En cada pasada, el mayor elemento del arreglo queda al final del subarreglo que se está ordenando
- Son necesarias  $n-1$  pasadas (ya que el primer elemento queda ordenado gratis)

```

public static void bubblesort( int [] a, int n ) {
    for( int i=n-1; i>=0; i-- ) {
        // Burbujear el item más grande en a[0], ..., a[i] a a[i]
        for( int j=0; j<i; j++ )
            if( a[j] > a[j+1] ) {
                // Intercambiar items
                int item = a[j];
                a[j] = a[j+1];
                a[j+1] = item;
            }

        // Ahora a[j] <= ... <= a[n] y
        // a[j] <= a[i] cuando 0 <= j < i-1
    }
}

```

$T_{\text{bubblesort}}(n) = O(n^2)$

**Optimización:** Cuando nunca se entra al if en una iteración de i significa que el arreglo está ordenado.

La optimización consiste en ubicar una bandera en el if y setearla en true al entrar.

Si al terminar la iteración, esta bandera está en falso entonces el arreglo está ordenado y se puede terminar.

Estructuras de datos - Dr. Sergio A. Gómez

7

## Ordenamiento por inserción

- Es equivalente a la forma de ordenar un mazo de cartas.
- Se comienza con un mazo vacío y uno desordenado.
- Cada carta se trata de insertar en la posición correcta.
- Entonces, en un momento dado, una parte del mazo está ordenada y se trata de insertar la siguiente carta de la porción desordenada del mazo en la posición correcta.
- Terminamos cuando la porción desordenada está vacía.

Estructuras de datos - Dr. Sergio A. Gómez

8

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019

```

public static void insertionsort( int [] a, int n ) {
    for( int i = 1; i<n; i++ ) {
        // Insertar a[i] en la secuencia ordenada a[0], ..., a[i-1]
        int item = a[i]; // item a insertar
        int j = i;      // puntero de inserción
        boolean found = false;
        while( j>0 && !found )
            if( a[j-1] <= item ) // El item debería ser a[j]
                found = true;
            else {
                a[j] = a[j-1]; // Mover a[j-1] para arriba
                j--;
            }
            a[j] = item; // Insertar item
            // ahora a[0] <= ... <= a[i-1]
    }
}

```

$T_{\text{insertionsort}}(n) = O(n^2)$

Estructuras de datos - Dr. Sergio A. Gómez

9

## Merge sort: Ordenamiento por mezcla

- Caso recursivo: Partir el arreglo en dos, ordenar recursivamente cada mitad y luego hacer la mezcla de cada mitad ordenada en un gran arreglo ordenado.
- Caso base: El arreglo tiene 0 o 1 componentes entonces está ordenado.

10

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019

## Merge sort

```
public static void mergesort( int [] a, int n)
{ msort( a, 0, n-1); }

private static void msort( int [] a, int ini, int fin)
{
    if( ini < fin) {
        int medio = (ini + fin) / 2;
        msort(a, ini, medio );
        msort(a, medio + 1, fin);
        merge( a, ini, medio, fin );
        // merge hace la mezcla de los sub-arreglos en O(n)
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

11

Tamaño de la entrada:  $n$  = cantidad de componentes de  $a$

Recurrencia para  $n$ :

$$T(n) = \begin{cases} c_1 & \text{si } n = 1 \\ c_2 n + 2T(n/2) & \text{si } n > 1 \end{cases}$$

Estructuras de datos - Dr. Sergio A. Gómez

12

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019

$$\begin{aligned}
 T(n) &= c_2n + 2T(n/2) = c_2n + 2(c_2(n/2) + 2T(n/2/2)) \\
 &= 2c_2n + 4T(n/4) = 2c_2n + 4(c_2(n/4) + 2T(n/4/2)) \\
 &= 3c_2n + 8T(n/8) = \dots \\
 &= ic_2n + 2^iT(n/2^i)
 \end{aligned}$$

Termina cuando  $n/2^i = 1$ , es decir  $n=2^i$ .

Luego,  $i = \log_2(n)$ .

$$\begin{aligned}
 T(n) &= \log_2(n)c_2n + nT(1) = \log_2(n)c_2n + nc_1 \text{ es} \\
 &O(n\log_2(n))
 \end{aligned}$$

## Quick sort

El quick sort trabaja como merge sort pero evita hacer la mezcla

Algoritmo quick sort:

si la lista es no vacía entonces

- 1) dividir la lista en dos de tal manera que los ítems en la primera mitad vengan antes que los ítems en la segunda mitad
- 2) ordenar con quick sort la primera mitad
- 3) ordenar con quick sort la segunda mitad

Pregunta: ¿Cómo realizar la división en forma eficiente?

Supongamos nos dan una lista de 6 números.

17 36 22 41 12 24

Pongamos 24 en su lugar reacomodando la lista de tal manera que los primeros items sean menores a 24 y el resto mayores.

Guardemos 24 en un lugar seguro (una variable auxiliar) y definamos dos cursores *left* y *right* que indiquen dónde estamos:

17 36 22 41 12 \*\*

*left* *right*

Ahora movamos *left* hacia la derecha hasta descubrir un ítem mayor a 24:

17 36 22 41 12 \*\*

*left* *right*

Movamos este ítem al “agujero” donde estaba el 24 originalmente:

17 \*\* 22 41 12 36

*left* *right*

Movamos *right* a la izquierda hasta encontrar un ítem menor a 24:

17 \*\* 22 41 12 36

*left* *right*

Movamos este ítem al agujero arriba de *left*:

17 12 22 41 \*\* 36

*left* *right*

Hemos progresado, *left* y *right* están más cerca, movamos *left* a la derecha hasta hallar elemento mayor a 24:

17 12 22 41 \*\* 36

Estructuras de datos - Dr. Sergio A. Gómez *left* *right* 15

Hemos progresado, *left* y *right* están más cerca, movamos *left* a la derecha hasta hallar elemento mayor a 24:

17 12 22 41 \*\* 36

*left* *right*

Pongamos este ítem en la posición *left* sobre el marcador que indica *right*:

17 12 22 \*\* 41 36

*left* *right*

Movamos *right* a la izquierda, cuando los marcadores se encuentran en el agujero, pongamos allí al 24:

17 12 22 24 41 36

*left* *right*

Ahora podemos ordenar los items a la izquierda de 24 y los items a la derecha de 24 en forma recursiva usando este mismo proceso.

La componente donde está el 24 se llama “pivot” (o pivote en castellano).

Análisis de la Complejidad:

Quick sort en promedio corta el arreglo por la mitad, como no hace el merge, en promedio es más eficiente que merge sort con una complejidad de  $O(n \log_2(n))$ .

Sin embargo, el peor caso se da cuando se quiere ordenar un arreglo ordenado, se invoca recursivamente con una parte que mide 1 y la otra  $n-1$ , dando una complejidad de  $O(n^2)$ .

Estructuras de datos - Dr. Sergio A. Gómez 16

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019



## Heap Sort

- Objetivo: Ordenar un arreglo  $A$  de  $N$  enteros en forma ascendente
- Algoritmo HeapSort(  $a, n$  )
  - cola  $\leftarrow$  new ColaConPrioridad()
  - para  $i \leftarrow 0..n-1$  hacer
    - cola.insert(  $a[i]$  )
  - para  $i \leftarrow 0..n-1$  hacer
    - $a[i] \leftarrow$  cola.removeMin()
- Complejidad:  $T_{\text{heapsort}}(n) = O(n \log_2(n))$

Resolución de Problemas y Algoritmos - Dr. Sergio A. Gómez

17

## Heap sort in place

- En lugar de usar una cola con prioridades externa al arreglo  $a$ , se puede usar una porción del mismo arreglo  $a$  para implementar la cola con prioridades.
- $a$ 

Max heap de tamaño $i$	Porción de tamaño $n-i$ del arreglo no ordenada
------------------------	---

  - $i$
- Paso 1: para  $i=0$  hasta  $n-1$  insertar  $a[i]$  en la heap
- Paso 2: para  $i=n-1$  hasta  $0$  eliminar el máximo elemento de la heap y ubicarlo en  $a[i]$ .
- Complejidad:  $T_{\text{heapsortinplace}}(n) = O(n \log_2(n))$

Estructuras de datos - Dr. Sergio A. Gómez

18

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019