UT 5: Algoritmos de ordenación

1. Algoritmo de ordenación de la burbuja

Consiste en ir colocando el elemento mayor en la última posición (burbujea).

```
private static void bubble(Integer[] listado) {
    // El bucle comienza en 1 puesto vamos a comprobar el tamaño
    // hasta que solo quede un elemento, en cuyo caso, ya estaría colocado.
    for (int i = 1; i < listado.length; i++) {
        boolean cambios = false;
        for (int j = 0; j < listado.length - 1; j++) {
            if (listado[j] > listado[j + 1]) {
                // Intercambio
                int aux = listado[j];
                listado[j] = listado[j + 1];
                listado[j + 1] = aux;
                cambios = true;
            }
        }
        if (!cambios) {
            break;
        }
    }
}
```

2. Algoritmo de ordenación por selección

Busca el elemento menor e intercambia su posición con el elemento actual.

```
private static void selection(Integer[] listado) {
    for (int i = 0; i < listado.length - 1; i++) {
        int pos_menor = i;
        // Fase de búsqueda del menor
        for (int j = i + 1; j < listado.length; j++) {
            if (listado[j] < listado[pos_menor]) {
                pos_menor = j;
            }
        }
        // Fase de intercambio
        int aux = listado[i];
        listado[i] = listado[pos_menor];
        listado[pos_menor] = aux;
    }
}
```

3. Algoritmo de ordenación por inserción

Inserta el elemento actual en su posición correspondiente, es decir, desplaza el resto de elementos mayores, una posición a la derecha.

```
private static void insertion(Integer[] listado) {
    // Comenzamos en el segundo elemento, para comparar con el de la izquierda
    for (int i = 1; i < listado.length; i++) {
        int aux = listado[i];
        int j = i - 1; // Permite comparar desde un elemento menor al actual
        // Primero se comprueba el índice, por si es negativo, que no dé error.
        while (j >= 0 && listado[j] > aux) {
            listado[j + 1] = listado[j];
            j--;
        }
        listado[j + 1] = aux; // Se cubre el "hueco" que se ha ido dejando
    }
}
```

4. Quicksort

Se basa en la técnica divide y vencerás, que consiste en ir subdividiendo el *array* en arrays más pequeños, y ordenar éstos. Para hacer esta división, se toma un valor del *array* como pivote, y se mueven todos los elementos menores que este pivote a su izquierda, y los mayores a su derecha. A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el *array*.

Después de elegir el pivote se realizan dos búsquedas:

- Una de izquierda a derecha, buscando un elemento mayor que el pivote
- Otra de derecha a izquierda, buscando un elemento menor que el pivote.

Cuando se han encontrado los dos elementos anteriores, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran.

```
private static void quicksort(Integer[] lista, int indice_inf, int indice_sup) {
    if (indice_inf < indice_sup) {
        int indice_pivote = partition(lista, indice_inf, indice_sup);
        // Ordenación de la lista izquierda
        quicksort(lista, indice_inf, indice_pivote - 1);
        // Ordenación de la lista derecha
        quicksort(lista, indice_pivote + 1, indice_sup);
    }
}

private static int partition(Integer[] lista, int i, int s) {
    // Se elige el primer elemento como pivote.
    int indice_pivote = i;
    while (i < s) {
        // Se buscan elementos superiores al pivote
        while (lista[i] <= lista[indice_pivote] && i < s) {
            i++; // El bucle para cuando se encuentra un elemento mayor al privote
        }
        // Buscamos ahora elementos inferiores al pivote
        while (lista[s] > lista[indice_pivote]) {
            s--; // El bucle para cuando se encuentra un elemento menor al privote
        }
        // Fase de intercambio
        if (i < s) {
            int aux = lista[i];
            lista[i] = lista[s];
            lista[s] = aux;
        }
    }
    // Se intercambia el pivote por el primer elemento de la lista superiores,
    // de esta forma, el privote queda justo en el medio de las dos lista.
    int aux = lista[indice_pivote];
    lista[indice_pivote] = lista[s];
    lista[s] = aux;
    return s; // Se devuelve la posición del privote
}
```