Práctica 10
Ordenamientos

Estructuras de datos

Meta

Que el alumno comprenda e implemente algoritmos de ordenamiento.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Implementar algoritmos de ordenamiento en un lenguaje orientado a objetos
- Identificar los mejores y peores casos de un algoritmo de ordenamiento.

Desarrollo

En esta práctica se programarán objetos capaces de ordenar arreglos de objetos tipo `Comparable<T>`. Estos objetos pertenecerán a clases que implementarán la interfaz `IOrdenador<C>`. Observe que es una interfaz genérica. Cada clase representará a un algoritmo de ordenamiento y por lo tanto se llamará igual que él, pero con la terminación *Sorter* (ej. `BubbleSorter` para *BubbleSort*). De este modo programarás una clase por cada algoritmo. El ordenamiento se realizará de forma ascendente (de menor a mayor).

Los algoritmos de ordenamiento a implementar serán los siguientes:

- BubbleSort.
- SelectionSort.
- InsertionSort.
- MergeSort.
- QuickSort. En este caso se implementará tomando como pivote el primer elemento del arreglo, para que sea más fácil generar el peor caso.

Además para cada ordenamiento se debe implementar el método que devuelve un arreglo de enteros, el cual representará el peor caso, en términos de complejidad, para cada uno de los algoritmos mencionados previamente.

```java
/*
 * Codigo utilizado para el curso de Estructuras de Datos.
 * Se permite consultarlo para fines didacticos en forma personal.
 */
package ed.ordenamientos;

/**
 * Interfaz que representa las acciones que debe realizar cualquier objeto
 * capaz de ordenar los elementos en un arreglo.
 * @author blackzafiro
 */
public interface IOrdenador<C extends Comparable<C>> {

	/**
	 * Crea un arreglo nuevo con los elementos ordenados.
	 * @param a El arreglo cuyos elementos se quieren ordenar.
	 * @return Un arreglo nuevo, del mismo tipo de <code>a</code> pero con los
	 *         elementos en el orden dictado por <code>compareTo</code>.
	 */
	C[] ordena(C[] a);

	/**
	 * Devuelve un arreglo de enteros de tal manera que, si es ordenado con
	 * un objeto de esta clase, será el peor caso para la complejidad en tiempo.
	 * @param tam La longitud del arreglo a generar.
	 * @return Arreglo con el peor caso para el algoritmo de ordenamiento
	 *         implementado.
	 */
	int[] peorCaso(int tam);

	/**
	 * Intercambia indicadas en el arreglo los elementos en las posiciones.
	 * @param a
	 * @param i
	 * @param j
	 */
	default void swap(C[] a, int i, int j) {
		C temp = a[i];
		a[i] = a[j];
		a[j] = temp;
	}
}
```

PREGUNTAS

1. Explique cómo generó cada uno de los peores casos y por qué es el peor caso para ese algoritmo, además de mencionar el orden de la complejidad del peor caso.
2. Explique cuáles son los mejores casos para los mismos algoritmos y cuál es su complejidad.
3. ¿En qué algoritmos la complejidad en el peor y el mejor caso es la misma?¿Cuál es ésta?
4. ¿En qué algoritmos difiere? Mencione sus complejidades en el mejor y peor caso.

Forma de entrega

- Asegúrense de borrar todos los archivos generados, incluyendo archivos de respaldo usando `ant clean`.
- Copien el directorio `Practica10` dentro de un directorio llamado `<apellido1_apellido2>` donde `apellido1` es el primer apellido de un miembro del equipo y `apellido2` es el primer apellido del otro miembro. Por ejemplo: `marquez_vazquez`.
- Borren el directorio libs en la copia.
- Agreguen un archivo `reporte.pdf` dentro del directorio `Practica10` de la copia, con el nombre completo de los integrantes del equipo, las repuestas a las preguntas de la sección anterior y cualquier comentario que quieran hacer sobre la práctica.
- Compriman el directorio `<apellido1_apellido2>` creando

  `<apellido1_apellido2>.tar.gz`.

  Por ejemplo: `marquez_vazquez.tar.gz`.
- Suban este archivo en la sección correspondiente del aula virtual. Basta una entrega por equipo.

Forma de evaluación

Para su calificación final se tomarán en cuenta los aspectos siguientes:

70 %	Calificación generada por las pruebas automáticas. Usaremos nuestros archivos, por lo que si realizan modificaciones a sus copias, éstas no tendrán efecto al momento de calificar.
10 %	Reporte con respuestas a las preguntas.
10 %	Documentación completa y adecuada. Entrega en el formato requerido, sin archivos `.class`, respaldos, bibliotecas de `JUnit` u otros no requeridos.
10 %	Revisión manual del código, para verificar que se cumpla con las especificaciones, que no se haya copiado, etcétera.