



UNIVERSIDAD DE ALMERÍA

Grado en Ingeniería Informática

Introducción a la Programación

2021-2022



Tema 4. Búsqueda y ordenación

- Arrays de objetos
- Operaciones avanzadas sobre arrays
- Búsqueda
- Ordenación

Tema 4. Búsqueda y ordenación

Arrays de objetos

- ☐ Arrays unidimensionales (vectores) de objetos
- ☐ Arrays bidimensionales (matrices) de objetos

Tema 4. Búsqueda y ordenación

Arrays unidimensionales de objetos

- Declaración, creación e inicialización de un array de objetos

Ejemplo

```
Fraccion [] fracciones;
```

Declarar

```
fracciones = new Fraccion[3];
```

Crear

```
fracciones[0] = new Fraccion(8, 9);
```

```
fracciones[1] = new Fraccion(11, -22);
```

```
fracciones[2] = new Fraccion(1, 3);
```

Inicializar

En una sola línea:

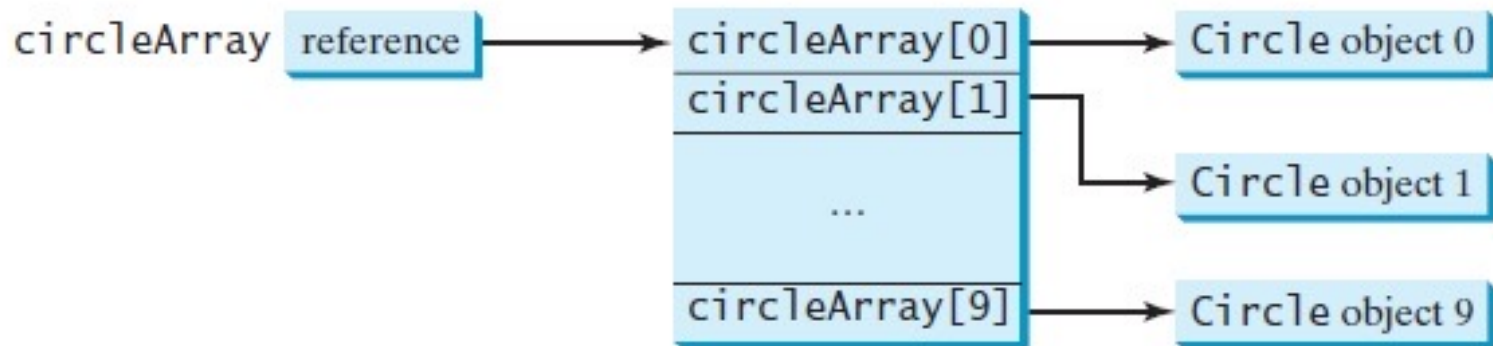
```
Fraccion [] fracciones = {new Fraccion(8, 9),  
                           new Fraccion(11, -22), new Fraccion(1, 3)};
```

Tema 4. Búsqueda y ordenación

➡ Arrays unidimensionales de objetos

□ Representación de un array de objetos

Ejemplo de array de Círculos (`circleArray`)



Tema 4. Búsqueda y ordenación

Arrays unidimensionales

□ Declaración, creación e inicialización

Array de objetos

- **Declarar**

```
Fraccion [] fracciones;
```

- **Crear el array:**

```
fracciones = new Fraccion[3];
```

- **Inicializar** el array: crear instancias

```
fracciones[0] = new Fraccion(8, 9);  
fracciones[1] = new Fraccion(11, -22);  
fracciones[2] = new Fraccion(1, 3);
```

- **Acceso a un elemento:**

```
fracciones[i] -> referencia inst Fraccion
```

Array tipo básico (int)

```
int [] enteros;
```

```
enteros = new int[3];
```

```
enteros[i] -> valor int
```

Arrays unidimensionales

Array de objetos

- Declarar

`Fraccion [] fracciones;`

- Crear el array:

`fracciones = new Fraccion[3];`

- Inicializar el array: crear instancias

`fracciones[0] = new Fraccion(8, 9);`
`fracciones[1] = new Fraccion(11, -22);`
`fracciones[2] = new Fraccion(1, 3);`

- Acceso a un elemento:

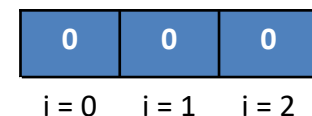
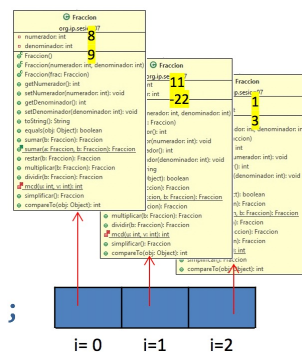
`fracciones[i] -> referencia inst Fraccion`

Array tipo básico (int)

`int [] enteros;`

`enteros = new int[3];`

`enteros[i] -> valor int`



Tema 4. Búsqueda y ordenación

Arrays unidimensionales de objetos

Ejemplo. Declaración, creación e inicialización de un array de objetos

```
1 package org.ip.tema05;
2
3 import org.ip.sesion07.Fraccion;
4
5 public class OtroTestArrayFracciones {
6
7     public static void main(String[] args) {
8
9         Fraccion [] arrayFracciones = new Fraccion[5];
10
11         for (int i = 0; i < arrayFracciones.length; i++)
12             arrayFracciones[i] = new Fraccion(i + 1, i + 2);
13
14         for (int i = 0; i < arrayFracciones.length; i++)
15             System.out.println(arrayFracciones[i].toString());
16     }
17
18 }
```

Declaración y creación

Inicialización

Mostrar contenido

Tema 4. Búsqueda y ordenación

Arrays unidimensionales de objetos

Ejemplo. Acceso a instancias del array y uso de métodos

```
1 package org.ip.tema05;
2
3 import org.ip.sesion07.Fraccion;
4
5 public class OtroTestArrayFracciones {
6
7     public static void main(String[] args) {
8
9         Fraccion [] arrayFracciones = new Fraccion[5];
10
11         for (int i = 0; i < arrayFracciones.length; i++)
12             arrayFracciones[i] = new Fraccion(i + 1, i + 2);
13
14         for (int i = 0; i < arrayFracciones.length; i++)
15             System.out.println(arrayFracciones[i].toString());
16
17         int denominador = arrayFracciones[2].getDenominador();
18         System.out.println("Denominado = " + denominador);
19
20         Fraccion fraccionSuma = new Fraccion(0, 1);
21         for (int i = 0; i < arrayFracciones.length; i++) {
22             arrayFracciones[i].simplificar();
23             fraccionSuma = fraccionSuma.sumar(arrayFracciones[i]);
24         }
25         System.out.println("Fraccion suma = " + (fraccionSuma.simplificar()).toString());
26     }
27 }
```

Acceso a una instancia del array y llamar a un método para obtener el denominador

Llamar al método `sumar()` para sumar todas las fracciones del array

Tema 4. Búsqueda y ordenación

Salida

```

1 package org.ip.tema05;
2 import org.ip.sesion07.Fraccion;
3 public class TestArrayFracciones {
4
5     public static void main(String[] args) {
6         Fraccion [] arrayFracciones = new Fraccion[5];
7         arrayFracciones[0] = new Fraccion(8, 9);
8         arrayFracciones[1] = new Fraccion(11, -22);
9         arrayFracciones[2] = new Fraccion(3, 9);
10        arrayFracciones[3] = new Fraccion(3, 9).simplificar();
11        arrayFracciones[4] = new Fraccion(1, 2);
12
13        Fraccion fraccionSuma = new Fraccion(0, 1);
14        Fraccion fraccionProducto = new Fraccion(1, 1);
15        for (int i = 0; i < arrayFracciones.length; i++) {
16            arrayFracciones[i].simplificar();
17            System.out.println("ArrayFracciones[" + i + "] = " + arrayFracciones[i].toString());
18            fraccionSuma = fraccionSuma.sumar(arrayFracciones[i]);
19            fraccionProducto = fraccionProducto.multiplicar(arrayFracciones[i]);
20        }
21
22        System.out.println("Fraccion suma = " + (fraccionSuma.simplificar()).toString());
23        System.out.println("Fraccion producto = " + (fraccionProducto.simplificar()).toString());
24
25        if (arrayFracciones[2].equals(arrayFracciones[3]))
26            System.out.println("Las fracciones: " + arrayFracciones[2] + " y " + arrayFracciones[3] + " son iguales");
27        else
28            System.out.println("Las fracciones: " + arrayFracciones[2] + " y " + arrayFracciones[3] + " NO son iguales");
29
30        if (arrayFracciones[3].compareTo(arrayFracciones[4]) == 1)
31            System.out.println("La fraccion: " + arrayFracciones[3] + " es mayor que la fraccion " + arrayFracciones[4]);
32        else if (arrayFracciones[3].compareTo(arrayFracciones[4]) == -1)
33            System.out.println("La fraccion: " + arrayFracciones[3] + " es menor que la fraccion " + arrayFracciones[4]);
34        else
35            System.out.println("Las fracciones: " + arrayFracciones[3] + " y " + arrayFracciones[4] + " son iguales");
36    }
37 }

```

```

ArrayFracciones[0] = 8/9
ArrayFracciones[1] = -1/2
ArrayFracciones[2] = 1/3
ArrayFracciones[3] = 1/3
ArrayFracciones[4] = 1/2
Fraccion suma = 14/9
Fraccion producto = -2/81
Las fracciones: 1/3 y 1/3 son iguales
La fraccion: 1/3 es menor que la fraccion 1/2

```

Tema 4. Búsqueda y ordenación

Arrays unidimensionales de objetos

□ Ejemplo: Copiar un array de objetos

```
Fraccion [] copiaArrayFracciones(Fraccion [] fracOrigen) {
```

1. Declarar array destino

```
    Fraccion [] fracDestino;
```

2. Crear el array: debe ser del mismo tamaño que el array origen

```
    fracDestino = new Fraccion[fracOrigen.length];
```

3. Inicializar el array destino: crear nuevas instancias iguales a las de array origen

```
    for (i = 0; i < fracOrigen.length; i++) {
```

```
        fracDestino[i] = new Fraccion(fracOrigen[i]);
```

```
    }
```

```
    return fracDestino;
```

```
}
```

Invocamos al constructor de copia

Tema 4. Búsqueda y ordenación

Arrays unidimensionales de objetos

❑ Ejemplo: Copiar un array de objetos

```
Fraccion [] copiaArrayFracciones(Fraccion [] fracOrigen) {
```

1. Declarar array destino

```
    Fraccion [] fracDestino;
```

2. Crear el array: debe ser del mismo tamaño que el array origen

```
    fracDestino = new Fraccion[fracOrigen.length];
```

3. Inicializar el array destino: crear nuevas instancias iguales a las de array origen

```
    for (i = 0; i < fracOrigen.length; i++) {  
        int numerador = fracOrigen[i].getNumerador();  
        int denominador = fracOrigen[i].getDenominador();  
        fracDestino[i] = new Fraccion(numerador, denominador);  
    }
```

```
}
```

```
    return fracDestino;
```

```
}
```

Invocamos al constructor normal

Tema 4. Búsqueda y ordenación

Arrays bidimensionales (matriz) de objetos

❑ Declaración, creación e inicialización de una matriz de objetos

```
Complejo [][] complejos;
```

Declarar

```
complejos = new Complejo[2][2];
```

Crear

```
complejos[0][0] = new Complejo(2.5, 5.5);
```

```
complejos[0][1] = new Complejo(-3.5, 1.0);
```

```
complejos[1][0] = new Complejo(1.5, 7.5);
```

```
complejos[1][1] = new Complejo(1.0, 5.0);
```

Inicializar

En una sola línea:

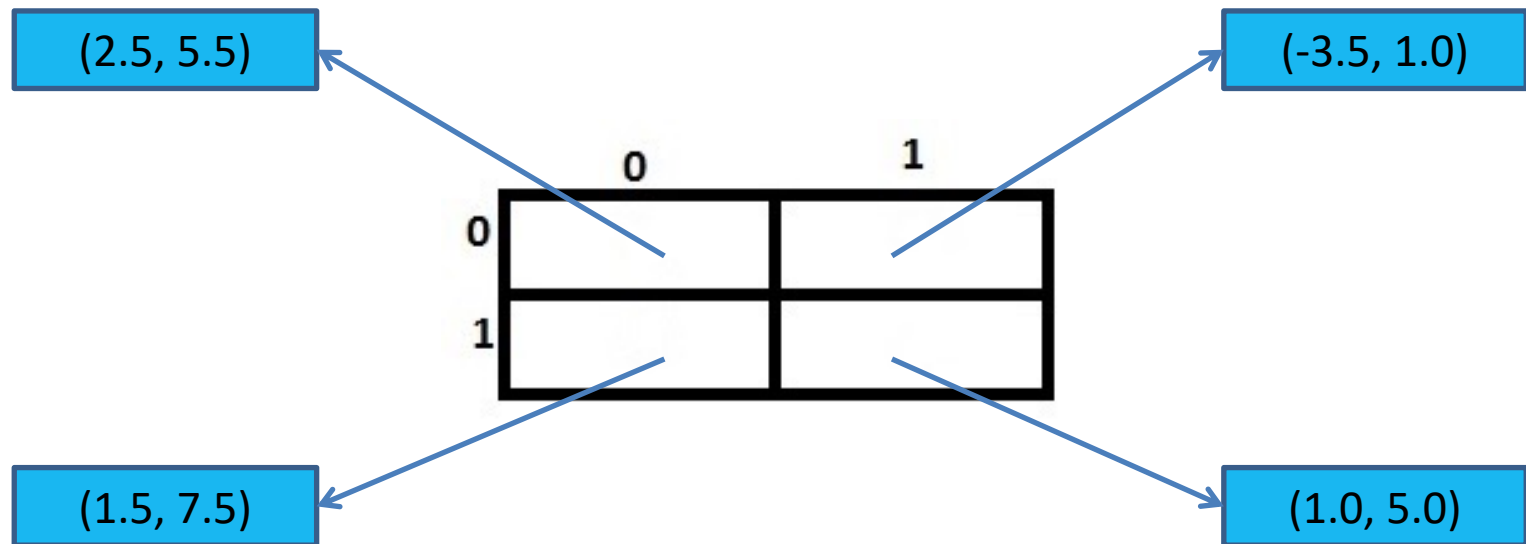
```
Complejo [][] complejos = {{new Complejo(2.5, 5.5), new Complejo(-3.5,  
1.0)}, {new Complejo(1.5, 7.5), new Complejo(1.0, 5.0)}};
```

Tema 4. Búsqueda y ordenación

➡ Arrays bidimensionales (matriz) de objetos

□ Representación de una matriz de objetos

Ejemplo de una matriz de Complejos



Tema 4. Búsqueda y ordenación

➡ Arrays bidimensionales (matriz) de objetos

```
1 package org.ip.tema05;
```

```
2  
3 import org.ip.sesion07.Complejo;
```

```
4  
5 public class OtroTestMatrizComplejos {
```

```
6  
7     public static void main(String[] args) {
```

```
8  
9         Complejo [][] matrizComplejos = new Complejo[3][3];
```

Declaración y creación

```
10  
11         for (int i = 0; i < matrizComplejos.length; i++) {  
12             for (int j = 0; j < matrizComplejos[i].length; j++) {  
13                 matrizComplejos[i][j] = new Complejo(i + 1, i + 2);  
14             }  
15         }
```

Inicialización de la matriz

```
16  
17         for (int i = 0; i < matrizComplejos.length; i++) {  
18             for (int j = 0; j < matrizComplejos[i].length; j++) {  
19                 if (j != matrizComplejos[i].length - 1)  
20                     System.out.print("MatrizComplejos[" + i + "][" + j + "] = " + matrizComplejos[i][j].toString() + ", ");  
21                 else  
22                     System.out.print("MatrizComplejos[" + i + "][" + j + "] = " + matrizComplejos[i][j].toString());  
23             }  
24             System.out.println();  
25         }
```

Mostrar contenido

```
26  
27         System.out.println(matrizComplejos[1][2].conjugado().toString());  
28     }  
29 }
```

Acceso a una instancia de la matriz y llamar a un método para obtener el conjugado

Tema 4. Búsqueda y ordenación

```

1 package org.ip.tema05;
2 import org.ip.sesion07.Complejo;
3 public class TestMatrizComplejos {
4
5     public static void main(String[] args) {
6         Complejo [][] matrizComplejos = new Complejo[2][2];
7         matrizComplejos[0][0] = new Complejo(2.5, 5.5);
8         matrizComplejos[0][1] = new Complejo(-3.5, 1.0);
9         matrizComplejos[1][0] = new Complejo(1.5, 7.5);
10        matrizComplejos[1][1] = new Complejo(1.0, 5.0);
11
12        Complejo complejoSuma = new Complejo(0, 0);
13        Complejo complejoProducto = new Complejo(1, 0);
14        for (int i = 0; i < matrizComplejos.length; i++) {
15            for (int j = 0; j < matrizComplejos[i].length; j++) {
16                if (j != matrizComplejos[i].length - 1)
17                    System.out.print("MatrizComplejos[" + i + "][" + j + "] = " + matrizComplejos[i][j].toString() + ", ");
18                else
19                    System.out.print("MatrizComplejos[" + i + "][" + j + "] = " + matrizComplejos[i][j].toString());
20                complejoSuma = complejoSuma.sumar(matrizComplejos[i][j]);
21                complejoProducto = complejoProducto.multiplicar(matrizComplejos[i][j]);
22            }
23            System.out.println();
24        }
25        System.out.println("Complejo suma de toda la matriz = " + complejoSuma.toString());
26        System.out.println("Modulo del complejo suma: " + complejoSuma.modulo());
27        System.out.println("Argumento del complejo suma: " + complejoSuma.argumento());
28        System.out.println("Complejo producto de toda la matriz = " + complejoProducto.toString());
29    }
30 }

```

Salida

```

MatrizComplejos[0][0] = 2.5 + 5.5i, MatrizComplejos[0][1] = -3.5 + 1.0i
MatrizComplejos[1][0] = 1.5 + 7.5i, MatrizComplejos[1][1] = 1.0 + 5.0i
Complejo suma de toda la matriz = 1.5 + 19.0i
Modulo del complejo suma: 19.059118552545918
Argumento del complejo suma: 1.4920123658057527
Complejo producto de toda la matriz = 764.25 + 389.25i

```


Tema 4. Búsqueda y ordenación

Operaciones avanzadas sobre arrays

- ☐ Operaciones avanzadas sobre arrays unidimensionales
- ☐ Operaciones avanzadas sobre matrices

Tema 4. Búsqueda y ordenación

Arrays unidimensionales

Ejemplo. Calcular la media de los valores en un subarray temperaturas[inferior..superior]

```
public static double media(int [] temperaturas, int inferior, int superior) {  
    double suma = 0.0;  
    int elementos = superior - inferior + 1;  
    if (inferior < 0 || superior >= temperaturas.length || inferior > superior)  
        throw new RuntimeException ("Indices del subarray fuera de rango");  
    for (int i = inferior; i <= superior; i++){  
        suma += temperaturas[i] ;  
    }  
    return suma / elementos;  
}
```

```

1 package org.ip.tema05;
2 import java.util.Scanner;
3 public class ArrayConExcepciones {
4     public static double media(int [] array, int inferior, int superior) {
5         if (inferior < 0 || superior >= array.length || inferior > superior)
6             throw new RuntimeException("Indices de array, fuera de rango");
7         double suma = 0.0;
8         int componentes = superior - inferior + 1;
9         for (int i = inferior; i <= superior; i++) {
10             suma += (double)array[i];
11         }
12         return suma / componentes;
13     }
14
15     public static void main(String[] args) {
16         int [] temperaturas = {5, 10, 25, -4, 3};
17         Scanner entrada = new Scanner(System.in);
18         int inferior = 0, superior = 0;
19         double valorMedio = 0.0;
20         boolean indicesCorrectos = false;
21         while (!indicesCorrectos) {
22             try {
23                 System.out.println("Introduce indices inferior y superior"
24                     + "para calcular la media del subarray");
25                 inferior = entrada.nextInt();
26                 superior = entrada.nextInt();
27                 valorMedio = media(temperaturas, inferior, superior);
28                 indicesCorrectos = true;
29             } catch (RuntimeException e) {
30                 System.out.println(e.getMessage());
31             }
32         }
33         System.out.println("La media del subarray es: " + valorMedio);
34     }
35 }

```

Salida

```

Introduce indices inferior y superiorpara calcular la media del subarray
5 0
Indices de array, fuera de rango
Introduce indices inferior y superiorpara calcular la media del subarray
7 9
Indices de array, fuera de rango
Introduce indices inferior y superiorpara calcular la media del subarray
0 3
La media del subarray es: 9.0

```

Tema 4. Búsqueda y ordenación

Arrays bidimensionales

Ejemplo. Aleatorizar el contenido de una matriz (**random suffling**).

```
for (int i = 0; i < matriz.length; i++) {  
    for (int j = 0; j < matriz[i].length; j++){  
        int i1 = (int)(Math.random() * matriz.length);  
        int j1 = (int)(Math.random() * matriz[i].length);  
        double temp = matriz[i][j] ;  
        matriz[i][j] = matriz[i1][j1];  
        matriz[i1][j1] = temp;  
    }  
}
```

Tema 4. Búsqueda y ordenación

Ejemplo. Método de clase para comprobar si una matriz es simétrica.

```
public static boolean esSimetrica(int [][] matriz) {  
    boolean simetria = true;  
    int fila = 0; int columna;  
    while (simetria && (fila < matriz.length)) {  
        columna = 0;  
        while (simetria && (columna < matriz[fila].length)) {  
            if (matriz[fila][columna] != matriz[columna][fila])  
                simetria = false;  
            columna++;  
        }  
        fila++;  
    }  
    return simetria;  
}
```

Salida

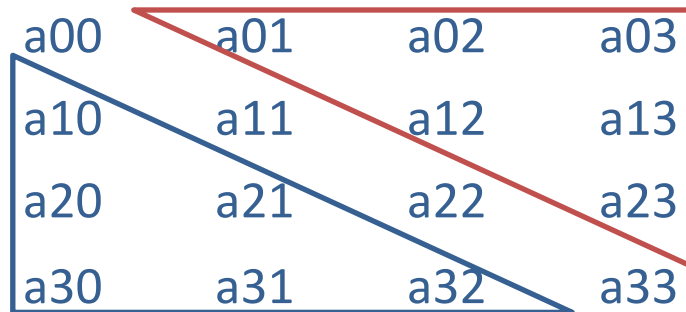
La matriz es simétrica

```
1 package org.ip.tema05;
2
3 public class MatrizSimetrica {
4
5     public static boolean esSimetrica(int [][] matriz) {
6         boolean simetrica = true;
7         int fila = 0;
8         int columna;
9         while (simetrica && (fila < matriz.length)) {
10             columna = 0;
11             while (simetrica && (columna < matriz[fila].length)) {
12                 if (matriz[fila][columna] != matriz[columna][fila])
13                     simetrica = false;
14                 columna++;
15             }
16             fila++;
17         }
18         return simetrica;
19     }
20     public static void main(String[] args) {
21         int [][] matriz = {{1, 2, 3}, {2, 1, 4}, {3, 4, 1}};
22         boolean simetrica = esSimetrica(matriz);
23         if (simetrica)
24             System.out.println("La matriz es simétrica");
25         else
26             System.out.println("La matriz no es simétrica");
27     }
28 }
```

Tema 4. Búsqueda y ordenación

Ejemplo. Otro método de clase para comprobar si una matriz es **simétrica**.

Otra solución sería comparar solo los elementos de la triangular superior con los de la triangular inferior o al contrario. Con ello, disminuiríamos el número de iteraciones de los bucles. Consideremos por ejemplo los elementos de la triangular superior, bastaría comparar los incluidos en el triángulo rojo con los del triángulo azul.



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23
a30	a31	a32	a33

En este caso se cumple:

El índice para la fila empieza en 0 y termina en el número de filas menos 1.

El índice para la columna empieza en la fila + 1 y termina en el número de columnas.

Tema 4. Búsqueda y ordenación

```
public static boolean esSimetrica(int [][] matriz) {  
    boolean simetria = true;  
    int fila = 0; int columna;  
    while (simetria && (fila < matriz.length - 1)) {  
        columna = fila + 1;  
        while (simetria && (columna < matriz[fila].length)) {  
            if (matriz[fila][columna] != matriz[columna][fila])  
                simetria = false;  
            columna++;  
        }  
        fila++;  
    }  
    return simetria;  
}
```


Tema 4. Búsqueda y ordenación

Ejemplo. Método de clase para obtener el **producto de dos matrices**.

Para calcular el producto de las matrices **A** (de dimensiones $m \times p$) y **B** (de dimensiones $p \times n$):

$$A_{m \times p} \times B_{p \times n} = C_{m \times n}$$

Utilizamos la expresión:

$$c_{ij} = \sum_{k=1}^p a_{ik} * b_{kj}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}$$

.....

En nuestro caso, las matrices son **cuadradas** $\Rightarrow m = p = n$

Tema 4. Búsqueda y ordenación

```
public static int [][] producto(int [][] a, int [][] b) { // OJO! Matrices cuadradas
    if (a.length != b.length) throw new RuntimeException("Las dimensiones no
        coinciden, no se puede realizar la operación");
    int [][] c = new int [a.length][b.length];
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < b.length; j++) {
            c[i][j] = 0;
            for (int k = 0; k < b.length; k++) { // Desarrolla la sumatoria
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

Tema 4. Búsqueda y ordenación

Búsqueda

- ☐ Búsqueda lineal o secuencial
- ☐ Búsqueda en un array de objetos

Tema 4. Búsqueda y ordenación

Búsqueda

□ Búsqueda lineal o secuencial - Idea

La *búsqueda* es un proceso que permite determinar si un valor específico está en un array. En el caso de una *búsqueda lineal*, compararemos el valor que buscamos y que llamaremos **clave**, secuencialmente con cada elemento del array. El proceso continuará hasta que encontremos el valor buscado o hayamos visitado todos los elementos del array sin éxito. La búsqueda lineal **devuelve la posición** del elemento que coincide con la clave o **-1** si no lo hemos encontrado.

Array	array	4	6	8	-5	1
		[0]	[1]		[4]

El valor **clave** se compara con `array[i]` utilizando un **for** `i = 0, 1, ...`

Ejemplos: `clave = 8` \Rightarrow devuelve posición que ocupa en el array: 2

`clave = 50` \Rightarrow no está en el array, devuelve -1

Tema 4. Búsqueda y ordenación

Búsqueda

□ Búsqueda lineal o secuencial - Implementación

Ejemplo 1. Método de clase para la búsqueda secuencial o lineal.

```
/** El método busca el valor clave en el array array */  
public static int busquedaLineal(int[] array, int clave) {  
    for (int i = 0; i < array.length; i++) {  
        if (clave == array[i])  
            return i;  
    }  
    return -1;  
}
```

```
1 package org.ip.tema05;
2
3 import java.util.Arrays;
4 import java.util.Scanner;
5
6 public class BusquedaLineal {
7
8     // El metodo busca el valor clave en el array 'array'
9     public static int busquedaLineal(int[] array, int clave) {
10         for (int i = 0; i < array.length; i++) {
11             if (clave == array[i])
12                 return i;
13         }
14         return -1;
15     }
16
17     public static void main(String[] args) {
18         int [] array = {4, 6, 8, -5, 1};
19         Scanner entrada = new Scanner(System.in);
20         System.out.println("Introduce el valor que buscas en el array ");
21         int clave = entrada.nextInt();
22         int posicion = busquedaLineal(array, clave);
23         if (posicion == -1) {
24             System.out.println("El valor " + clave + " no está en el array");
25             System.out.println(Arrays.toString(array));
26         }
27         else {
28             System.out.println("El valor " + clave + " está en el array "
29                 + "en la posicion " + (posicion + 1));
30             System.out.println(Arrays.toString(array));
31         }
32     }
33 }
```

Tema 4. Búsqueda y ordenación

Búsqueda

□ La interface Comparable

La interface `java.lang.Comparable` define un único método

```
public interface Comparable {  
    public int compareTo(Object o);  
}
```

- El método `compareTo` debe devolver un entero negativo (-1), 0, o positivo (1) cuando el objeto que recibe el mensaje es menor, igual o mayor que el `Object o`, pasado como parámetro
- Es aconsejable que el método `compareTo` compare igual que el método `equals` para la igualdad, esto es, que devuelva 0 si y solo si los dos objetos que se comparan con `equals` son iguales. Ya que tanto `equals` como `compareTo` no deben cambiar mientras que los objetos están dentro de las colecciones

Tema 4. Búsqueda y ordenación

➔ Búsqueda

❑ La interface Comparable

Forma de hacer `compareTo` *consistente* con `equals`

```
public boolean equals(Object o) {  
    if (!(o instanceof Punto2D))  
        return false;  
    Punto2D punto = (Punto2D) o;  
    return (x == punto.x) && (y == punto.y);  
}  
  
public int compareTo(Object o) {  
    if (this.equals(o))  
        return 0; // consistente con equals  
    Punto2D punto = (Punto2D) o;  
    if (x >= punto.x && y > punto.y)  
        return 1;  
    return -1;  
}
```


Tema 4. Búsqueda y ordenación

Búsqueda

□ La interface Comparable

- Si en una ordenación el objeto obj1 que invoca el método, va antes (menor) que el objeto obj2 que se le pasa como parámetro, el método devolverá un número negativo (-1), si va después (mayor) devolverá un número positivo (1) y si se consideran iguales a efectos de ordenación devolverá un cero (0)
- `obj1.compareTo(obj2) < 0` si obj1 va antes que obj2
- `obj1.compareTo(obj2) > 0` si obj1 va después que obj2
- `obj1.compareTo(obj2) = 0` si obj1 es igual que obj2
- Como ya sabemos, las **interfaces** no son instanciables, pueden heredar unas de otras y **pueden crear variables cuyo tipo es una interfaz**, con las que podemos referenciar cualquier objeto de cualquier clase que la implemente. Y como tal, con la interface Comparable se puede hacer todo eso

Tema 4. Búsqueda y ordenación

➡ Búsqueda

□ Búsqueda lineal en un array de objetos Comparable

```
public static int busquedaLineal(Comparable[] array, Comparable clave) {  
    for (int i = 0; i < array.length; i++) {  
        if (array[i].equals(clave))  
            return i;  
    }  
    return -1;  
}
```

```
public static int busquedaLineal(Comparable[] array, Comparable clave) {  
    for (int i = 0; i < array.length; i++) {  
        if (array[i].compareTo(clave) == 0) // debe ser consistente con equals => if (array[i].equals(clave))  
            return i;  
    }  
    return -1;  
}
```

Tema 4. Búsqueda y ordenación

➡ Búsqueda

❑ Búsqueda lineal en un array de objetos Comparable

```
public static int busquedaLineal(Comparable[] array, Comparable clave) {
    for (int i = 0; i < array.length; i++) {
        if (array[i].equals(clave))
            return i;
    }
    return -1;
}

public static int busquedaLineal(Comparable[] array, Comparable clave) {
    for (int i = 0; i < array.length; i++) {
        if (array[i].compareTo(clave) == 0) // debe ser consistente con equals => if (array[i].equals(clave))
            return i;
    }
    return -1;
}

// El metodo busca el valor clave en el array 'array'
public static int busquedaLineal(int[] array, int clave) {
    for (int i = 0; i < array.length; i++) {
        if (clave == array[i])
            return i;
    }
    return -1;
}
```

Tema 4. Búsqueda y ordenación

➡ Búsqueda

❑ Búsqueda lineal en un array de objetos Comparable

```
public static void main(String[] args) {  
    Fraccion [] arrayFracciones = {new Fraccion(9, 2), new Fraccion(-3, 4), new Fraccion(1, 2),  
        new Fraccion(7, 5), new Fraccion(1, 5), new Fraccion(1, 7), new Fraccion(7, 9),  
        new Fraccion(1, 3), new Fraccion(7, 13), new Fraccion(-3, 7)};  
    System.out.println("El array de fracciones es: " + Arrays.toString(arrayFracciones));  
    Fraccion clave = new Fraccion(1, 2);  
    int posicion = busquedaLineal(arrayFracciones, clave);  
    System.out.println("La fraccion " + clave + " esta en la posicion " + posicion);  
    Fraccion otraClave = new Fraccion(5, 7);  
    System.out.println("La fraccion " + otraClave + " esta en la posicion " + busquedaLineal(arrayFracciones, otraClave));  
}
```

Salida

```
El array de fracciones es: [9/2, -3/4, 1/2, 7/5, 1/5, 1/7, 7/9, 1/3, 7/13, -3/7]  
La fraccion 1/2 esta en la posicion 2  
La fraccion 5/7 esta en la posicion -1
```

Tema 4. Búsqueda y ordenación



Ordenación

- ☐ Introducción
- ☐ Método de la burbuja
- ☐ Método de selección
- ☐ Ordenación de un array de objetos

Tema 4. Búsqueda y ordenación

Ordenación

□ Introducción

Una de las tareas más frecuentes en programación es la **ordenación**. Los métodos de ordenación se dividen en dos categorías, según la estructura de datos utilizada:

- Métodos de **ordenación interna**: consiste en la ordenación de arrays, que se almacenan en memoria interna. Estos a su vez se clasifican:
 - ✓ Métodos **directos**: son métodos sencillos, pero poco eficientes con complejidades de $O(n^2)$. Burbuja, selección, inserción (se verá en MP).
 - ✓ Métodos **avanzados**: algoritmos más eficientes, $O(n \log n)$. Quicksort y Mergesort.
- Métodos de **ordenación externa**: consiste en la ordenación de archivos, es decir, de estructuras que se almacenan en dispositivos externos.

Tema 4. Búsqueda y ordenación

Ordenación

☐ Método de la Burbuja - Idea

Se desea ordenar un array en **orden creciente**. Este método debe su nombre al hecho de que los valores más pequeños *burbujean* hacia la parte superior.

Supongamos que queremos ordenar el array:

23	15
19	19
45	23
31	31
15	45

La idea básica consiste en **comparar** elementos que ocupan **posiciones consecutivas** e **intercambiarlos** si no están en el orden deseado (creciente o decreciente), repitiendo la operación hasta que el array quede ordenado.

❑ Método de la Burbuja - Proceso

Pasada $i = 1$

23
19
45
31
15



19
23
45
31
15



19
23
45
31
15



19
23
31
45
15



19
23
31
15
45

$j = 0$

$j = 1$

$j = 2$

$j = 3$

4 Comparaciones

Pasada $i = 2$

19
23
31
15
45



19
23
31
15
45



19
23
31
15
45



19
23
15
31
45



19
23
15
31
45

❑ Método de la Burbuja - Proceso

Pasada $i = 3$

19
23
15
31
45



19
23
15
31
45



19
15
23
31
45



19
15
23
31
45



19
15
23
31
45

$j = 0$

$j = 1$

$j = 2$

$j = 3$

4 Comparaciones

Pasada $i = 4$

19
15
23
31
45



15
19
23
31
45



15
19
23
31
45



15
19
23
31
45



15
19
23
31
45

Se han necesitado **$N - 1$ comparaciones** y **$N - 1$ pasadas** para ordenar el array, siendo **N** el nº de elementos del array

Tema 4. Búsqueda y ordenación

❑ Método de la Burbuja - Implementación

/** El método ordena el array array por el método de la burbuja */

```
public static void burbuja(int[] array) {
```

```
    int aux;
```

```
    for (int i = 1; i < array.length; i++)
```

→ Bucle de las pasadas

```
        for (int j = 0; j < array.length - 1; j++)
```

→ Bucle de las comparaciones

```
            if (array[j] > array[j + 1]) {
```

```
                // Intercambio
```

```
                aux = array[j];
```

```
                array[j] = array[j + 1];
```

```
                array[j + 1] = aux;
```

```
            }
```

```
}
```

Tema 4. Búsqueda y ordenación

❑ Método de la Burbuja - Primera mejora - Idea

Una **primera mejora** consiste en considerar que en la primera pasada el elemento mayor ocupa la última posición, en la segunda pasada el siguiente mayor ocupa la penúltima posición y así sucesivamente. Esto va a suponer que no es necesario comparar hasta el último elemento del array en todas las pasadas

Pasada 1 haremos $N - 1$ comparaciones

Pasada 2 haremos $N - 2$ comparaciones

.....

.....

Pasada i haremos $N - i$ comparaciones

Podemos modificar el **bucle de las comparaciones** teniendo en cuenta esta mejora, y que itere con j desde **0** hasta **`array.length - i`**

Tema 4. Búsqueda y ordenación

❑ Método de la Burbuja - Primera mejora - Implementación

/** El método ordena el array array por el método de la burbuja */

```
public static void burbujaMejorado(int[] array) {  
    int aux;  
    for (int i = 1; i < array.length; i++) → Bucle de las pasadas  
        for (int j = 0; j < array.length - i; j++) → Bucle de las comparaciones  
            if (array[j] > array[j + 1]) {  
                // Intercambio  
                aux = array[j];  
                array[j] = array[j + 1];  
                array[j + 1] = aux;  
            }  
}
```

Tema 4. Búsqueda y ordenación

❑ Método de la Burbuja - Segundo mejora - Idea

Una **segunda mejora** consiste en considerar que, si en una determinada pasada el array queda ordenado, no es necesario seguir realizando pasadas. Un array estará ordenado si en una pasada no se han producido *Intercambios*. Podemos utilizar una variable lógica que me informe de esta situación. Utilizamos la variable boolean (flag) *cambiado*, si está a verdadero (true) significa que hemos realizado algún *Intercambio*, luego debemos de continuar realizando pasadas y, si está a falso (false) el array está ordenado, puesto que no se ha producido ningún *Intercambio* y no hará más pasadas (es decir, el algoritmo termina).

En esta mejora, será el **bucle de las pasadas** el que se deba de modificar, incluyendo la variable boolean (cambiado), y cuando ésta sea false después de la pasada, el algoritmo terminará, pues no se ha producido ningún intercambio en esa última pasada y el array estará ordenado.

❑ Método de la Burbuja - Segunda mejora - Implementación

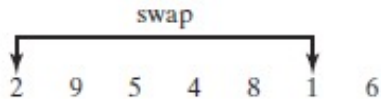
```
public static void burbujaMejoradoFlag(int[] array) {  
    boolean cambiado = true;  
    int i = 1;  
    int aux;  
    while (cambiado && (i < array.length)) { —————> Bucle de las pasadas  
        cambiado = false;  
        for (int j = 0; j < array.length - i; j++) { ———> Bucle de las comparaciones  
            if (array[j] > (array[j + 1])) {  
                // Intercambio  
                aux = array[j];  
                array[j] = array[j + 1];  
                array[j + 1] = aux;  
                cambiado = true;;  
            }  
        }  
        i++;  
    }  
}
```

```
1 package org.ip.tema05;
2
3 import java.util.Arrays;
4
5 public class Ordenacion {
6
7     public static void burbuja(int[] array) {
8         int aux;
9         for (int i = 1; i < array.length; i++)
10             for (int j = 0; j < array.length - 1; j++)
11                 if (array[j] > array[j + 1]) {
12                     // Intercambiar
13                     aux = array[j];
14                     array[j] = array[j + 1];
15                     array[j + 1] = aux;
16                 }
17     }
18
19     public static void burbujaMejorado(int[] array) {
20         int aux;
21         for (int i = 1; i < array.length; i++)
22             for (int j = 0; j < array.length - i; j++)
23                 if (array[j] > array[j + 1]) {
24                     // Intercambiar
25                     aux = array[j];
26                     array[j] = array[j + 1];
27                     array[j + 1] = aux;
28                 }
29     }
30 }
```

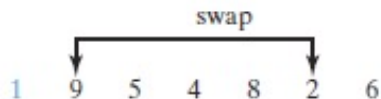
```
31 public static void burbujaMejoradoFlag(int[] array) {  
32     boolean cambiado = true;  
33     int aux;  
34     int i = 1;  
35     while (cambiado && (i < array.length)) {  
36         cambiado = false;  
37         for (int j = 0; j < array.length - i; j++) {  
38             if (array[j] > array[j + 1]) {  
39                 // Intercambiar  
40                 aux = array[j];  
41                 array[j] = array[j + 1];  
42                 array[j + 1] = aux;  
43                 cambiado = true;  
44             }  
45         }  
46         i++;  
47     }  
48 }  
49
```


❑ Método de Ordenación por Selección - Idea

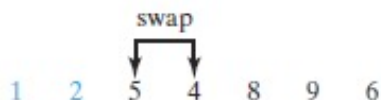
Consiste en encontrar y **seleccionar** el elemento menor y colocarlo en la primera posición, el siguiente menor se coloca en la segunda posición, y así sucesivamente.



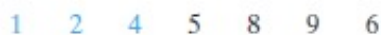
El valor más pequeño, 1, se intercambia con el que ocupa la 1ª posición, 2.



El nº 1 está bien colocado y ya no lo tendremos en cuenta. Seleccionamos 2 (siguiente más pequeño) y lo intercambiamos con 9 (el primero en la lista que queda)



El nº 2 está bien colocado y ya no lo tendremos en cuenta. Seleccionamos 4 (siguiente más pequeño) y lo intercambiamos con 5 (el primero en la lista que queda)



El nº 4 está bien colocado y ya no lo tendremos en cuenta. Seleccionamos 5 (siguiente más pequeño) y está bien colocado. No es necesario intercambiarlo.



El nº 5 está bien colocado y ya no lo tendremos en cuenta. Seleccionamos 6 (siguiente más pequeño) y lo intercambiamos con 8 (el primero en la lista que queda)



El nº 6 está bien colocado y ya no lo tendremos en cuenta. Seleccionamos 8 (siguiente más pequeño) y lo intercambiamos con 9 (el primero en la lista que queda)



El nº 8 está bien colocado y ya no lo tendremos en cuenta. Queda un solo elemento en la lista, la ordenación ha terminado.

❑ Método de Ordenación por Selección - Implementación

```
public static void seleccion(int[] array) {  
    for (int i = 0; i < array.length - 1; i++) {  
        // Buscamos el mínimo en array[i .. array.length - 1]  
        int valorMinimo = array[i];  
        int posicionMinimo = i;  
        for (int j = i + 1; j < array.length; j++) {  
            if (array[j] < valorMinimo) {  
                valorMinimo = array[j];  
                posicionMinimo = j; }  
        }  
        // Intercambiamos array[i] con array[posicionMinimo] si es necesario  
        if (posicionMinimo != i) {  
            array[posicionMinimo] = array[i];  
            array[i] = valorMinimo;  
        }  
    }  
}
```

**Búsqueda del valor mínimo y
de la posición que ocupa**

```
62 public static void seleccion(int[] array) {  
63     for (int i = 0; i < array.length - 1; i++) {  
64         // Buscar el mínimo en array[i .. a.length-1]  
65         int valorMinimo = array[i];  
66         int posicionMinimo = i;  
67         for (int j = i + 1; j < array.length; j++) {  
68             if (array[j] < valorMinimo) {  
69                 valorMinimo = array[j];  
70                 posicionMinimo = j;  
71             }  
72         }  
73         // Intercambio array[i] con  
74         // array[posicionMinimo] si es necesario  
75         if (posicionMinimo != i) {  
76             array[posicionMinimo] = array[i];  
77             array[i] = valorMinimo;  
78         }  
79     }  
80 }  
81
```

```
70 public static void main(String[] args) {  
71     int [] array1 = {23, 19, 45, 31, 15, -7, 0};  
72     int [] array2 = {15, 2, 20, 39, -5, 14, 0};  
73     System.out.println("Array sin ordenar: " + Arrays.toString(array1));  
74     burbujaMejoradoFlag(array1);  
75     System.out.println("Array ordenado por el metodo de la burbuja mejorado: "  
76         + Arrays.toString(array1));  
77     System.out.println("Array sin ordenar: " + Arrays.toString(array2));  
78     seleccion(array2);  
79     System.out.println("Array ordenado por el metodo de seleccion: "  
80         + Arrays.toString(array2));  
81 }
```

Salida

```
Array sin ordenar: [23, 19, 45, 31, 15, -7, 0]  
Array ordenado por el metodo de la burbuja mejorado: [-7, 0, 15, 19, 23, 31, 45]  
Array sin ordenar: [15, 2, 20, 39, -5, 14, 0]  
Array ordenado por el metodo de seleccion: [-5, 0, 2, 14, 15, 20, 39]
```

Tema 4. Búsqueda y ordenación

Ordenación

☐ Ordenación de un array de objetos Comparable

En este apartado se presenta un método de clase **genérico** para ordenar cualquier array de objetos siempre que dichos objetos sean instancias o ejemplares de clases que implementan la interface **Comparable** y que por tanto pueden utilizar el método `compareTo`

Las clases `Integer`, `Double`, `Character` (todos los envoltorios) y `String` implementan la interface `Comparable`, esto significa que los objetos de estas clases pueden compararse utilizando el método `compareTo`. Asimismo, la clase `Fraccion` que hemos diseñado también implementa la interface `Comparable`

Vamos a reescribir los métodos de ordenación estudiados para que nos sirvan para ordenar cualquier array de objetos (**genérico**)

❑ Método de Ordenación por Selección - Implementación genérica

Ordenación array de enteros (int)

```
public static void seleccion(int[] a) {  
    for (int i = 0; i < a.length - 1; i++) {  
        // Busco el mínimo en a[i ..a.length-1]  
        → int valorMinimo = a[i];  
        int posicionMinimo = i;  
        for (int j = i + 1; j < a.length; j++) {  
            → if (a[j] < valorMinimo) {  
                valorMinimo = a[j];  
                posicionMinimo = j;  
            }  
        }  
        // Intercambio a[i] con a[posicionMinimo]  
        //si es necesario  
        if (posicionMinimo != i) {  
            a[posicionMinimo] = a[i];  
            a[i] = valorMinimo;  
        }  
    }  
}
```

Ordenación array de Comparable

```
public static void seleccion(Comparable[] a) {  
    for (int i = 0; i < a.length - 1; i++) {  
        // Busco el mínimo en a[i ..a.length-1]  
        → Comparable valorMinimo = a[i];  
        int posicionMinimo = i;  
        for (int j = i + 1; j < a.length; j++) {  
            → if (a[j].compareTo(valorMinimo) < 0) {  
                valorMinimo = a[j];  
                posicionMinimo = j;  
            }  
        }  
        // Intercambio a[i] con a[posicionMinimo]  
        //si es necesario  
        if (posicionMinimo != i) {  
            a[posicionMinimo] = a[i];  
            a[i] = valorMinimo;  
        }  
    }  
}
```



```
1 package org.ip.tema05;
2
3 import java.util.Arrays;
4
5 import org.ip.sesion06.Fraccion;
6
7 public class OrdenacionGenerica {
8
9     public static void seleccion(Comparable[] array) {
10         for (int i = 0; i < array.length - 1; i++) {
11             // Buscar el mínimo en array[i .. a.length-1]
12             Comparable valorMinimo = array[i];
13             int posicionMinimo = i;
14             for (int j = i + 1; j < array.length; j++) {
15                 if (array[j].compareTo(valorMinimo) < 0) {
16                     valorMinimo = array[j];
17                     posicionMinimo = j;
18                 }
19             }
20             // Intercambio array[i] con
21             // array[posicionMinimo] si es necesario
22             if (posicionMinimo != i) {
23                 array[posicionMinimo] = array[i];
24                 array[i] = valorMinimo;
25             }
26         }
27     }
```

```
29 public static void main(String[] args) {
30     Fraccion [] arrayFracciones = {new Fraccion(9, 2), new Fraccion(-3, 4),
31         new Fraccion(7, 5), new Fraccion(1, 5)};
32     System.out.println("El array de fracciones es: " + Arrays.toString(arrayFracciones));
33     seleccion(arrayFracciones);
34     System.out.println("El array de fracciones ordenado es: " + Arrays.toString(arrayFracciones));
35     Integer [] arrayEnteros = {new Integer(7), new Integer(9), new Integer(3)};
36     System.out.println("El array de enteros es: " + Arrays.toString(arrayEnteros));
37     seleccion(arrayEnteros);
38     System.out.println("El array de enteros ordenado es: " + Arrays.toString(arrayEnteros));
39     String [] arrayStrings = {"Lopez", "Ayala", "Garcia", "Benitez"};
40     System.out.println("El array de strings es: " + Arrays.toString(arrayStrings));
41     seleccion(arrayStrings);
42     System.out.println("El array de strings ordenado es: " + Arrays.toString(arrayStrings));
43 }
44 }
```

Salida

```
El array de fracciones es: [9/2, -3/4, 7/5, 1/5]
El array de fracciones ordenado es: [-3/4, 1/5, 7/5, 9/2]
El array de enteros es: [7, 9, 3]
El array de enteros ordenado es: [3, 7, 9]
El array de strings es: [Lopez, Ayala, Garcia, Benitez]
El array de strings ordenado es: [Ayala, Benitez, Garcia, Lopez]
```


Tema 4. Búsqueda y ordenación

Ordenación

Comparativa métodos de ordenación \Rightarrow *Ranking runtime* se refiere al tiempo de ordenar n enteros generados aleatoriamente, Merge Sort (referencia en JCF) y Quick Sort (primitiva en JCF)

Algoritmo	Estable	Com. worstTime	Com. averageTime	Ranking runTime	worstSpace
Insertion Sort	SI	Cuadrático $O(n^2)$	Cuadrático $O(n^2)$	5	Constante
Selection Sort	NO	Cuadrático $O(n^2)$	Cuadrático $O(n^2)$	6	Constante
Bubble Sort	SI	Cuadrático $O(n^2)$	Cuadrático $O(n^2)$	7	Constante
Merge Sort	SI	$O(n * \log_2 n)$	Logarítmico $O(n * \log_2 n)$	2	Lineal
Quick Sort	NO	Cuadrático $O(n^2)$	Logarítmico $O(n * \log_2 n)$	1	Lineal
Radix Sort	SI	$O(n * \log_2 n)$	Logarítmico $O(n * \log_2 n)$	4	Lineal
Heap Sort	NO	$O(n * \log_2 n)$	Logarítmico $O(n * \log_2 n)$	3	Lineal

Tema 4. Búsqueda y ordenación

Ordenación

❑ Aclaración sobre **estabilidad** en ordenación \Rightarrow Un algoritmo es **estable** sólo cuando hay dos objetos R y S con la misma clave y, apareciendo R antes que S en la lista original, aparecen en ese orden en la lista ordenada

❑ Algoritmos mantienen el orden relativo entre éstos y otros no. Teniendo una lista de (nombre, edad): “Pedro 19, Juan 23, Fidel 15, María 17, Juan 18, María 20”, y la ordenamos alfabéticamente por el nombre. Con un algoritmo **estable** tendríamos: “Fidel 15, Juan 23, Juan 18, María 17, María 20, Pedro 19”. Un algoritmo **no estable** podríamos tener a Juan 18 antes de Juan 23, o a María 17 después de María 20

¡MUCHAS GRACIAS!



UNIVERSIDAD DE ALMERÍA

