UNIVERSIDAD DE ALMERÍA

Grado en Ingeniería Informática
Metodología de la programación
2013

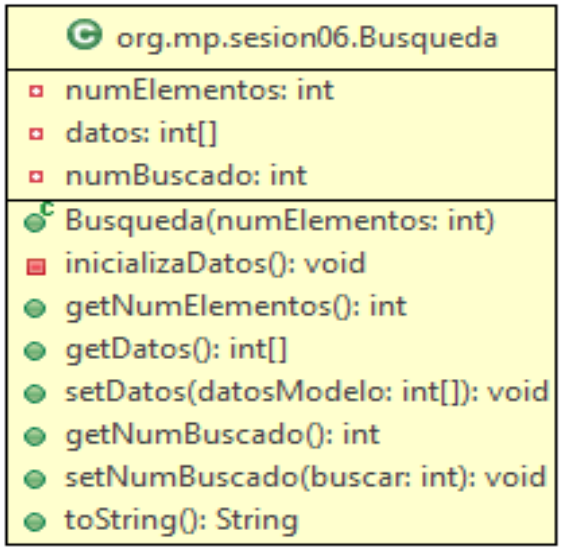
Ordenación y búsqueda

- Búsqueda binaria
- Ordenación. MergeSort
- Arrays, Collections. Comparator

La **búsqueda** es un proceso que permite determinar si un valor específico está en un array

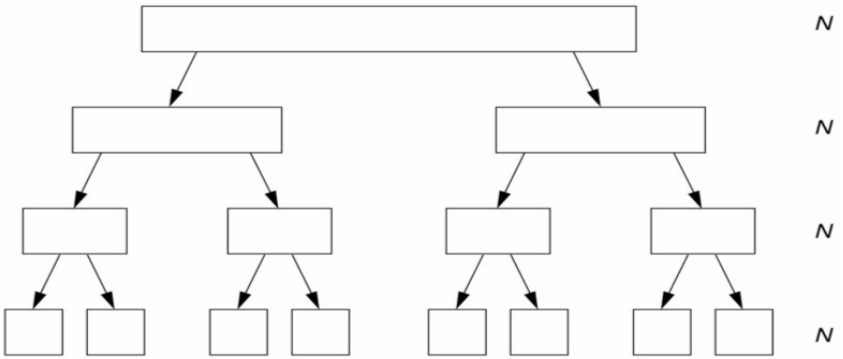
```
public static int busquedaLineal(Busqueda busqueda) throws ElementoNoEncontradoException{

    int buscado;
    buscado = busqueda.getNumBuscado();
    boolean encontrado = false;
    int[] datos = busqueda.getDatos();
    for (int i = 0; i < datos.length; i++) {
        if (buscado==datos[i]){
            encontrado = true;
            break;
        }
    }
    if (!encontrado)
        throw new ElementoNoEncontradoException("No encontrado");
    return buscado;
}
```

***O*(N)**



Búsqueda binaria recursiva.
Requisito imprescindible que el array esté ordenado

Se resuelve dividiendo el problema en problemas menores.
O lo que es lo mismo, utilizando al técnica recursiva divide y vencerás.




```
public static int busquedaBinariaRec(Busqueda busqueda)
        throws ElementoNoEncontradoException {
    int inicio = 0;
    int[] datos = busqueda.getDatos();
    int fin = datos.length - 1;
    int buscado = busqueda.getNumBuscado();
    busquedaBinariaRec(datos, buscado, inicio, fin);
    return buscado;
}
```

***O*(logN)**

La ordenación de un vector de N elementos

“Dado un vector A[1..n] de n elementos. Ordenar estos elementos por orden ascendente”.

Nota: Sólo pueden ser ordenados objetos que implementen la interfaz java.lang.**Comparable**.

Ordenación por inserción

• Es el método de ordenación más simple. Su implementación utiliza dos bucles cada uno de los cuales puede realizar N iteraciones. Por lo tanto el algoritmo de ordenación por inserción es $\boldsymbol{O(N^2)}$, es decir cuadrático.

➔El peor de los casos se da si el vector viene ordenado en orden inverso.

➔El mejor de los casos se da si el vector ya viene ordenado, entonces seria de orden lineal $\boldsymbol{O(N)}$

```java
private static void ordenacionPorInsercion(Comparable[] a, int izq, int der) {
    int j;
    Comparable tmp;
    for (int p = izq + 1; p <= der; p++) {
        tmp = a[p];
        for (j = p; j > izq && (tmp.compareTo(a[j - 1]) < 0); j--){
            a[j] = a[j - 1];
        }
        a[j] = tmp;
    }
}
```

Funcionamiento del algoritmo de ordenación por inserción

Array Position	0	1	2	3	4	5
Initial State	8	5	9	2	6	3
After a[0..1] is sorted	5	8	9	2	6	3
After a[0..2] is sorted	5	8	9	2	6	3
After a[0..3] is sorted	2	5	8	9	6	3
After a[0..4] is sorted	2	5	6	8	9	3
After a[0..5] is sorted	2	3	5	6	8	9

Las celdillas de fondo oscuro señalan el área ordenada.

Las celdillas de fondo más claro marcan los intercambios

Array Position	0	1	2	3	4	5
Initial State	8	**5**				
After a[0..1] is sorted	5	8	**9**			
After a[0..2] is sorted	5	8	9	**2**		
After a[0..3] is sorted	2	5	8	9	**6**	
After a[0..4] is sorted	2	5	6	8	9	**3**
After a[0..5] is sorted	2	3	5	6	8	9

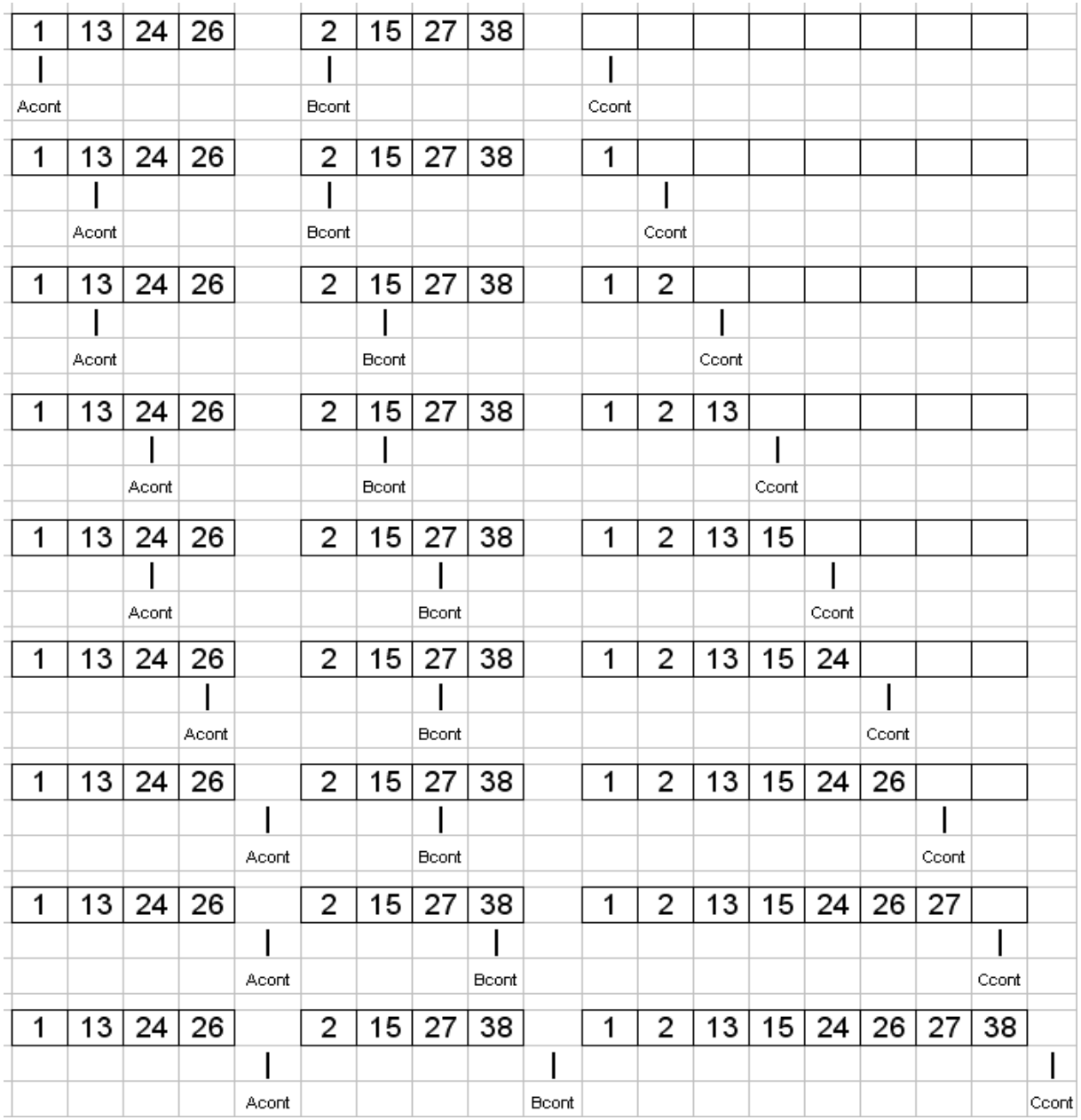
MergeSort

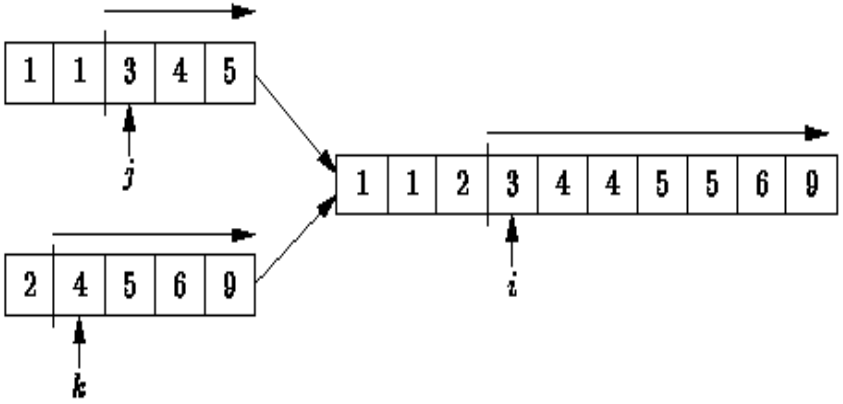
El algoritmo Mergesort utiliza la técnica recursiva divide y vencerás para obtener un tiempo de ejecución de O(NlogN).

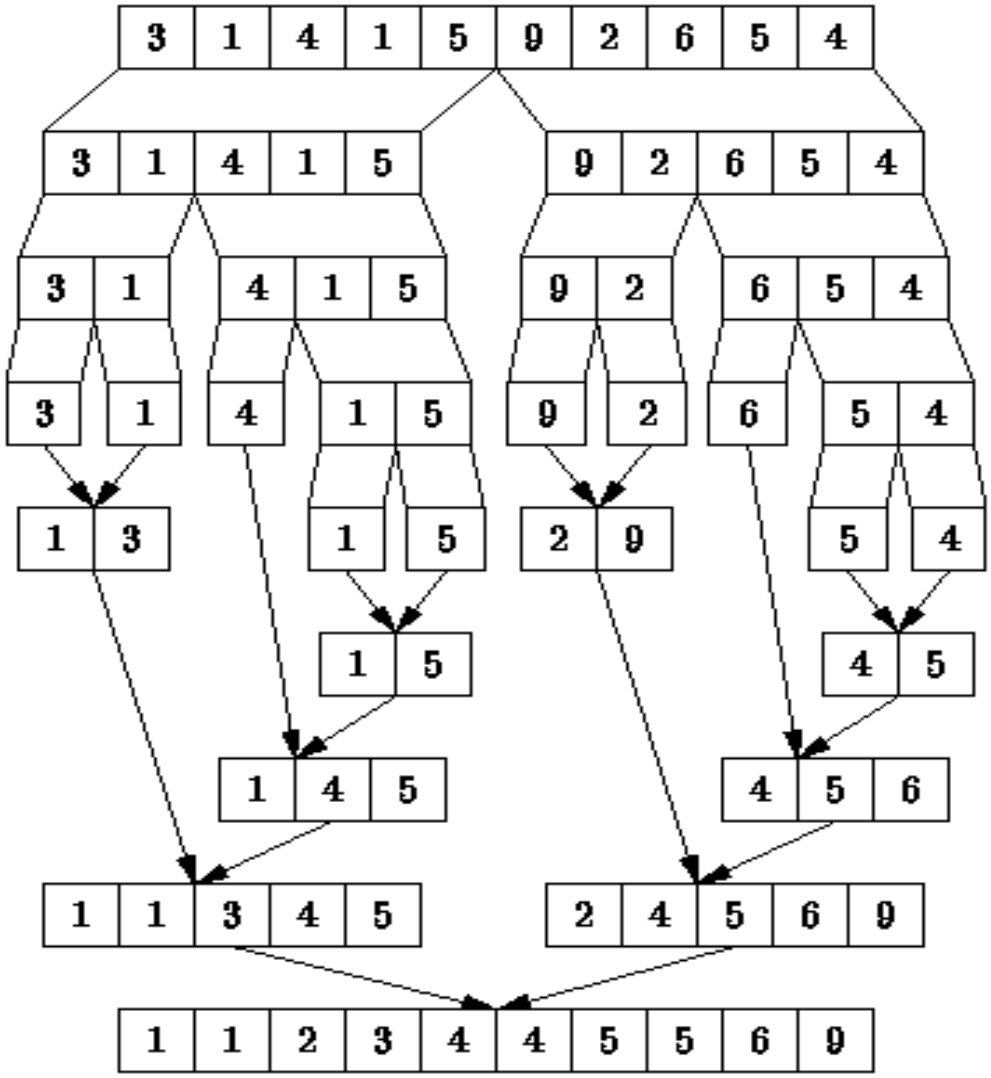
El algoritmo consta de tres pasos:

1. Si el número de elementos a ordenar es 0 o 1, acabar.
2. Ordenar recursivamente las dos mitades del vector.
3. Mezclar las dos mitades ordenadas en un vector ordenado.

La mezcla lineal de vectores ordenados:




1 1 3 4 5
j
2 4 5 6 9
k
1 1 2 3 4 4 5 5 6 9
i

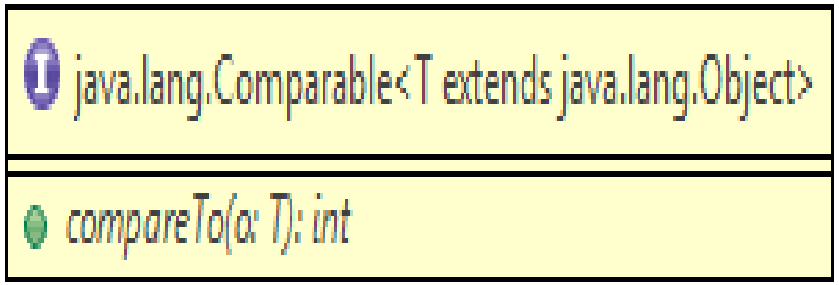

3 1 4 1 5 9 2 6 5 4
3 1 4 1 5
9 2 6 5 4
3 1
4 1 5
9 2
6 5 4
3
1
4
1 5
9
2
6
5 4
1 3
1 5
2 9
5 4
1 5
4 5
1 4 5
4 5 6
1 1 3 4 5
2 4 5 6 9
1 1 2 3 4 4 5 5 6 9

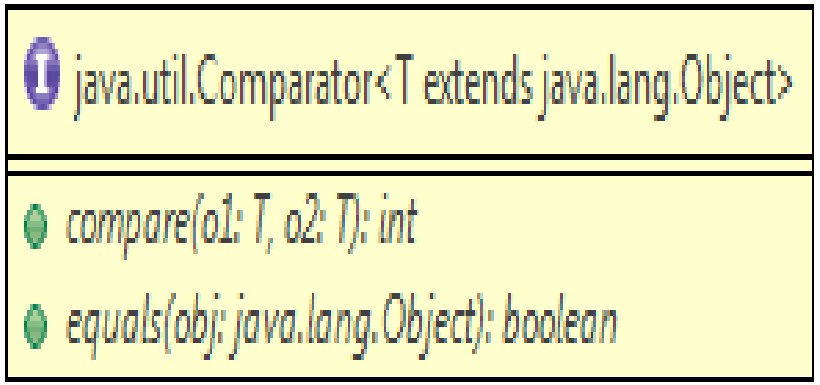
```java
public static void mergeSort(Comparable[] a) {
    Comparable[] vectorTemp = new Comparable[a.length];
    mergeSort(a, vectorTemp, 0, a.length - 1);
}
@SuppressWarnings("rawtypes")
private static void mergeSort(Comparable[] a, Comparable[] vectorTemp,
        int izq, int der) {
    if (izq < der) {
        int centro = (izq + der) / 2;
        mergeSort(a, vectorTemp, izq, centro);
        mergeSort(a, vectorTemp, centro + 1, der);
        mezclar(a, vectorTemp, izq, centro + 1, der);
    }
}
```

```java
private static void mezclar(Comparable[] a, Comparable[] vectorAux,
        int posIzq, int posDer, int posFin) {

    int finIzq = posDer - 1;
    int posAux = posIzq;
    int numElementos = posFin - posIzq + 1;
    while ((posIzq <= finIzq) && (posDer <= posFin)) {
        if (a[posIzq].compareTo(a[posDer]) <= 0)
            vectorAux[posAux++] = a[posIzq++];
        else
            vectorAux[posAux++] = a[posDer++];
    }
    // Copiar el resto de la primera mitad
    while (posIzq <= finIzq)
        vectorAux[posAux++] = a[posIzq++];
    // Copiar el resto de la segunda mitad
    while (posDer <= posFin)
        vectorAux[posAux++] = a[posDer++];
    // Copiar el vector temporal en el original
    for (int i = 0; i < numElementos; i++, posFin--)
        a[posFin] = vectorAux[posFin];
}
```

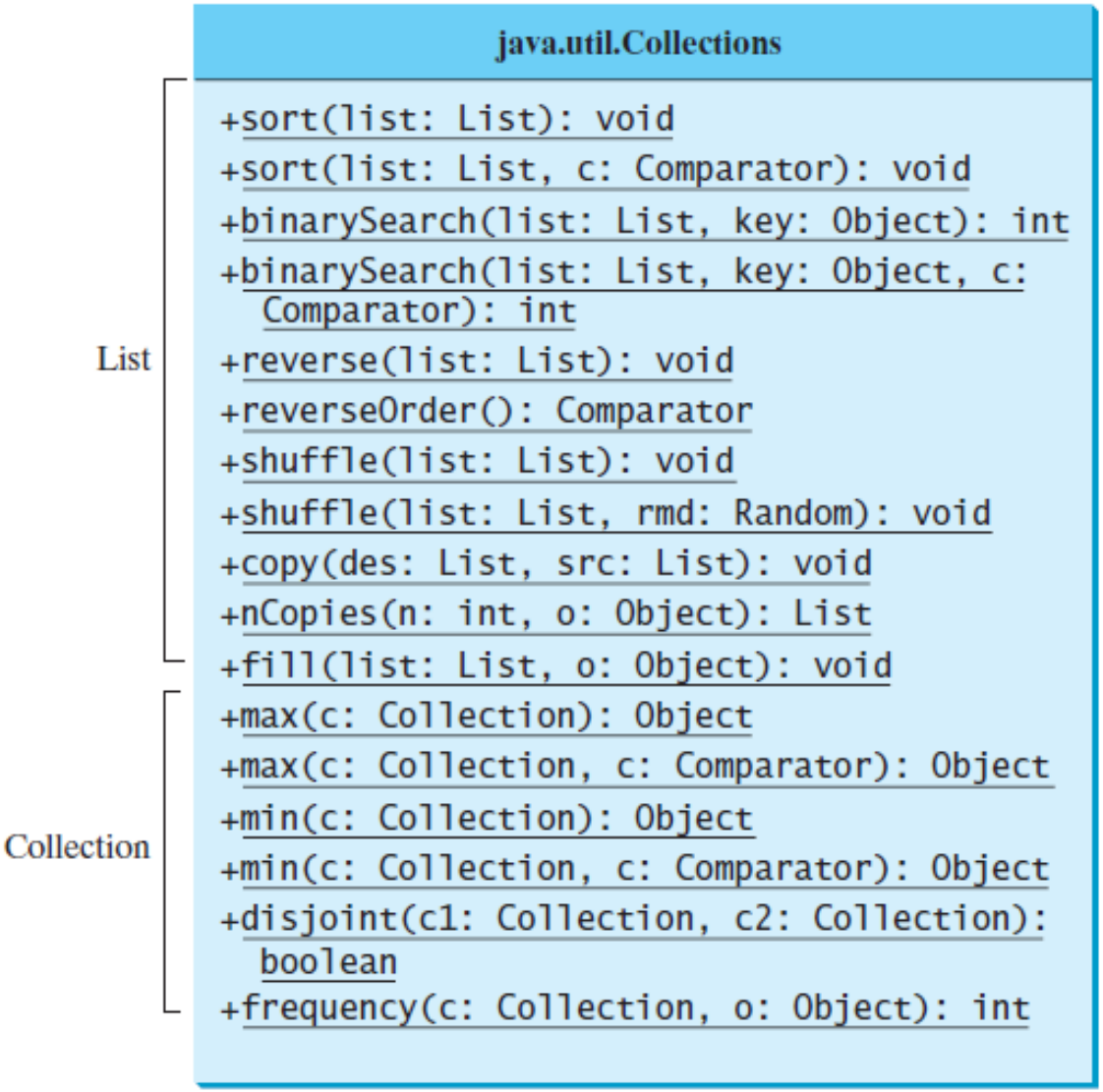
a	1	1	2	3	4	5	6	7
	24	13	26	1	2	27	38	15
	|				|			|
	posIzq				posDer			posFin
	numElementos=posFin-posIz +1							
vectorTemp	1	1	2	3	4	5	6	7
	24	13	26	1	2	27	38	15
	|			|				
	posAux			finIzq				

La interface java.lang.Comparator





La clase Collections, equivalente de Arrays




java.util.Arrays
sort(a: int[]): void
sort(a: int[], fromIndex: int, toIndex: int): void
sort(a: long[]): void
sort(a: long[], fromIndex: int, toIndex: int): void
sort(a: short[]): void
sort(a: short[], fromIndex: int, toIndex: int): void
sort(a: char[]): void
sort(a: char[], fromIndex: int, toIndex: int): void
sort(a: byte[]): void
sort(a: byte[], fromIndex: int, toIndex: int): void
sort(a: float[]): void
sort(a: float[], fromIndex: int, toIndex: int): void
sort(a: double[]): void
sort(a: double[], fromIndex: int, toIndex: int): void
sort(a: java.lang.Object[]): void
sort(a: java.lang.Object[], fromIndex: int, toIndex: int): void
sort(a: T[], c: java.util.Comparator<? super T>): void
sort(a: T[], fromIndex: int, toIndex: int, c: java.util.Comparator<? super T>): void
binarySearch(a: long[], key: long): int
binarySearch(a: long[], fromIndex: int, toIndex: int, key: long): int
binarySearch(a: int[], key: int): int
binarySearch(a: int[], fromIndex: int, toIndex: int, key: int): int
binarySearch(a: short[], key: short): int
binarySearch(a: short[], fromIndex: int, toIndex: int, key: short): int
binarySearch(a: char[], key: char): int
binarySearch(a: char[], fromIndex: int, toIndex: int, key: char): int
binarySearch(a: byte[], key: byte): int
binarySearch(a: byte[], fromIndex: int, toIndex: int, key: byte): int
binarySearch(a: double[], key: double): int
binarySearch(a: double[], fromIndex: int, toIndex: int, key: double): int
binarySearch(a: float[], key: float): int
binarySearch(a: float[], fromIndex: int, toIndex: int, key: float): int
binarySearch(a: java.lang.Object[], key: java.lang.Object): int
binarySearch(a: java.lang.Object[], fromIndex: int, toIndex: int, key: java.lang.Object): int
binarySearch(a: T[], key: T, c: java.util.Comparator<? super T>): int
binarySearch(a: T[], fromIndex: int, toIndex: int, key: T, c: java.util.Comparator<? super T>): int
equals(a: long[], a2: long[]): boolean
equals(a: int[], a2: int[]): boolean
equals(a: short[], a2: short[]): boolean
equals(a: char[], a2: char[]): boolean
equals(a: byte[], a2: byte[]): boolean
equals(a: boolean[], a2: boolean[]): boolean
equals(a: double[], a2: double[]): boolean
equals(a: float[], a2: float[]): boolean
equals(a: java.lang.Object[], a2: java.lang.Object[]): boolean
fill(a: long[], val: long): void
fill(a: long[], fromIndex: int, toIndex: int, val: long): void
fill(a: int[], val: int): void
fill(a: int[], fromIndex: int, toIndex: int, val: int): void
fill(a: short[], val: short): void
fill(a: short[], fromIndex: int, toIndex: int, val: short): void
fill(a: char[], val: char): void
fill(a: char[], fromIndex: int, toIndex: int, val: char): void
fill(a: byte[], val: byte): void
fill(a: byte[], fromIndex: int, toIndex: int, val: byte): void
fill(a: boolean[], val: boolean): void
fill(a: boolean[], fromIndex: int, toIndex: int, val: boolean): void
fill(a: double[], val: double): void
fill(a: double[], fromIndex: int, toIndex: int, val: double): void
fill(a: float[], val: float): void
fill(a: float[], fromIndex: int, toIndex: int, val: float): void
fill(a: java.lang.Object[], val: java.lang.Object): void
fill(a: java.lang.Object[], fromIndex: int, toIndex: int, val: java.lang.Object): void

.....

Ejemplos

```java
public class EjemploSortStringArray {

    public static void main(String args[]) {
        String[] nombres = new String[] { "Juan", "ana", "Cristobal", "vanesa",
                "Miguel", "Luis" };

        Arrays.sort(nombres);
        System.out.println("String array ordenado (sin tener en cuenta mayusculas)");
        for (int i = 0; i < nombres.length; i++) {
            System.out.println(nombres[i]);
        }

        Arrays.sort(nombres, String.CASE_INSENSITIVE_ORDER);
        System.out.println("String array ordenado (teniendo en cuenta mayusculas)");

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

```
String array ordenado (sin tener en cuenta mayusculas)
Cristobal
Juan
Luis
Miguel
ana
vanesa
String array ordenado (teniendo en cuenta mayusculas)
ana
Cristobal
Juan
Luis
Miguel
vanesa
```

```java
public class EjemploBinarySearchArrayList {

  @SuppressWarnings({ "rawtypes", "unchecked" })
public static void main(String[] args) {

    ArrayList arrayList = new ArrayList();
    arrayList.add("1");
    arrayList.add("4");
    arrayList.add("2");
    arrayList.add("5");
    arrayList.add("3");

    Collections.sort(arrayList);
    System.out.println("ArrayList ordenado : " + arrayList);

    int indice = Collections.binarySearch(arrayList,"4");
    System.out.println("Elemento encontrado en : " + indice);
  }
}
```

```
ArrayList ordenado : [1, 2, 3, 4, 5]
Elemento encontrado en : 3
```

```java
public class EjemploShuffleElementsOfArrayList {

  @SuppressWarnings({ "rawtypes", "unchecked" })
public static void main(String[] args) {
    ArrayList arrayList = new ArrayList();
    arrayList.add("A");
    arrayList.add("B");
    arrayList.add("C");
    arrayList.add("D");
    arrayList.add("E");

    System.out.println("Antes desordenar: " + arrayList);
    Collections.shuffle(arrayList);
    System.out.println("Después de desordenar : " + arrayList);
    Collections.shuffle(arrayList);
    System.out.println("Después de desordenar otra vez : " + arrayList);
  }
}
```

```
Antes desordenar: [A, B, C, D, E]
Después de desordenar : [B, A, C, D, E]
Después de desordenar otra vez : [E, A, B, D, C]
```

```java
public class EjemploSortArrayListEnOrdenDescendente {

  @SuppressWarnings({ "rawtypes", "unchecked" })
public static void main(String[] args) {

    ArrayList arrayList = new ArrayList();
    arrayList.add("A");
    arrayList.add("B");
    arrayList.add("C");
    arrayList.add("D");
    arrayList.add("E");

    Comparator comparator = Collections.reverseOrder();
    System.out.println("Antes de ordenar  : "
    Collections.sort(arrayList,comparator);
    System.out.println("Antes de ordenar en orden descendente : " + arrayList);

  }
}
```

```
Antes de ordenar  : [A, B, C, D, E]
Antes de ordenar en orden descendente : [E, D, C, B, A]
```

```java
public class EjemploShuffleElementsOfArrayList {

  @SuppressWarnings({ "rawtypes", "unchecked" })
public static void main(String[] args) {
    ArrayList arrayList = new ArrayList();
    arrayList.add("A");
    arrayList.add("B");
    arrayList.add("C");
    arrayList.add("D");
    arrayList.add("E");

    System.out.println("Antes desordenar: " + arrayList);
    Collections.shuffle(arrayList);
    System.out.println("Después de desordenar : " + arrayList);
    Collections.shuffle(arrayList);
    System.out.println("Después de desordenar otra vez : " + arrayList);
  }
}
```

```
Antes desordenar: [A, B, C, D, E]
Después de desordenar : [B, A, C, D, E]
Después de desordenar otra vez : [E, A, B, D, C]
```

¡Muchas Gracias y Fin!