# 15 | Ordenamientos

## 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.

## 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.