Universidad Nacional de Entre Ríos
Facultad de Ingeniería

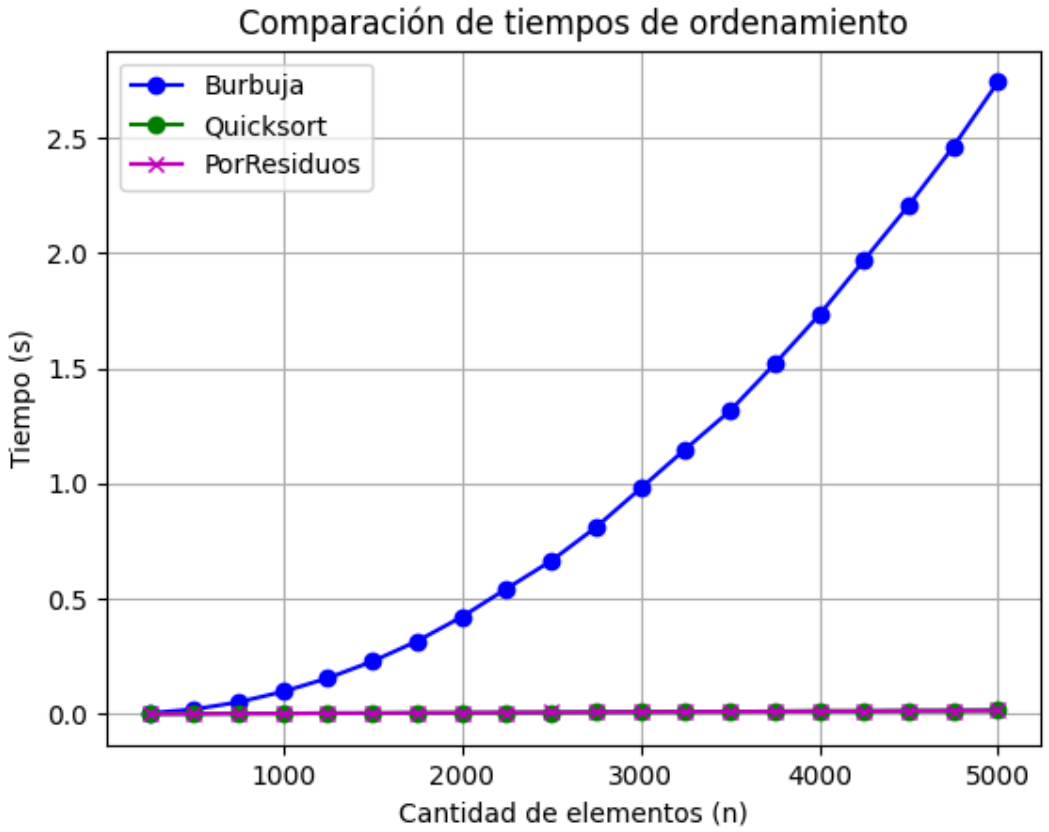
Algoritmos y estructuras de datos

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

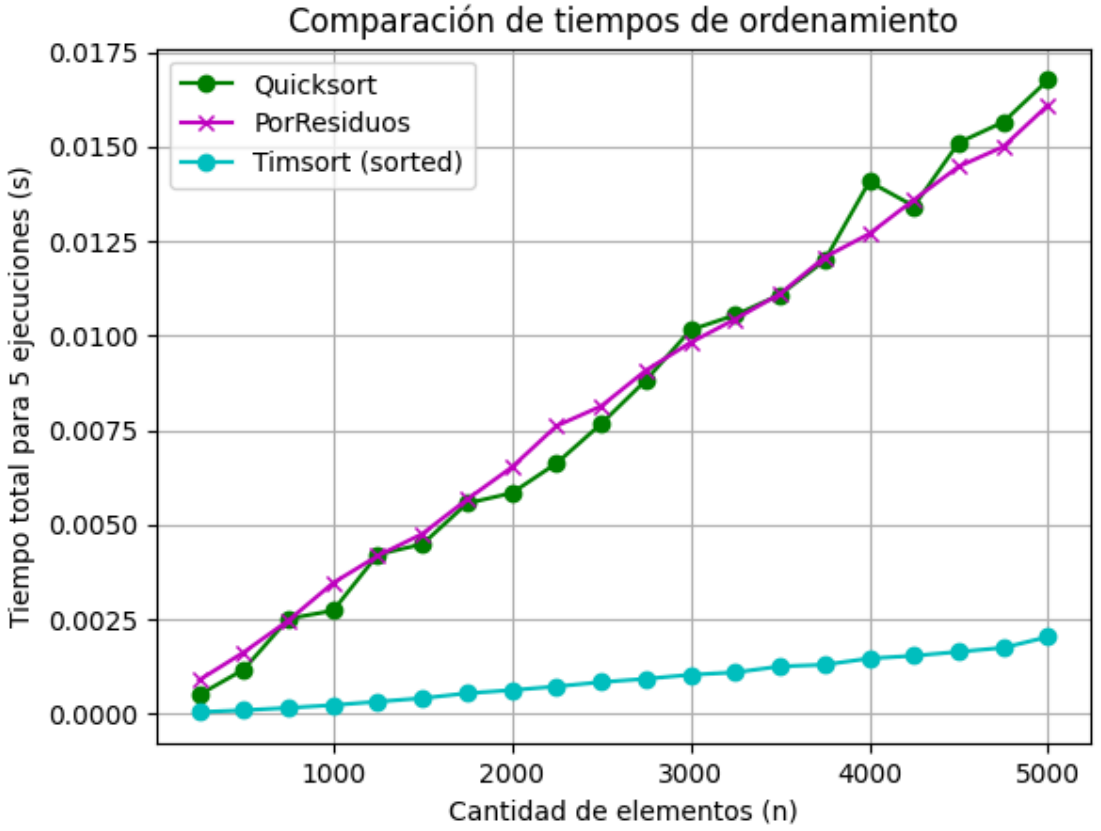
Alumnos:

- 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 comparan los tiempos de ejecución de Quicksort, Por Residuos y Timsort (sorted()), excluyendo al método Burbuja para apreciar mejor las diferencias entre los algoritmos más eficientes.
Se observa que Quicksort y Por Residuos presentan un crecimiento similar a medida que aumenta el tamaño de la lista, mientras que Timsort mantiene tiempos considerablemente menores y un incremento mucho más suave.
Esto evidencia la mayor optimización del algoritmo integrado en Python, que combina estrategias de ordenamiento por mezcla e inserción, logrando una complejidad promedio de O(n log n) y un desempeño superior en la práctica.

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 permite ordenar los elementos de cualquier objeto iterable (como listas, tuplas o conjuntos) y devuelve una nueva lista con los elementos ordenados, sin modificar el original. Internamente utiliza el algoritmo Timsort, una combinación optimizada de Merge Sort e Insertion Sort, que divide la secuencia en pequeñas porciones ya ordenadas (“runs”) y las fusiona eficientemente. Su complejidad promedio y en el mejor caso es O(n log n), mientras que en el peor caso puede llegar a O(n) si la lista ya está parcialmente ordenada.

Fuente utilizada:
Bradley N. Miller, David L. Ranum. Solución de problemas con algoritmos y estructuras de datos usando Python. Luther College. Traducido por Mauricio Orozco-Alzate, Universidad Nacional de Colombia - Sede Manizales. 2018.
Con licencia Creative Commons.
https://drive.google.com/file/d/1EUHiLHlaShb015hOHNhnwXBw6fmhCHsn/view?usp=sharing
(*Material proporcionado por la cátedra de Algoritmos y Estructuras)*