



Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 4

Curso 2022-2023

Marta N. Gómez

Análisis del Coste Temporal

- Suma de una Serie Aritmética.
- Algoritmos de Búsqueda.
- **Algoritmos de Ordenación:**
 - ♦ **Ordenación por selección.**
 - ♦ Ordenación por intercambio.
Método de la burbuja.
 - ♦ Ordenación por inserción.
 - ♦ Ordenación rápida.




Se basa en **realizar varias pasadas** e ir **localizando el elemento** que **hay que reubicar** en su **posición correcta**.

Ejemplo, se quiere ordenar de menor a mayor y se **localiza el menor** de los valores: **25**

[0]	[1]	[2]	[3]	[4]	[5]	[6]
62	31	25	87	48	52	77

Se **intercambia** con valor de la **primera posición**:

[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	62	87	48	52	77




Se recorre la sublista de elementos desde la siguiente posición al elemento ubicado correctamente, buscando el **valor menor: 31**

[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	62	87	48	52	77

Se **intercambia** con valor de la **segunda posición**:

[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	62	87	48	52	77



Se procede de igual manera hasta llegar al final de la lista.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	62	87	48	52	77



[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	48	87	62	52	77



[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	48	52	62	87	77



[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	48	52	62	87	77



[0]	[1]	[2]	[3]	[4]	[5]	[6]
25	31	48	52	62	77	87

```
void ordenacionSeleccion (string & f)
{
    int  posmenor;
    char menor;

    for (unsigned int i{0}; i < f.size(); i++)
    {
        posmenor = i;
        menor = f.at(posmenor);
        for (unsigned int j{i+1}; j < f.size(); j++)
        {
            if (f.at(j) < menor)
            {
                posmenor = j;
                menor = f.at(posmenor);
            }
        }
        f.at(posmenor) = f.at(i);
        f.at(i) = menor;
    }
}
```

Análisis en el mejor caso

```

void ordenacionSeleccion (string & f)
{
    int  posmenor;
    char menor;

    for (unsigned int i{0}; i < f.size(); i++)
    {
        posmenor = i;
        menor = f.at(posmenor);
        for (unsigned int j{i+1}; j < f.size(); j++)
        {
            if (f.at(j) < menor)
            {
                posmenor = j;
                menor = f.at(posmenor);
            }
        }
        f.at(posmenor) = f.at(i);
        f.at(i) = menor;
    }
}

```

Diagram illustrating the complexity analysis of the selection sort algorithm in the best case:

- The inner loop (for j) is marked with $\leftarrow O(1)$.
- The inner loop is grouped by a bracket labeled $O(n)$.
- The swap operation (for i) is marked with $\leftarrow O(1)$.
- The entire algorithm is grouped by a large bracket labeled $O(n^2)$.

$$(n-1) + (n-2) + \dots + 2 + 1 = (1 + (n-1)) n/2$$

Luego: $T(n)$ es $O(n^2)$

Análisis en el peor caso

```

void ordenacionSeleccion (string & f)
{
    int  posmenor;
    char menor;

    for (unsigned int i{0}; i < f.size(); i++)
    {
        posmenor = i;
        menor = f.at(posmenor);
        for (unsigned int j{i+1}; j < f.size(); j++)
        {
            if (f.at(j) < menor)
            {
                posmenor = j;
                menor = f.at(posmenor);
            }
        }
        f.at(posmenor) = f.at(i);
        f.at(i) = menor;
    }
}

```

$\left. \begin{array}{l} \left. \begin{array}{l} \left. \begin{array}{l} \text{posmenor} = i; \\ \text{menor} = f.at(\text{posmenor}); \end{array} \right\} \leftarrow O(1) \\ \left. \begin{array}{l} \text{for (unsigned int j\{i+1\}; j < f.size(); j++)} \\ \left\{ \begin{array}{l} \text{if (f.at(j) < menor)} \\ \left\{ \begin{array}{l} \text{posmenor} = j; \\ \text{menor} = f.at(\text{posmenor}); \end{array} \right\} \end{array} \right\} \leftarrow O(1) \end{array} \right\} O(n) \end{array} \right\} O(n^2)$

$\left. \begin{array}{l} \left. \begin{array}{l} \left. \begin{array}{l} \text{f.at(posmenor)} = \text{f.at(i)}; \\ \text{f.at(i)} = \text{menor}; \end{array} \right\} \leftarrow O(1) \end{array} \right\} \end{array} \right\}$

Igual que antes, luego: $T(n)$ es $O(n^2)$

Análisis del Coste Temporal

- Suma de una Serie Aritmética.
- Algoritmos de Búsqueda.
- **Algoritmos de Ordenación:**
 - ♦ Ordenación por selección.
 - ♦ **Ordenación por intercambio.**
Método de la burbuja.
 - ♦ Ordenación por inserción.
 - ♦ Ordenación rápida.



Ordenación por intercambio - Método de la burbuja

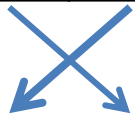
Se basa en el **intercambio sistemático de elementos no ordenados** hasta lograr que la lista esté ordenada.

Ejemplo, se quiere ordenar de menor a mayor y se **localiza el menor de los valores**.

62	31	25	87	48	52	77
----	----	----	----	----	----	----

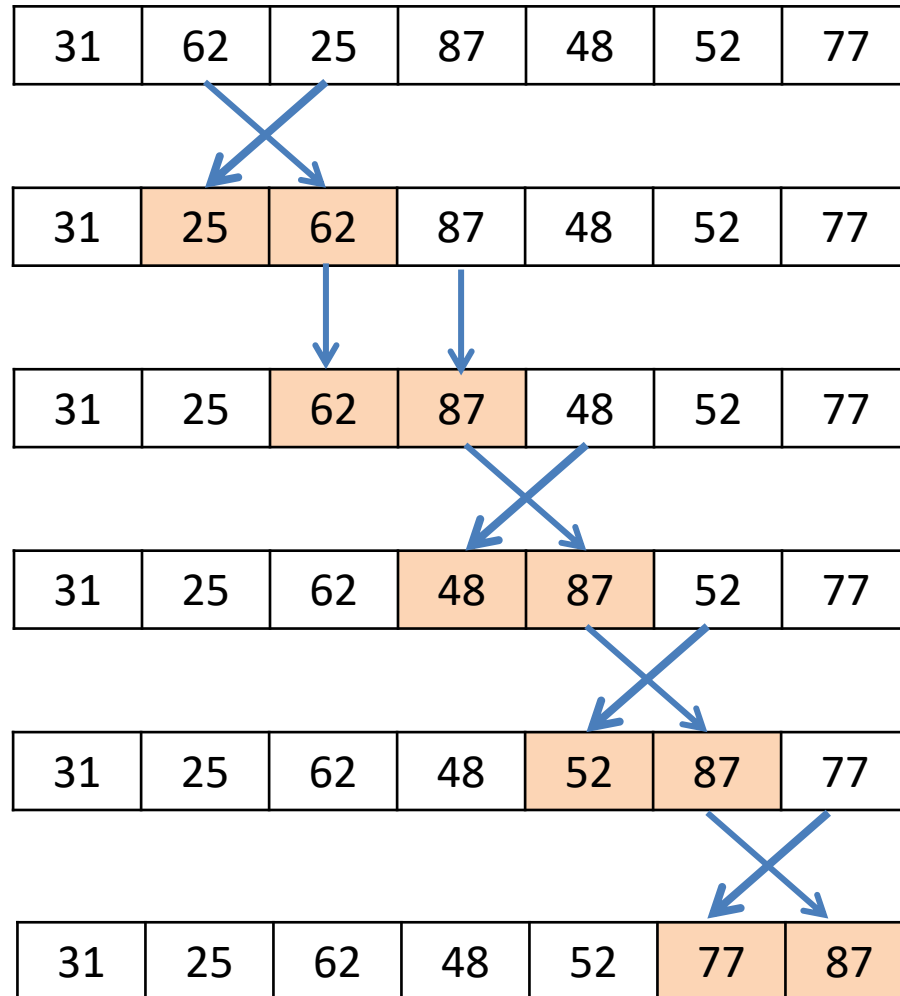
Se comparan los dos primeros y se intercambian porque no están ordenados:

62	31	25	87	48	52	77
31	62	25	87	48	52	77



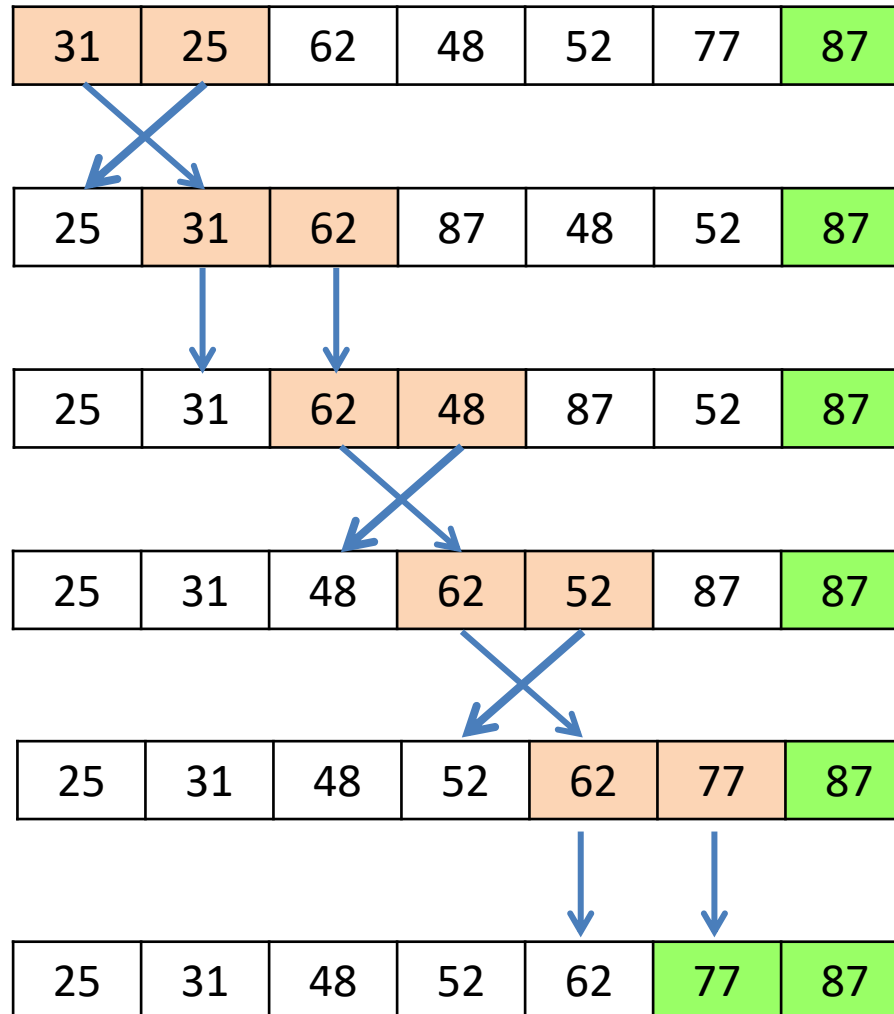
Ordenación por intercambio- Método de la burbuja

Procedemos igual con el 2do. y 3ro.. **Repetimos el proceso hasta completar el vector (1era. vuelta):**



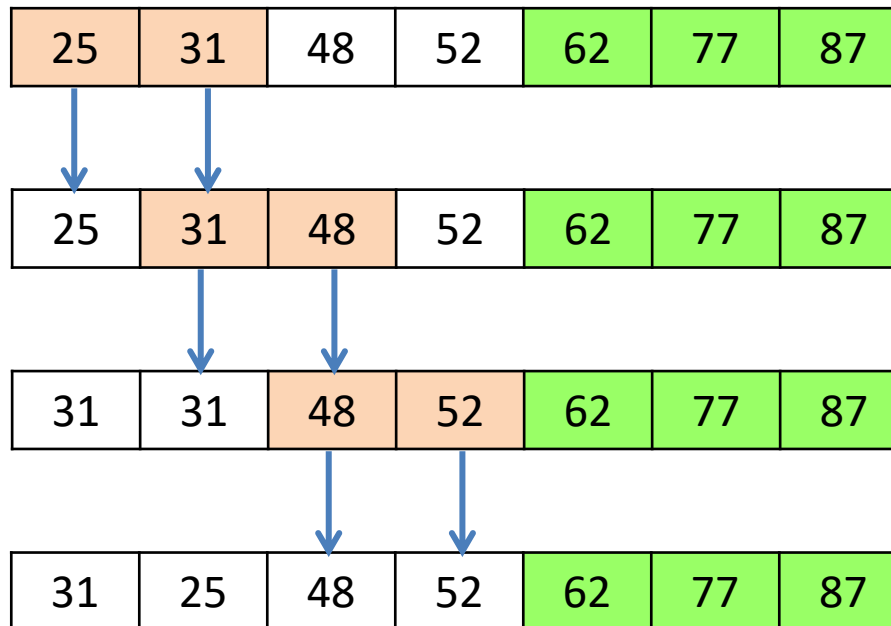
Ordenación por intercambio- Método de la burbuja

Empezamos la **2da. vuelta** dejando fuera el **último elemento** que está en su **posición correcta**:



Ordenación por intercambio- Método de la burbuja

Procedemos hasta el final:



Ordenación por intercambio- Método de la burbuja

```
void metodoBurbuja (string & f)
{
    char menor;

    for (unsigned int compar{0}; compar < f.size()-1; compar++)
    {
        for (unsigned int i{0}; i < f.size()-1-compar; i++)
        {
            if (f.at(i) > f.at(i+1))
            {
                menor = f.at(i);
                f.at(i) = f.at(i+1);
                f.at(i+1)=menor;
            }
        }
    }
}
```

Ordenación por intercambio- Método de la burbuja

Análisis en el peor caso

```
void metodoBurbuja (string & f)
{
    char menor;

    for (unsigned int compar{0}; compar < f.size()-1; compar++)
    {
        for (unsigned int i{0}; i < f.size()-1-compar; i++)
        {
            if (f.at(i) > f.at(i+1))
            {
                menor = f.at(i);
                f.at(i) = f.at(i+1);
                f.at(i+1)=menor;
            }
        }
    }
}
```

Diagram illustrating the complexity analysis of the Bubble Sort algorithm in the worst case:

- The innermost loop (swapping elements) is labeled $O(1)$.
- The middle loop (iterating over the array) is labeled $O(n)$.
- The outer loop (iterating over the array) is labeled $O(n^2)$.

Luego: $T(n)$ es $O(n^2)$

Análisis del Coste Temporal

- Suma de una Serie Aritmética.
- Algoritmos de Búsqueda.
- **Algoritmos de Ordenación:**
 - ♦ Ordenación por selección.
 - ♦ Ordenación por intercambio.
Método de la burbuja.
 - ♦ **Ordenación por inserción.**
 - ♦ Ordenación rápida.



Ordenación por inserción

Método preferido de los jugadores de cartas.

Se basa en insertar un nuevo elemento en una lista que ya está ordenada manteniendo el orden.

62	31	25	87	48	52	77
31	62	25	87	48	52	77
25	31	62	87	48	52	77
25	31	62	87	48	52	77
25	31	48	62	87	52	77
25	31	48	52	62	87	77
25	31	48	52	62	77	87

- Se va tomando un elemento de la **parte no ordenada** para **colocarlo en su lugar en la parte ordenada**.
- **El primer elemento se considera ordenado** (la lista inicial consta de un elemento).
- A continuación, se procede a insertar el segundo elemento en la posición correcta (delante o detrás de primero) dependiendo de que sea menor o mayor.
- Se **repite esta operación sucesivamente** de tal modo que se va colocando cada elemento en la posición correcta.
- El proceso se **repetirá $n-1$ veces**.

```
void ordenarInsercion(vector<int> & v) {  
    int temp;  
    unsigned int i;  
  
    for (unsigned int e=1; e < v.size(); e++){  
        temp=v.at(e);  
        i=0;  
        while (v.at(i)<temp) {  
            i++;  
        }  
        if (i < e)  
        {  
            for (unsigned int k=e ; k>i; k--) {  
                v.at(k) = v.at(k-1);  
            }  
            v.at(i) = temp;  
        }  
    }  
}
```

```
void ordenarInsercion(vector<int> & v) {  
    int temp;  
    unsigned int i;  
  
    for (unsigned int e=1; e < v.size(); e++){  
        temp=v.at(e);  
        i=0;  
        while (v.at(i)<temp) {  
            i++;  
        }  
        if (i < e)  
        {  
            for (unsigned int k=e ; k>i; k--) {  
                v.at(k) = v.at(k-1);  
            }  
            v.at(i) = temp;  
        }  
    }  
}
```

Luego: $T(n)$ es $O(n)$

```
void ordenarInsercion(vector<int> & v) {  
    int temp;  
    unsigned int i;  
  
    for (unsigned int e=1; e < v.size(); e++){  
        temp=v.at(e);  
        i=0;  
        while (v.at(i)<temp) {  
            i++;  
        }  
        if (i < e)  
        {  
            for (unsigned int k=e ; k>i; k--) {  
                v.at(k) = v.at(k-1);  
            }  
            v.at(i) = temp;  
        }  
    }  
}
```

Luego: $T(n)$ es $O(n^2)$

Análisis del Coste Temporal

- Suma de una Serie Aritmética.
- Algoritmos de Búsqueda.
- **Algoritmos de Ordenación:**
 - ♦ Ordenación por selección.
 - ♦ Ordenación por intercambio.
Método de la burbuja.
 - ♦ Ordenación por inserción.
 - ♦ **Ordenación rápida (quicksort).**



Ordenación rápida (Quicksort)

- El algoritmo consiste en **dividir el vector** que se desea ordenar en **dos bloques**. En el primero se sitúan todos los elementos que son **menores** que un valor que se toma como referencia (**pivote**) y en segundo bloque el resto.
- Este procedimiento se repite dividiendo a su vez cada uno de estos bloques y repitiendo la operación anteriormente descrita.
- La condición de parada se da cuando el bloque que se desea ordenar está formado por un **único elemento** (bloque ordenado).
- El resultado se obtiene de la combinación de todos los resultados parciales.
- El esquema seguido por este algoritmo es el de “***divide y venceras***”.

Ordenación rápida (Quicksort)

50	60	20	30	40	10
----	----	----	----	----	----

Pivote = $(0+5)/2=2$

10	60	20	30	40	50
----	----	----	----	----	----

10	20	60	30	40	50
----	----	----	----	----	----

1era. ejecución

10	20	60	30	40	50
----	----	----	----	----	----

Pivote = $(2+5)/2=3$

30	60	40	50
----	----	----	----

2da. ejecución

30	60	40	50
----	----	----	----

Pivote = $(3+5)/2=4$

40	60	50
----	----	----

3era. ejecución

40	60	50
----	----	----

Pivote = $(4+5)/2=4$

50	60
----	----

4ta. ejecución

Ordenación rápida (Quicksort)

```
void ordenQuickSort(vector<int> &v, int izq, int der)
{
    int i{izq}, d{der}, pivote, aux;

    pivote = v.at((i+d)/2);

    while (i < d)
    {
        while (v.at(i) < pivote) { i++; }

        while (pivote < v.at(d)) { d--; }

        if (i <= d) // intercambio de elementos
        {
            aux = v.at(i);
            v.at(i) = v.at(d);
            v.at(d) = aux;
            i++; d--; // se ajustan las posiciones
        }
    }

    if (izq < d) { ordenQuickSort(v, izq, d); }
    if (i < der) { ordenQuickSort(v, i, der); }
}
```

Análisis del mejor caso: Se produce cuando el pivote divide al vector en **dos partes iguales** y el orden de los elementos es aleatorio:

$$T(n) = 2T\left(\frac{n}{2}\right) + n - 1$$

$T(n/2)$ es el coste de ordenar una de las mitades y n el número de comparaciones realizadas.

Si desarrollamos la recurrencia se obtiene:

$$T(n) = n \log n + n - 1$$

Que es de orden $T(n) \in O(n \log n)$

Análisis del mejor caso: Supongamos que el tamaño del vector es una potencia de 2: $n = 2^k \rightarrow \log n = k$

Recorrido	Comparaciones
1º	(n-1) comparaciones * 1 vector 2 vectores de tamaño n/2
2º	(n/2) comparaciones * 2 vector = n 4 vectores de tamaño n/4
3º	(n/4) comparaciones * 4 vector = n 8 vectores de tamaño n/8
...	...
k-ésimo	(n/2 ^{k-1}) comparaciones * 2 ^{k-1} vector = n 2 ^k vectores de tamaño n/2 ^k

$$n + n + \dots + n = kn = n \log n$$

Luego es de orden $T(n) \in O(n \log n)$

Análisis del peor caso: Si se elige como pivote el **primer elemento del vector** (elemento menor) y además se considera que el vector está **ordenado** entonces, el bucle *while* se ejecutará en total:

$$(n - 1) + (n - 2) + (n - 3) + \dots + 1$$

Cada miembro de este sumando proviene de cada una de las sucesivas **ordenaciones recursivas**. Este sumatorio da lugar a la siguiente expresión:

$$(n - 1) + (n - 2) + (n - 3) + \dots + 1 = \sum_{i=1}^{n-1} (n - i) = \frac{[(n-1)+1](n-1)}{2} = \frac{n(n-1)}{2}$$

Luego es de orden cuadrático **$T(n) \in O(n^2)$**