1. Objetivo

Identificar la estructura de los algoritmos de ordenamiento bubble sort y merge sort.

2. Descripción

- Implementar en lenguaje Python los algoritmos de ordenamiento ascendente (de menor a mayor) bubble sort y de merge sort que permitan ordenar números.
- A partir de los algoritmos implementados en Python, obtener los polinomios del mejor, el peor y el caso promedio de complejidad de bubble sort y de merge sort.
- A partir del algoritmo implementado en Python, generar la gráfica del tiempo de ejecución para los algoritmos de bubble sort y de merge sort para diferentes instancias de tiempo (listas de 1 al máximo de elementos posibles según la complejidad del algoritmo) para los siguientes casos:
    - Lista ordenada en forma ascendente (de menor a mayor)
    - Lista ordenada en forma descendente (de mayor a menor)
    - Lista con valores aleatorios

3. Resultados

El código fuente y las gráficas obtenidas se encuentran en el archivo "practica1.ipynb".

4. Conclusiones

Esta práctica me permitió comprender de manera clara las diferencias entre algoritmos de ordenamiento con distintas complejidades, bubble sort, aunque es sencillo de implementar, es poco eficiente en conjuntos grandes de números debido a su crecimiento cuadrático $O(n^2)$, y además su rendimiento varía considerablemente dependiendo de si la lista está ordenada ascendentemente, ordenada descendentemente o contiene valores aleatorios.

Por otro lado, merge sort, tiene una complejidad $O(nlog(n))$, mantuvo un desempeño mucho más estable y eficiente en todos los casos, mostrando la importancia de utilizar algoritmos más avanzados para conjuntos de datos grandes.

Además, el análisis del modelo RAM y la obtención de polinomios de operaciones me permitieron relacionar la teoría con los resultados prácticos observados en las gráficas de tiempos de ejecución, esto reforzó la idea de que no basta con que un algoritmo funcione correctamente, sino que también es fundamental considerar su eficiencia y adecuación al problema.