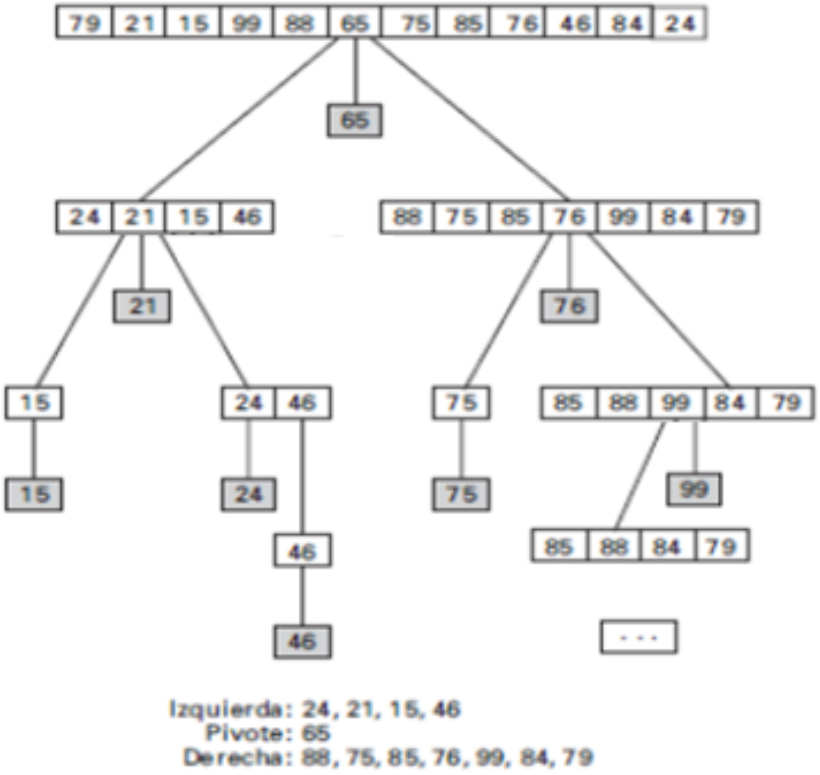
ALGORITMOS DE ORDENACIÓN DE ARRAYS

Intercambio: Consiste en comparar el primer valor con el resto de las posiciones posteriores, cambiando el valor de las posiciones en caso de que el segundo sea menor que el primero comparado. Después la segunda posición con el resto de posiciones posteriores y así sucesivamente hasta que el array se ordena. Se realizan muchas pasadas sobre el array, por lo que es lento y costoso.

Burbuja: Consiste en comparar el primero con el segundo, si el segundo es menor que el primero se intercambian los valores. Después el segundo con el tercero y así sucesivamente, cuando no haya ningún intercambio, el array estará ordenado. Lo peor de este método de ordenación, es que tiene una complejidad de **$O(n^2)$** haciendo que cuantos más valores a ordenar tengamos, mayor tiempo tardará en ordenar.

Quicksort: Consiste en ordenar un array mediante un pivote, que tomaremos como un punto intermedio en el array. Es como si se ordenaran pequeños trozos del array, haciendo que a la izquierda estén los menores a ese pivote y en la derecha los mayores a este. Después se vuelve a calcular el pivote de trozos de listas. Usa recursividad. Le pasamos el array, su posición inicial y su posición final como parámetro. Tiene una complejidad de **O(n log2 n),** haciendo que mejore el rendimiento aun teniendo muchos valores que ordenar.



Sort: Consiste en usar el método **sort** de **java.util.Arrays**.

Para ejecutarlo escribimos:

Arrays.sort (array a ordenar);

Simplemente insertamos como parámetro el array que queremos ordenar.

Por ejemplo, dado el siguiente array de Strings:

```
String [] nombres = {"juan", "pedro", "ana", "maria", "felipe", "luis", "eduardo"};
```

Con el método sort, se ordena ascendentemente con la instrucción:

```
Arrays.sort(nombres);
```

Para ordenar un array de forma descendente (de mayor a menor) hay que indicarlo utilizando el método **reverseOrder()** de la clase **Collections**.

Para utilizar reverseOrder es necesario incluir el import:

```
import java.util.Collections;
```

Por ejemplo, para ordenar el array nombres de forma descendente escribimos la instrucción Arrays.sort de la siguiente forma:

```
Arrays.sort(nombres, Collections.reverseOrder());
```

Con Arrays.sort podemos ordenar arrays de cualquier tipo de datos. Por ejemplo, para ordenar un array de enteros:

```
int [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};
Arrays.sort(numeros);
// Mostrarlo ordenado
for (int n : numeros) {
        System.out.println(n);
}
```

Collections.reverseOrder() solo funciona para arrays de objetos. Por este motivo si queremos ordenar de forma descendente arrays de tipos de datos simples debemos utilizar la **clase envolvente** equivalente al tipo de dato básico.

Por ejemplo, para ordenar un array de enteros de forma descendente hay que declararlo de tipo Integer en lugar de int.

```
Integer [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};
Arrays.sort(numeros, Collections.reverseOrder());
for (int n : numeros) {
        System.out.println(n);
}
```

ALGORITMOS DE BÚSQUEDA BINARIA EN ARRAYS

Es un algoritmo de búsqueda que encuentra la posición de un valor en un array ordenado.

Compara el valor con el elemento que tengamos en medio del array, si no son iguales, la mitad en la cual el valor no puede estar es eliminada y la búsqueda continúa en la mitad restante hasta que el valor se encuentre.

Tienes un ejemplo en la implementación del algoritmo QuickSort de los ejemplos.