PROBLEMA 1

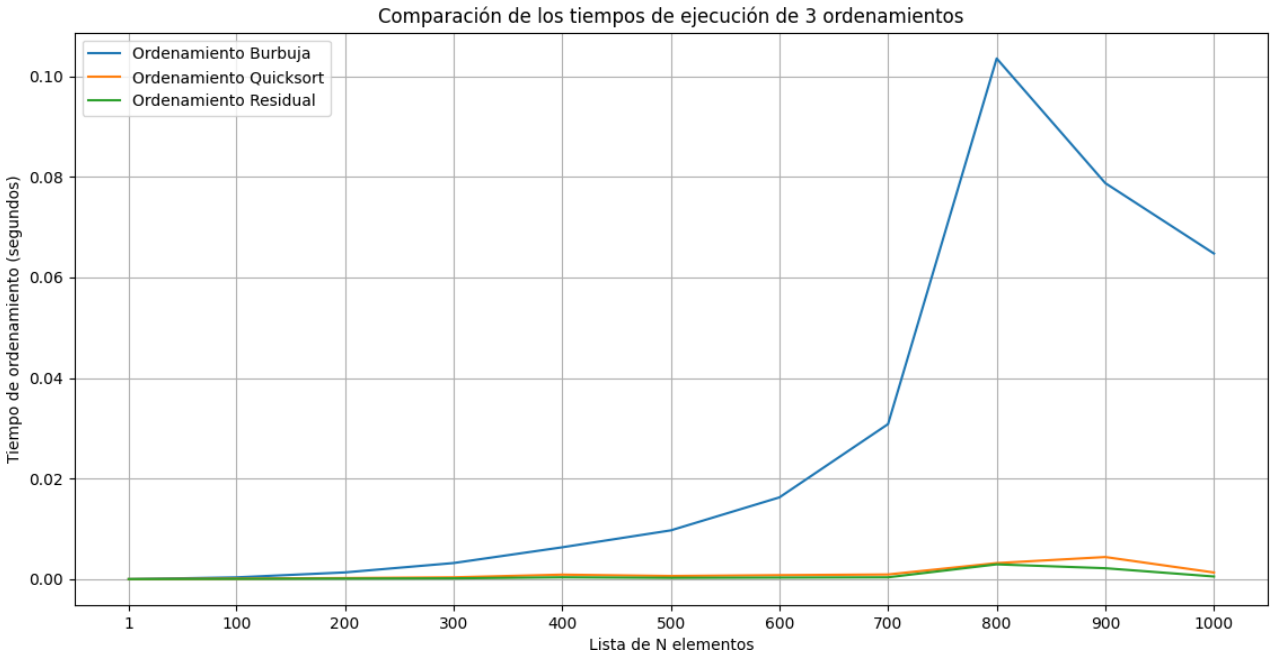

Figura 1: Gráfico de una lista de N elementos vs. Tiempo de ordenamiento.

Resultados por ordenamientos:

- Ordenamiento Burbuja: Compara todos contra todos. Es un método muy lento con un orden de complejidad O(n^2); se considera ineficiente para grandes conjuntos de datos.

- Ordenamiento Quicksort: es un método muy eficiente, más lento que el radix sort, pero mejor que el ordenamiento burbuja. Con una buena elección de pivote, su orden de complejidad promedio es O(n log n), aunque en el peor caso su complejidad puede degradarse a O(n^2).

- Ordenamiento por residuos: es el método más eficiente ya que es más rápido en la práctica que los algoritmos O(n log n) debido a su naturaleza no comparativa y la eficiencia de sus operaciones internas. Se considera que su orden de complejidad es O(kn) debido a que el bucle principal se ejecuta k veces, (una por cada posición del dígito), y dentro de cada iteración del bucle principal, se recorre la lista 'n' veces para distribuir los números en los buckets, y luego se recorren los buckets (que en total contienen 'n' elementos) para reconstruir la lista.

El funcionamiento de los tres algoritmos de ordenamiento fue probado con una lista de 500 números de 5 dígitos formada al azar, disponible en la carpeta apps.

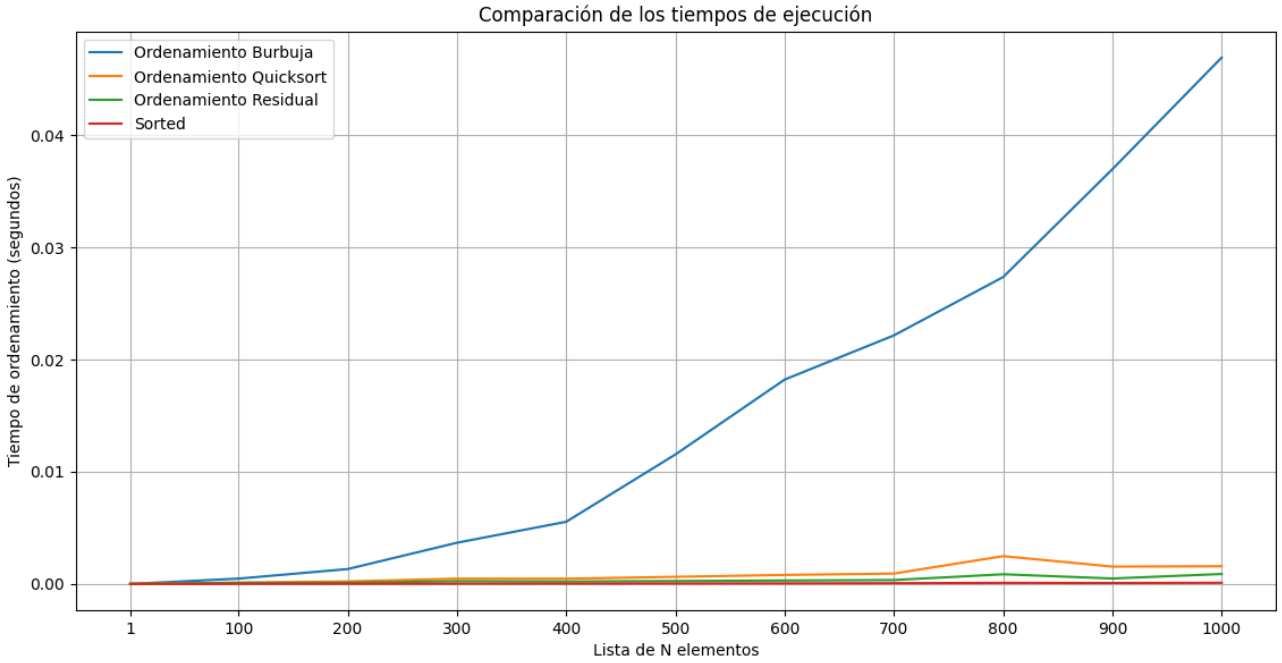

Figura 2: Gráfico de una lista de N elementos vs. Tiempo de ordenamiento.

Cómo funciona sorted() en Python? Características

- La función sorted() en Python no utiliza un único algoritmo de ordenamiento. Desde la versión 2.3, Python utiliza un algoritmo híbrido llamado Timsort: combina las ventajas de Insertion Sort (eficiente para listas pequeñas y parcialmente ordenadas) y Merge Sort (eficiente para listas grandes y con una complejidad en el peor caso garantizada).

- Generalmente tiene un excelente rendimiento, con una complejidad O(n) en el mejor caso, y en el promedio y peor caso de O(n log n).

- Por su sólida complejidad temporal en todos los casos, junto con su estabilidad, sorted() es la función de ordenamiento de propósito general recomendada en Python.

En la gráfica se visualizan claramente las diferencias en la eficiencia de los algoritmos de ordenamiento a medida que aumenta el tamaño de la lista: el Ordenamiento Burbuja es significativamente el más lento, el Quicksort es mucho más eficiente, pero el Ordenamiento Residual (Radix Sort) y la función sorted() de Python (Timsort) demuestran ser los más rápidos para los rangos de tamaño de lista mostrados, con un crecimiento de tiempo de ejecución mucho menor. La pequeña diferencia entre estos últimos puede estar relacionada a que, aunque las operaciones internas de Radix Sort son generalmente rápidas, la implementación puede tener una sobrecarga constante mayor debido a la necesidad de crear y gestionar los buckets en cada pasada, mientras que sorted() es una implementación muy optimizada dentro de Python por lo que sus operaciones de comparación y fusión están altamente afinadas.