Barrera, Muchiut, Oliver
AyED 2C2025

Trabajo Práctico N°1
Informe Actividad N°3

En esta actividad consiste en aplicar los algoritmos de ordenamiento, donde trabajamos con los asignados por la cátedra, que fueron: **ordenamiento burbuja, ordenamiento por residuo (radix sort) y ordenamiento rápido (quick sort)**. A su vez, los compararemos con el método **sorted** predeterminado de python.

El objetivo fue implementar cada uno de ellos, y verificar su funcionamiento a base de los test cumpliendo correctamente con la tarea de ordenar listas de distintos tamaños para finalmente, realizar mediciones de tiempo para analizar sus órdenes de complejidad y compararlas en la gráfica.

Los algoritmos implementados fueron:

- Ordenamiento Burbuja:

Compara los ítems adyacentes e intercambia si no están en orden. Se repite hasta que la lista está ordenada. Su complejidad es **O(n²)**, ya que utiliza dos bucles anidados para recorrer todos los elementos múltiples veces.

- Ordenamiento rápido:

Usa dividir y conquistar. Selecciona un pivote y reorganiza la lista en dos sublistas: una con los elementos menores al pivote y otra con los mayores. Su complejidad es O(n log n), aunque en el peor caso puede ser **O(n²)** si los pivotes no se eligen bien.

- Ordenamiento por residuo:

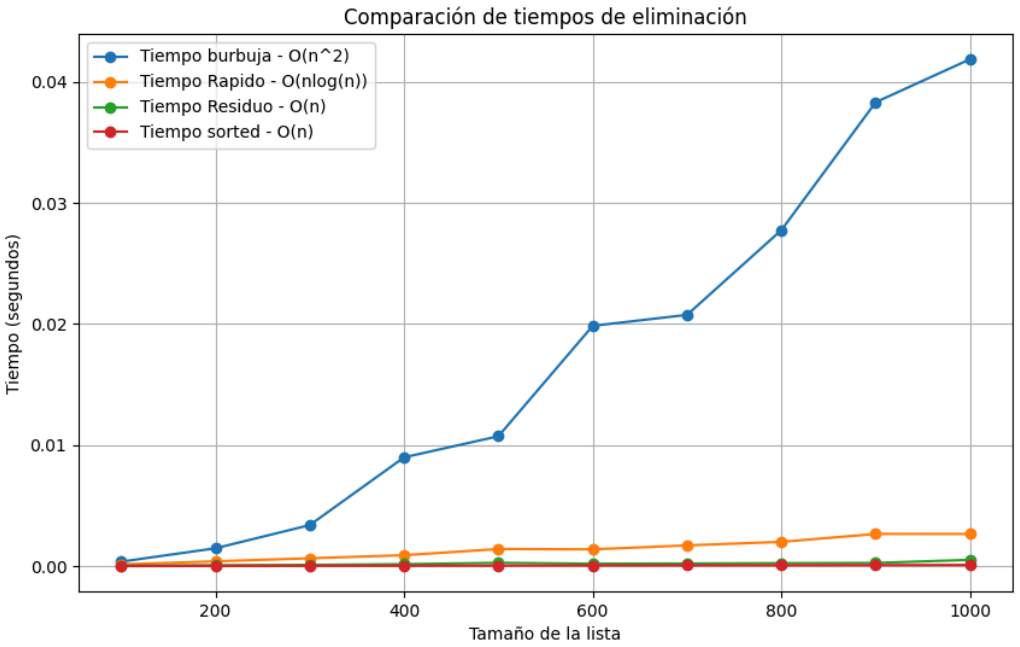
El ordenamiento por residuos utiliza el algoritmo **counting sort**. Comienza por el “dígito menos significativo” y avanza hacia el “dígito más significativo”. Su nivel de complejidad es **O(n)**.

Medición de tiempo en las gráficas:
Para comparar el rendimiento de cada algoritmo, generamos listas con distintos tamaños (100, 200, 300, … , 1000) con valores aleatorios entre 1 y 100. Luego utilizamos la función **time.perf_counter()** para medir los tiempos de ejecución de cada algoritmo y así obtener resultados precisos. Los tiempos de cada ordenamiento lo asignamos a tres listas:

- **tiempo_burbuja** para el algoritmo de burbuja
- **tiempo_rapido** para el algoritmo rápido.
- **tiempo_residuo** para el algoritmo por residuo.

Donde graficamos los resultados con la biblioteca **matplotlib**

Gráfica de tiempo de cada ordenamiento:



La curva **azul** **(burbuja)** crece de forma cuadrática, confirmando que el ordenamiento burbuja tiene un orden de complejidad **O(n²)**. Es el menos eficiente de los tres

La curva **naranja** **(rápido)** muestra un comportamiento cercano a una función **n log n**, creciendo mucho más lentamente que el método de ordenamiento burbuja.

La curva **verde** **(residuo)** se mantiene casi lineal, siendo el algoritmo más rápido en todos los tamaños de lista probados. Siendo el más eficiente de los tres corroborando su orden de complejidad **O(n)** en la práctica.

La curva **roja** **(sorted)** posee la misma linealidad que la curva verde el método residuo teniendo el mismo orden de complejidad de O(n) y un comportamiento similar entre ambos.

Los resultados confirmaron las complejidades teóricas de cada algoritmo, mostrando la gran diferencia de rendimiento que existe entre un algoritmo cuadrático y uno lineal o logarítmico. A su vez, a la hora de comparar el método predeterminado de python “sorted”, notamos que tiene un comportamiento similar al método de ordenamiento radix sort o residuo.