Universidad Nacional de Entre Ríos
Facultad de Ingeniería

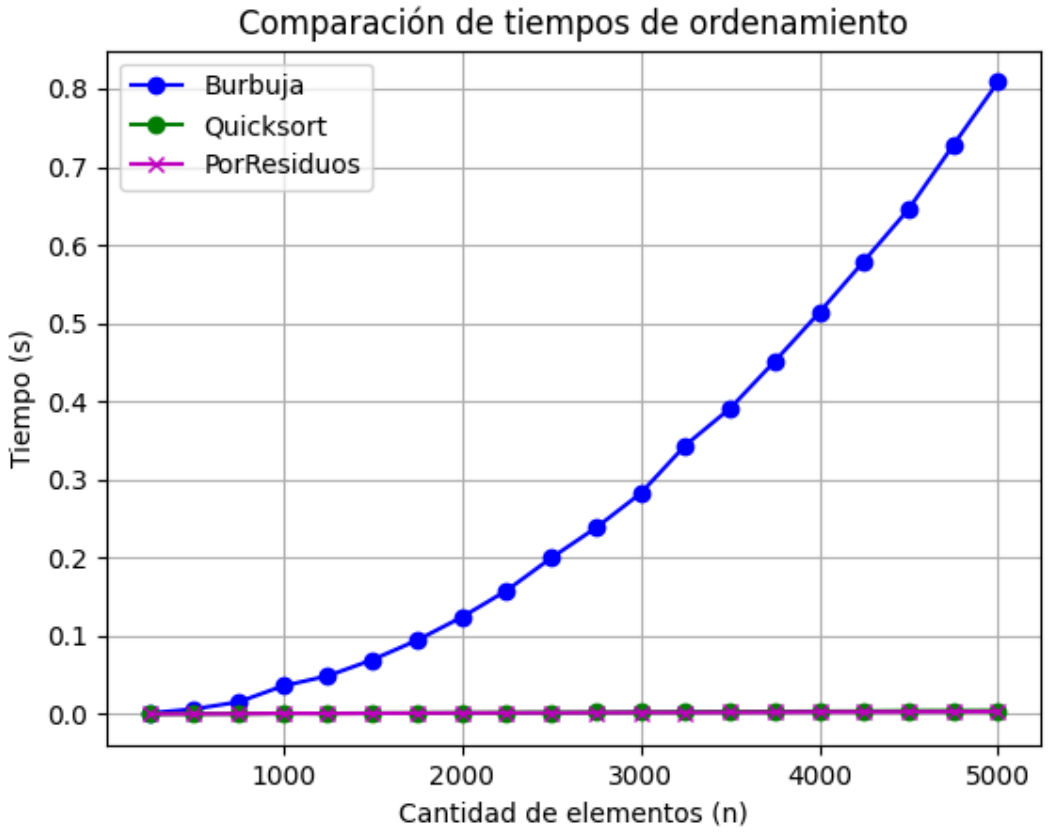
Algoritmos y estructuras de datos

Informe general del Trabajo Práctico N°1
“Aplicaciones de los tipos abstractos de datos”

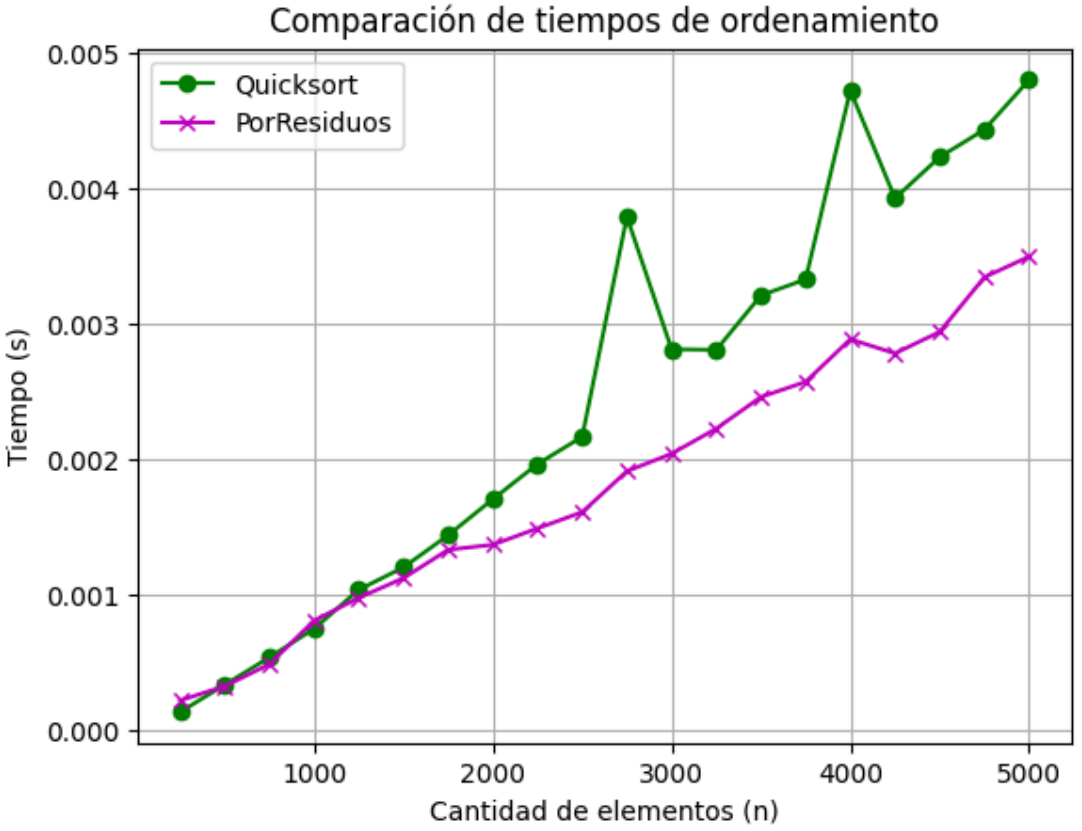
Alumnos:

- Tarabini Melina
- Rodríguez Esteban

Problema 3:



Descripción:
En este gráfico se comparan los tiempos de ejecución de los tres algoritmos de ordenamiento implementados. Se observa claramente que el método Burbuja presenta un crecimiento cuadrático, aumentando el tiempo de forma muy pronunciada a medida que crece el tamaño de la lista. En contraste, Quicksort y el método Por Residuos (Radix Sort) mantienen tiempos prácticamente constantes en la escala del gráfico, evidenciando una mayor eficiencia.



Descripción:
En este gráfico se amplía la comparación entre **Quicksort** y **Por Residuos**, dejando afuera al Burbuja para poder apreciar mejor las diferencias. Se ve que ambos algoritmos crecen con el tamaño de la lista, pero el **Radix Sort** tiende a ser más eficiente y presenta un crecimiento más suave en el tiempo de ejecución.

Ordenamiento Burbuja Complejidad: O(n^2):
● Compara pares de elementos adyacentes y los intercambia si están en orden incorrecto.
● En el peor caso, requiere recorrer el arreglo n veces y en cada pasada compara hasta n elementos.
● En el gráfico se observa que el tiempo de ejecución de Burbuja crece de manera no lineal y rápidamente a medida que n aumenta, lo cual concuerda con una complejidad cuadrática.

Quicksort:
Complejidad promedio: O(n log (n))

Peor caso: si la lista ya está ordenada o semi-ordenada, el algoritmo puede presentar una complejidad O(n^2), debido al desbalance que quedaría en las listas.
● Quicksort utiliza el enfoque de “divide y vencerás”.
● Divide el arreglo en dos subarreglos y luego ordena recursivamente.
● En el gráfico, la línea de Quicksort es prácticamente plana, lo que indica que su tiempo de ejecución es muy bajo en comparación con la de burbuja, y crece mucho más lentamente, como se espera de un algoritmo eficiente.

Por residuos:
Complejidad estimada:O(n·k), donde k es el número de dígitos analizados.
● Aunque no es un algoritmo de ordenamiento típico, puede estar clasificando elementos con base en una operación rápida (como el módulo).
● En el gráfico, su tiempo también se mantiene constante y muy bajo, incluso más bajo que Quicksort, lo cual sugiere que se trata de un algoritmo lineal o casi constante en este rango de valores.

La función sorted() de Python se utiliza para ordenar elementos de cualquier iterable (como listas, tuplas, diccionarios, etc.) y devuelve una nueva lista ordenada, sin modificar el iterable original. Esta divide una lista en sublistas de tamaño óptimo y las ordena con un algoritmo de inserción, para luego fusionarlas, lo que logra un orden de complejidad de O(n log n) donde n es el número de elementos en el iterable que se está ordenando.