



Tema 5. Arrays

5.1. Descripción

5.2. Operaciones

5.3. Arrays multidimensionales

5.4. Algoritmos con arrays
(Búsqueda y Ordenación)



Objetivos

- ✓ Conocer el tipo de dato array, cómo se define y las condiciones de su aplicación.
- ✓ Conocer el tipo de dato string, cómo se define, las condiciones de su aplicación y las principales funciones y procedimientos predefinidos.
- ✓ Presentar algoritmos fundamentales de búsqueda y ordenación de arrays.



5.1. Descripción

- ✓ **Primera visión:** un *array* es un tipo de dato *estructurado* que permite almacenar la información de forma compacta y manejable.

1	-2	4
---	----	---

ElVector

ElTablero

‘c’	‘X’	‘c’
‘X’	‘X’	
	‘c’	



5.1. Descripción

- ✓ Un **array** es una ***colección estructurada*** de componentes del ***mismo tipo*** a las que se puede acceder de forma individual por su posición dentro de la colección.
- ✓ Toda la colección de datos se almacena en un área de memoria contigua bajo ***un solo nombre***.
- ✓ Para ***acceder*** a cada ***componente individual*** se utilizan ***índices*** que indican la posición de la componente dentro de la colección.



5.1. Descripción

- **Ejemplo:**

1	-2	4
---	----	---



el array se denomina: ElVector

el elemento que ocupa la segunda posición del array ElVector

el array se denomina: ElTablero

elemento que ocupa la tercera fila y primera columna del array ElTablero

'c'	'x'	'c'
'x'	'x'	
	'c'	





5.1. Descripción

- ✓ Un **array** es **unidimensional** si a cada componente se accede mediante un único índice.

Ejemplo:

1	-2	4
---	----	---

 [posición]

- ✓ Un **array** es **bidimensional** si a cada componente se accede mediante 2 índices.

Ejemplo:

'c'	'x'	'c'
'x'	'x'	
	'c'	

 [posFila, posColumna]

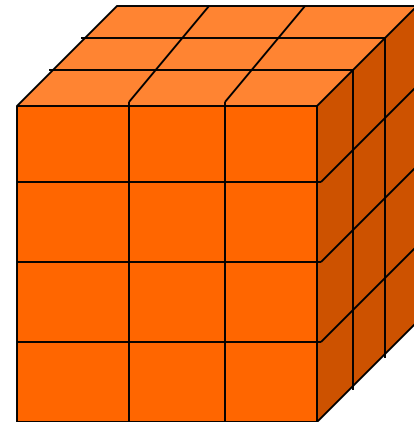


5.1. Descripción

- ✓ Un **array** es **multidimensional** si a cada componente se accede mediante más de un índice.

Ejemplo:

[pos1D, pos2D, pos3D]





5.1. Declaración de arrays de una dimensión

Declaración de un array de una dimensión:	tipo_dato [] nombre_array;
Declaración de un array de enteros:	int [] numeros;
Crear un array de 4 números enteros:	numeros=new int[4]; Nombre_array = new tipo_dato[tamaño]
La declaración y la creación del array se pueden hacer en una misma línea	int [] numeros=new int[4];
Crear un array de 20 números reales:	float [] arr=new float[20];



5.1. Inicializar y usar los elementos de un array

- ✓ Java por defecto **inicializa** todas sus posiciones a 0.
- ✓ Si quisiéramos hacerlo nosotros →
- ✓ Puede **inicializarse** en un bucle **for** como resultado de alguna operación

```
int[] numeros = new int[4];
```

```
numeros[0]=2;  
numeros[1]=-4;  
numeros[2]=15;  
numeros[3]=-25;
```

```
for(int i=0; i<4; i++){  
    numeros[i]=i*i+4;  
}
```

- ✓ Su miembro **length** nos proporciona la **dimensión** del array

```
for(int i=0; i<numeros.length; i++){  
    numeros[i]=i*i+4;  
}
```

- ✓ Se pueden declarar, crear e inicializar en una misma línea

```
int[] numeros={2, -4, 15, -25};  
String[] nombres={"Juan", "José", "Miguel", "Antonio"};
```

- ✓ Para imprimir los elementos del array *nombres* y *numeros* haríamos

```
for(int i=0; i<nombres.length; i++){  
    System.out.println(nombres[i]);  
}
```

Salida:

Juan
José
Miguel
Antonio

```
for(int i=0; i<numeros.length; i++){  
    System.out.println(numeros[i]);  
}
```

Salida:

2
-4
15
-25



5.1. Declaración de arrays multidimensionales

Declaración de un array multidimensionales	<code>tipo_dato [] [] nombre_array;</code>
Declaración de un array de 2 dimensiones de enteros:	<code>int [] [] tabla;</code>
Crear un array de 2 filas y 3 columnas de números enteros:	<code>tabla=new int [2] [3];</code> <code>nombre_array = new tipo_dato[tamaño][tamaño];</code>
La declaración y la creación del array se pueden hacer en una misma línea	<code>int [] []tabla=new int[2][3];</code>
Crear un array de 4 números reales:	<code>float [] [] tabla=new float[2] [2];</code>



5.1.1 DECLARACION

✓ Declaración de arrays, ejemplos:

1	-2	4
---	----	---

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

'c'	'x'	'c'
'x'	'x'	
	'c'	

```
char [] [] caracteres=new char [3][3];
```

0	2
7	-2

```
int [][][] m_enteros = new int [4][3][3];
```



5.1.2 ACCESO

- ✓ En el **acceso a un elemento** del array se utilizan tantos índices como dimensiones tenga dicho array.
- ✓ Para acceder a un elemento: se expresa el nombre de la variable array y entre corchetes el o los valores de los índices del elemento dentro del array.



5.1.1

✓ Acceso a los elementos:

1	-2	4
---	----	---

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

```
a_enteros [1];
```

'c'	'x'	'c'
'x'	'x'	
	'c'	

```
char [] [] caracteres=new char [3][3];
```

```
caracteres[1][1];
```

0		
		2
7		
		-2

```
int [][][] m_enteros = new int [4][3][3];
```

```
m_enteros [1][2][0];
```



5.1.3 Descripción

- ✓ Los arrays son **estructuras de acceso directo**, ya que permiten almacenar y recuperar directamente los datos, especificando su posición dentro de la estructura.
- ✓ Los arrays son **estructuras de datos homogéneas**: sus elementos son TODOS del MISMO TIPO.
- ✓ El **tamaño** de un array se puede establecer de forma *fija*, cuando se define una variable de este tipo o *variable* en tiempo de ejecución.



asignación fija o variable

Descripción

- ✓ **Definición de tipos arrays**, ejemplos:



5.2 Operaciones

- ✓ **Asignación** de un array.
- ✓ Operación de **entrada** de cada una de las componentes de un array.
- ✓ Operación de **salida** de cada una de las componentes de un array.
- ✓ **NO** se pueden realizar operaciones de entrada/salida con arrays completos.



Operaciones sobre arrays

- Operación de **entrada** de los componentes de un array:

```
int[] numeros = new int[4];  
  
for (int i = 0; i < numeros.length; i++) {  
    System.out.println("Dame el número " + i);  
    numeros[i] = input.nextInt();  
}
```

- Operación de **salida** de los componentes de un array:

```
for (int i = 0; i < numeros.length; i++) {  
    System.out.println("La posicion " + i + " es " + numeros[i]);  
}
```



Ejemplo 5.4



- Ejemplos de **Proceso de todas las componentes** de un array :

```
int [] numeros={2, -4, 15, -25};  
String[] nombres={"Juan", "José", "Miguel", "Antonio"};
```

```
for(int i=0; i<nombres.length; i++){  
    System.out.println(nombres[i]);  
}
```

```
for(int i=0; i<numeros.length; i++){  
    System.out.println(numeros[i]);  
}
```

- NO** se pueden realizar operaciones de entrada/salida con arrays completos:

 `System.out.println(numeros);` 



5.2 Operaciones

Asignación a arrays completos

- ✓ Dado que los arrays se manejan usando referencias, si asignamos un array a otro, no copiamos los valores de uno en otro, lo que hacemos es apuntar las dos referencias al mismo array.

```
int a[] = {1, 2, 3};  
int b[] = {2, 4, 6};  
b = a;  
System.out.println(b[1]);  
a[1] = 99;  
System.out.println(b[1]);
```

a y b apuntan al array {1,2,3}

Salida: 2

{1,99,3}

Salida: 99

- ✓ a y b apuntan al mismo conjunto de datos, no hemos realizado copia de un array en otro.



5.2 Operaciones

Copia de los valores de un array en otro

- ✓ Recorremos uno y copiamos cada uno de los elementos de un array en el otro.
- ✓ Los datos del array origen y del array destino no ocuparan el mismo lugar → serán independientes

```
int[] origen = {1, 3, 5, 7};  
int[] destino = new int[origen.length];  
for (int i = 0; i < origen.length; i++) {  
    destino[i] = origen[i];  
}
```



5.2 Operaciones

Si hacemos la asignación anterior

```
System.out.println("Despues de la copia");
System.out.print("origen=[ ");
for (int i = 0; i < origen.length; i++) {
    System.out.print(origen[i] + " ");
}
System.out.println("]");
System.out.print("destino=[ ");
for (int i = 0; i < destino.length; i++) {
    System.out.print(destino[i] + " ");
}
System.out.println("]");
origen[1] = 99;
System.out.println("Despues de la asignación");
System.out.print("origen=[ ");
for (int i = 0; i < origen.length; i++) {
    System.out.print(origen[i] + " ");
}
System.out.println("]");
System.out.print("destino=[ ");
for (int i = 0; i < destino.length; i++) {
    System.out.print(destino[i] + " ");
}
System.out.println("]");
```

Salida:

```
Despues de la copia
origen=[ 1 3 5 7 ]
destino=[ 1 3 5 7 ]
Despues de la asignación
origen=[ 1 99 5 7 ]
destino=[ 1 3 5 7 ]
```



5.2 Operaciones

✓ Recomendaciones

➤ Evitar errores de intervalo:

→ cuando un subíndice toma valores fuera de su rango definido.



5.3 Arrays multidimensionales

- ✓ En algunos problemas los datos que se procesan se pueden organizar de forma natural como una tabla (2 dimensiones) o con un array de más de 2 índices.

Ejemplo: las temperaturas máximas de varias ciudades a lo largo de los días de un mes.

Día\Ciudad	Soria	Madrid	Ávila	...
1	15.3	17.2	12.4	...
2	...			
...	...			
31



5.3 Arrays multidimensionales

- ✓ El acceso a cada elemento de un array de N dimensiones se realiza a través de N índices.
- ✓ Para acceder a todas las componentes de un array multidimensional serán necesarios tantos bucles anidados como dimensiones tenga el array (un bucle para cada dimensión).



5.3 Arrays multidimensionales

- ✓ Los arrays bidimensionales nos permiten representar matrices y por lo tanto operar con ellas.
- ✓ Una matriz de 2 dimensiones $n \times m$ se podrá representar:

`TipoBase [][] nombre = new tipoBase [tam] [tam]`



5.2 Ejemplo Array con 2 dimensiones

- Crear una matriz cuadrada de dimensión 4 (rellenar con ceros todos los elementos excepto los de la diagonal principal $i=j$).

```
double[][] mUnidad = new double[4][4];

for (int i = 0; i < mUnidad.length; i++) {
    for (int j = 0; j < mUnidad[i].length; j++) {
        if (i == j) {
            mUnidad[i][j] = 1.0;
        } else {
            mUnidad[i][j] = 0.0;
        }
    }
}

for (int i = 0; i < mUnidad.length; i++) {
    for (int j = 0; j < mUnidad[i].length; j++) {
        System.out.print(mUnidad[i][j] + "\t");
    }
    System.out.println("");
}
```

Salida:

1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0



5.2 Operaciones

- ✓ **Arrays como parámetros de métodos**
 - Parámetros formales **por valor**, que sean arrays, deben ser **del mismo tipo** que sus correspondientes parámetros reales, **a menos que sean cadenas** (más adelante). En este caso **basta** con que sean de la **misma longitud**.
 - A los métodos se les pueden pasar un array y también pueden devolverlo



5.2 Arrays como parámetros en métodos

```
public static void main(String[] args) {  
  
    int num[] = rellenarArrayDesde(5);  
    imprimirArray(num);  
}
```

```
public static int[] rellenarArrayDesde(int a) {  
    int num[] = new int[10];  
    for (int i = 0; i < num.length; i++) {  
        num[i] = a;  
        a++;  
    }  
    return num;  
}
```

Devuelve un
array de 10
elementos

```
public static void imprimirArray(int lista[]) {  
    for (int i = 0; i < lista.length; i++) {  
        System.out.print(lista[i] + " ");  
    }  
    System.out.println();  
}
```

Recibe un
array como
parámetro



5.2 Método que inserta datos en un array

```
public static void main(String[] args) {  
    int[] array = new int[5];  
    insertarDatosArray(array);  
}  
  
public static void insertarDatosArray(int [] anombre) {  
    Scanner entrada = new Scanner(System.in);  
    for (int i = 0; i < anombre.length; i++) {  
        System.out.print("insertar posicion[" + i + "]:");  
        anombre[i]=entrada.nextInt();  
    }  
}
```

Salida:

```
insertar posicion[0]:18  
insertar posicion[1]:10  
insertar posicion[2]:11  
insertar posicion[3]:
```



valor máximo de un array

```
int max = maxArray(array);  
System.out.println("Max= " + max);
```

```
private static int maxArray(int[] array) {  
    int max = 0;  
    for (int i = 0; i < array.length; i++) {  
        if (array[i] > max) {  
            max = array[i];  
        }  
    }  
    return max;  
}
```



Método que devuelve el promedio de un array

```
double promedio = promedioArray(array);  
System.out.println("Promedio= " + promedio);
```

```
private static double promedioArray(int[] array) {  
    double promedio = 0;  
    for (int i = 0; i < array.length; i++) {  
        promedio = promedio + array[i];  
    }  
    promedio = promedio / array.length;  
    return promedio;  
}
```





5.2 Operaciones

Arrays parcialmente llenos

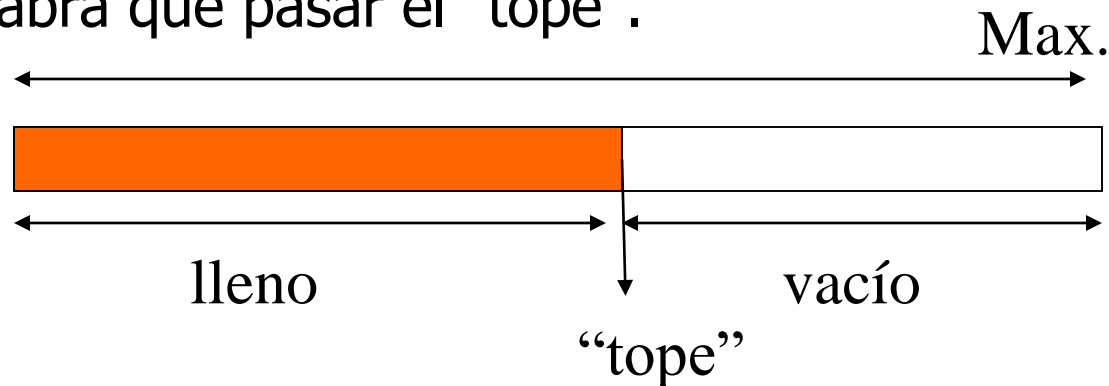
- ✓ Los arrays se declaran en tiempo de compilación y no puede variar su tamaño en tiempo de ejecución; la memoria correspondiente a un array se declara estáticamente (un array tiene un número fijo de componentes).
- ✓ Si este número puede variar de una ejecución a otra, habrá que hacer una estimación del número máximo de elementos que el array puede contener y llevar un registro de la parte ocupada.



5.2 Operaciones

Arrays parcialmente llenos (cont.)

- ✓ Se puede utilizar una variable que vaya marcando el extremo superior de la parte ocupada.
- Definir el array con el máximo y utilizar una variable indicadora ("tope") del número de componentes del array.
- Cuando se pasa el array como parámetro también habrá que pasar el "tope".





datos en un array con tope

```
public static void main(String[] args) throws IOException {
    double[] array = new double[30];
    int top;
    top = insertarDatosArray(array);
    MostrarArrayTope(array, top);
}

private static int insertarDatosArray(double[] array) {
    int select = 0;
    int tope = 0;
    Scanner entrada = new Scanner(System.in);
    System.out.print("¿Quieres introducir empleados? 1 NO 2 SI ");
    select = entrada.nextInt();
    while ((select != 1) && (tope <= array.length)) {
        System.out.println("Empleado N°: " + tope + " : ");
        System.out.print("Salario: ");
        array[tope] = entrada.nextDouble();
        tope = tope + 1;
        System.out.print("1 salir, 2 insertar más");
        select = entrada.nextInt();
    }
    return tope;
}
```



promedio de un array con TOPE

```
double promedio1 = promedioArrayTope(array, top);  
System.out.println("Promedio con tope= " + promedio1);
```

```
private static double promedioArrayTope(int[] array, int tope) {  
    double promedio = 0;  
    for (int i = 0; i < tope; i++) {  
        promedio = promedio + array[i];  
    }  
    promedio = promedio / tope;  
    return promedio;  
}
```



Ejemplo Empleados

Ejemplo: Leer por teclado y mostrar por pantalla los salarios de los empleados de una empresa. (Máximo 30)

Entrada:

Empleado N° 1

Salario: 234.50

¿Más empleados (S/N) ? S

Empleado N° 2

Salario: 345.50

¿Más empleados (S/N) ? n

Salida:

<u>Empleado</u>	<u>Salario</u>
1	234.50
2	345.50

Salario Medio: 290 €



Ejemplo Empleados

```
public static void main(String[] args) throws IOException {
    double[] array = new double[30];
    int top;
    top = insertarDatosArray(array);
    MostarArrayTope(array, top);
}

private static int insertarDatosArray(double[] array) {
    int select = 0;
    int tope = 0;
    Scanner entrada = new Scanner(System.in);
    System.out.print("¿Quieres introducir empleados? 1 NO 2 SI ");
    select = entrada.nextInt();
    while ((select != 1) && (tope <= array.length)) {
        System.out.println("Empleado N°: " + tope + " : ");
        System.out.print("Salario: ");
        array[tope] = entrada.nextDouble();
        tope = tope + 1;
        System.out.print("1 salir, 2 insertar más");
        select = entrada.nextInt();
    }
    return tope;
}
```

Salida:

```
¿Quieres introducir empleados? 1 NO 2 SI 2
Empleado N°: 0 :
Salario: 234,5
1 salir, 2 insertar más2
Empleado N°: 1 :
Salario: 345,5
1 salir, 2 insertar más1
```



Ejemplo Empleados

```
public static void MostrarArrayTope(double array[], int tope) {  
    System.out.println("Empleado          Salario ");  
    System.out.println("-----          ----- ");  
    for (int i = 0; i < tope; i++) {  
        System.out.println(i + "          " + array[i] + " ");  
    }  
    System.out.println();  
}
```

Salida:

Empleado	Salario
-----	-----
0	234.5
1	345.5



5.5. Algoritmos con arrays

- ✓ Dos de las operaciones más usuales con los arrays son:
 - Búsqueda de un dato en un array.
 - Ordenación de las componentes de un array.



5.5.1. Algoritmos de búsqueda

- ✓ Determinan si un dato concreto se encuentra en una colección de datos del mismo tipo.
- ✓ En caso de que se encuentre el dato, permiten conocer la posición que ocupa.
- ✓ Los algoritmos de búsqueda más usuales son:
 - Búsqueda secuencial o lineal
 - Búsqueda binaria o dicotómica



5.5.1. Algoritmos de búsqueda

✓ Precondiciones:

- la búsqueda se hará en un array **unidimensional (lista)**, tratando de encontrar un elemento, ***elemento***.
- el array ***vector*** es de ***n*** elementos.
- ***primero*** y ***ultimo***: primera y última posición entre las que efectuar la búsqueda.



5.5.1. Algoritmos de búsqueda

✓ Objetivo:

Definir una función ***Búsqueda*** que encuentre el primer valor del índice ***posición***, de tal manera que se cumpla que:

vector[posición] = elemBuscado



5.5.1. Algoritmos de búsqueda

Búsqueda secuencial o lineal:

- ✓ Se realiza un recorrido del array comparando el elemento buscado con los valores contenidos en las componentes del array.
- ✓ Se comienza con la primera componente y se va avanzando de forma secuencial hasta que:
 - Se encuentra el valor buscado, o
 - Se llega al final del array (o de su parte ocupada) sin encontrarlo, i. e. devuelve -1.

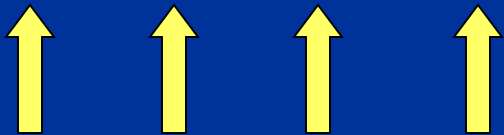


Búsqueda lineal

Búsqueda lineal (el elemento buscado existe)

0	1	2	3	4	5	6	7	8
5	6	7	3	4	9	0	2	1

3



Índice devuelto:

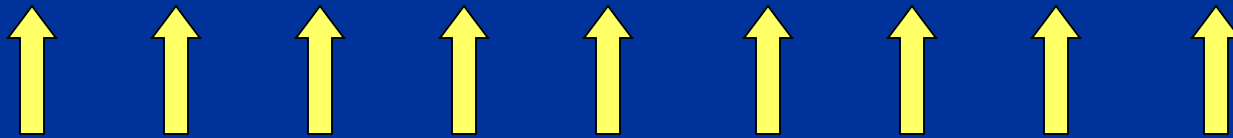
3



Búsqueda lineal

Búsqueda lineal (el elemento buscado no existe)

0	1	2	3	4	5	6	7	8
5	6	7	3	4	9	0	2	1



8

Índice devuelto:

-1



Busqueda Secuencial, Pseudocódigo

Procedimiento busquedaSecuencial (lista, elemento)

Inicializamos las variables

i=primero

encontrado=falso

Entramos en un bucle y permanecemos en él mientras no se cumplan las siguientes condiciones:

- Que no encontremos el elemento
- Que no alcancemos el final de array

Si alguna de ellas no se cumple, se finalizará el bucle

Dentro del bucle comparamos el **elemento** con la **posición i** si coincide **encontrado** será cierto

mientras (elemento no encontrado y $i \leq$ ultimo)

 si elemento = lista [i] entonces

 encontrado = cierto

 si-no

 incrementar i

fin.-mientras

Fuera del bucle, si se encuentra, devuelvo la posición, si no, devuelvo -1

 Si encontrado entonces

 Devolver i

 Si-no
Raquel Hijón, Neira
Devolver -1



Busqueda Secuencial Java

```
public static int secuencial(int[] vector, int elem) {  
    //recibe una matriz no ordenada, un elemento a buscar  
    // devuelve la posicion del elencnto o -1 si no está  
    int i;  
    boolean encontrado;  
    i = 0;  
    encontrado = false;  
    while (i < vector.length && !encontrado) {  
        if (elem == vector[i]) {  
            encontrado = true;  
        } else {  
            i++;  
        }  
    }  
    if (encontrado) {  
        return i;  
    } else {  
        return -1;  
    }  
}
```



Ejemplo 5.16 (Cont.)

```
public static int secuencial2(int[] vector, int elem) {  
    //recibe una matriz no ordenada, un elemento a buscar  
    // devuelve la posicion del elemento o -1 si no está  
    boolean encontrado;  
    int posicion;  
    int devolver = -1;  
    posicion = -1;  
    encontrado = false;  
  
    do {  
        posicion++;  
        if (vector[posicion] == elem) {  
            encontrado = true;  
            devolver = posicion;  
        }  
    } while (posicion < (vector.length-1) && !encontrado); // no se ha encontrado  
    return devolver;  
}
```




5.5.1. Algoritmos de búsqueda

Búsqueda secuencial o lineal en un array ordenado:

- ✓ Precondición: los elementos del array están ordenados de forma creciente (decreciente).
 - Se comienza con la primera componente y se va avanzando de forma secuencial hasta que:
 - Se encuentra el valor buscado, o
 - Se llega a una componente con un valor mayor que el buscado (orden ascendente), o
 - Se llega al final del array sin encontrarlo.



Búsqueda lineal

Búsqueda lineal en un array ordenado (el elemento buscado no existe)

0	1	2	3	4	5	6	7
1	2	3	4	5	9	10	12

↑ ↑ ↑ ↑ ↑

8

Índice devuelto:

-1



Ejemplo 5.17

```
public static int BusSecOrd(int[] vector, int elem) {  
    //recibe una matriz no ordenada, un elemento a buscar  
    // devuelve la posicion del elencnto o -1 si no está  
  
    int posicion = -1;  
    do {  
        posicion++;  
    } while ((posicion < vector.length-1) && (vector[posicion] < elem)); // no se ha encon  
  
    if (vector[posicion] == elem) {  
        return posicion;  
    } else {  
        return -1;  
    }  
}
```



5.5.1. Algoritmos de búsqueda

Búsqueda binaria o dicotómica

- ✓ Precondición: los valores de los elementos del array están ordenados (Creciente/Decreciente).
- ✓ La búsqueda binaria divide el array en dos mitades y determina si ***elemBuscado*** está en la primera o en la segunda mitad.
- ✓ Continúa dividiendo el espacio de búsqueda en mitades hasta encontrar el elemento o determinar que no está.
- ✓ La búsqueda binaria es el método utilizado para buscar en un diccionario, listín telefónico, etc.



5.5.1. Algoritmos de búsqueda

Fases:

1. Comparar ***elemBuscado*** con el elemento ***central*** del array.
2. Si es igual, la búsqueda ha terminado.
3. Si no es igual, se busca en la mitad adecuada del array (descartando la búsqueda en la otra mitad):
 1. La primera mitad, si ***elemBuscado*** < ***vector[central]***
 2. La segunda mitad, si ***elemBuscado*** > ***vector[central]***
4. Se vuelve al paso 1. la mitad elegida.

Este proceso se repite hasta que se encuentra ***elemBuscado*** o la mitad elegida está vacía o tiene un solo elemento.



Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9

2

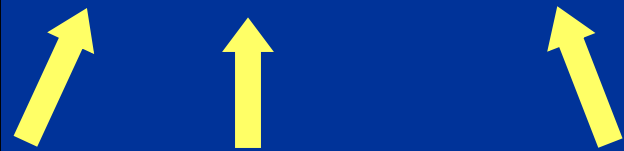




Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9

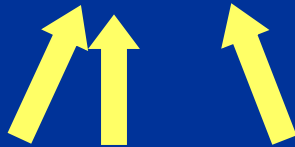
2





Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9



2

Índice devuelto:

2



Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9

4

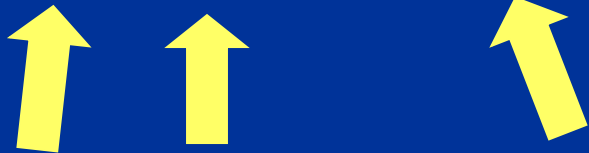




Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9

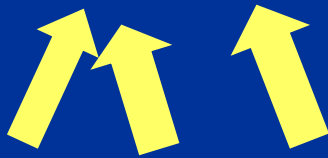
4





Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9



4



Búsqueda binaria o dicotómica

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	7	8	9



4

Índice devuelto:

-1



Busqueda Binaria, Pseudocódigo

Procedimiento `busquedaBinaria (lista, elemento)`

Mayor y menor serán índices (superior e inferior) de la lista o sublista donde se busca el elemento)

central será la posición que se encuentra en el centro (entre mayor y menor)

Inicializamos las variables

`menor=0;`

`mayor= lista.length`

`Encontrado= falso`

Entramos en un bucle y permanecemos en él mientras no se cumplan las siguientes condiciones:

- Que no encontremos el elemento
- Que no se crucen los índices mayor y menor

mientras (no encontrado) y (mayor \geq menor) hacer

Dentro del bucle se hará:

(central = (mayor + menor) div 2) // comprobará si elemento buscado coincide con el central

Si lista [central] = elemento entonces

encontrado= cierto



Busqueda Binaria, Pseudocódigo

Si el elemento no coincide con el central, mirar en que sublista buscar

Si-no

 si elemento > lista [central] entonces

 menor = central +1

 si no

 mayor = central -1

 fin si

fin si

Al salir del bucle, hay que ver cual ha sido el motivo: si hemos encontrado el elemento o bien se han cruzado los índices mayor y menor. En el primer caso, la posición buscada es la del índice central, y en el segundo caso, el elemento no está en el array.

Si encontrado entonces

 posicion = central

Si no

 posición = menor

Fin si



Búsqueda binaria, implementación

```
public static int BusBinaria(int[] vector, int elem) {  
    //recibe una matriz ordenada, un elemento a buscar  
    //devuelve la posición del elemento o -1 si no está  
    int i;  
    int menor = 0, mayor = vector.length-1, medio = 0;  
    boolean encontrado;  
    encontrado = false;  
    while (!encontrado && mayor >= menor) {  
        medio = (mayor + menor) / 2;  
        if (elem == vector[medio]) {  
            encontrado = true;  
        } else if (elem > vector[medio]) // buscar la mitad superior  
        {  
            menor = medio + 1;  
        } else // buscar en la mitad inferior  
        {  
            mayor = medio - 1;  
        }  
    }  
    if (encontrado) {  
        return medio;  
    } else {  
        return -1;  
    }  
}
```



5.5.2. Algoritmos de ordenación

- ✓ Mecanismo que permite ordenar los elementos de un array.
- ✓ Existen numerosos algoritmos de ordenación. Clasificándoles por la forma en que la realizan. Tendríamos algoritmos de:
 - Intercambio
 - Fusión



5.5.2. Algoritmos de ordenación

Algoritmos de ordenación por intercambio

- ✓ Los algoritmos de intercambio se caracterizan por intercambiar pares de elementos del array hasta conseguir su ordenación.
- ✓ Los más conocidos son:
 - Selección directa
 - Inserción directa
 - Intercambio directo (burbuja)



5.5.2. Algoritmos de ordenación

Intercambio directo:

- ✓ También llamado **método** de la **burbuja**.
- ✓ Eficiente para conjuntos pequeños de datos.

Resumen

- ✓ Recorrer el array, buscando el menor elemento, desde la última posición hasta la actual. Una vez encontrado, se sitúa en la posición actual.
- ✓ Para ello, se intercambian valores adyacentes siempre que estén colocados en orden decreciente.



5.5.2. Algoritmos de ordenación

Intercambio directo:

Paso 1. Situar el elemento mayor en la última posición del array.

Paso 2. Situar el segundo elemento mayor en la penúltima posición del array.

Paso i. Se repite el proceso para las posiciones intermedias.

Paso n-1. Se comparan los valores de las dos primeras posiciones, situando el n-1 mayor en la segunda posición.



Ordenación por intercambio directo

5 6 7 3 4 9 0 2 1





Ordenación por intercambio directo

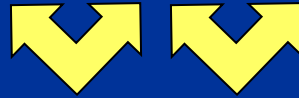
5 6 3 7 4 9 0 2 1





Ordenación por intercambio directo

5 6 3 4 7 9 0 2 1





Ordenación por intercambio directo

5 6 3 4 7 0 9 2 1





Ordenación por intercambio directo

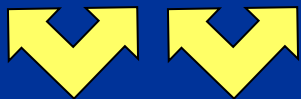
5 6 3 4 7 0 2 9 1





Ordenación por intercambio directo

5 6 3 4 7 0 2 1 9





Ordenación por intercambio directo

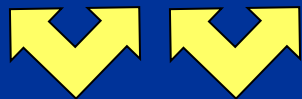
5 3 6 4 7 0 2 1 9





Ordenación por intercambio directo

5 3 4 6 7 0 2 1 9





Ordenación por intercambio directo

5 3 4 6 0 7 2 1 9





Ordenación por intercambio directo

5 3 4 6 0 2 7 1 9





Ordenación por intercambio directo

5 3 4 6 0 2 1 7 9



Ordenación por intercambio directo

5 3 4 6 0 2 1 7 9





Ordenación por intercambio directo

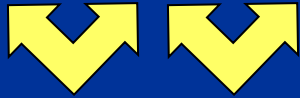
3 5 4 6 0 2 1 7 9





Ordenación por intercambio directo

3 4 5 6 0 2 1 7 9





Ordenación por intercambio directo

3 4 5 0 6 2 1 7 9





Ordenación por intercambio directo

3 4 5 0 2 6 1 7 9





Ordenación por intercambio directo

3 4 5 0 2 1 6 7 9



Ordenación por intercambio directo

3 4 5 0 2 1 6 7 9





Ordenación por intercambio directo

3 4 0 5 2 1 6 7 9





Ordenación por intercambio directo

3 4 0 2 5 1 6 7 9





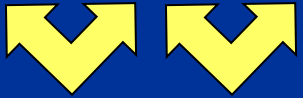
Ordenación por intercambio directo

3 4 0 2 1 5 6 7 9



Ordenación por intercambio directo

3 4 0 2 1 5 6 7 9





Ordenación por intercambio directo

3 0 4 2 1 5 6 7 9





Ordenación por intercambio directo

3 0 2 4 1 5 6 7 9





Ordenación por intercambio directo

3 0 2 1 4 5 6 7 9



Ordenación por intercambio directo

3 0 2 1 4 5 6 7 9





Ordenación por intercambio directo

0 3 2 1 4 5 6 7 9





Ordenación por intercambio directo

0 2 3 1 4 5 6 7 9





Ordenación por intercambio directo

0 2 1 3 4 5 6 7 9



Ordenación por intercambio directo

0 2 1 3 4 5 6 7 9





Ordenación por intercambio directo

0 1 2 3 4 5 6 7 9



Ordenación por intercambio directo

0 1 2 3 4 5 6 7 9





Ordenación por intercambio directo

0 1 2 3 4 5 6 7 9



Algoritmo de la Burbuja

- pseudocódigo

Se necesitan los siguientes parámetros:

Vector: el array que contiene la lista de elementos a ordenar

Se necesitan las siguientes variables:

i: nos indicará el número de pasada

j: será el índice para recorrer el array

Aux: variable auxiliar para realizar el intercambio de los elementos del array

Se utilizará un bucle controlado por **i** que realizará **n-1** iteraciones, tantas como el número de pasadas necesarias en el peor caso.

Desde $i \leftarrow$ hasta $n-1$ hacer

Para cada iteración del bucle, se comparará cada elemento, comenzando por el primero, con el siguiente, de forma que si no están ordenados se intercambian utilizando una variable auxiliar.

desde $j \leftarrow 0$ hasta $n-1-i$ hacer

// hasta el penúltimo elemento

si $L[j] > L[j+1]$ entonces

// intercambio

$Aux \leftarrow L[j]$

$L[j] \leftarrow L[j+1]$

$L[j+1] \leftarrow aux$

fin-si

fin-desde

En la primera pasada, el elemento de más valor queda colocado en la última posición, en la segunda pasada, el segundo elemento de más valor queda en la penúltima posición y así sucesivamente...



Intercambio Directo (Burbuja)

```
public static void burbujaAsc(int[] vector) {  
    // ordena ascendentemente por el algoritmo de la burbuja un vector  
    int aux;  
    for (int i = 1; i < vector.length; i++) {  
        for (int j = 0; j < vector.length - i; j++) {  
            if (vector[j] > vector[j + 1]) {  
                aux = vector[j];  
                vector[j] = vector[j + 1];  
                vector[j + 1] = aux;  
            }  
        }  
    }  
}
```



Intercambio Directo (Burbuja) ordenación descendente

```
public static void burbujaDes(int[] vector) {  
    // ordena descendente por el algoritmo de la burbuja un vector  
    int aux;  
    for (int i = 1; i < vector.length; i++) {  
        for (int j = 0; j < vector.length - i; j++) {  
            if (vector[j] < vector[j + 1]) {  
                aux = vector[j];  
                vector[j] = vector[j + 1];  
                vector[j + 1] = aux;  
            }  
        }  
    }  
}
```



Llamadas a los dos métodos previos

```
public static void main(String[] args) throws IOException {  
    int[] vector = {9, 4, 6, 7, 8};  
    burbujaAsc(vector);  
    System.out.println("el array ordenado ascendentemente es ");  
    for (int i = 0; i < vector.length; i++) {  
        System.out.println("La posicion " + i + " es " + vector[i]);  
    }  
    burbujaDes(vector);  
    System.out.println("el array ordenado descendientemente es ");  
    for (int i = 0; i < vector.length; i++) {  
        System.out.println("La posicion " + i + " es " + vector[i]);  
    }  
}
```

Salida:

```
el array ordenado ascendentemente es  
La posicion 0 es 4  
La posicion 1 es 6  
La posicion 2 es 7  
La posicion 3 es 8  
La posicion 4 es 9  
el array ordenado descendientemente es  
La posicion 0 es 9  
La posicion 1 es 8  
La posicion 2 es 7  
La posicion 3 es 6  
La posicion 4 es 4
```