Facultad de Ingeniería
UNER

Informe sobre la implementación y análisis de algoritmos de ordenamiento

Algoritmos y Estructura de Datos

Integrantes:

- Zapata Mariana Gabriela
- Weimer Valentin
- Kerbs Javier

2° Cuatrimestre, 2025

Problema 3: Algoritmos de Ordenamiento.

1. Introducción.

En el tercer ejercicio se aplicaron los conceptos de algoritmos de ordenamiento y de tipos abstractos de datos (TAD). Para ello se implementaron distintos métodos de ordenamiento (burbuja, quicksort y radixsort), se analizaron sus características y se compararon sus rendimientos en términos de tiempo de ejecución.

Para llevarlo a cabo se crearon listas con números aleatorios de 5 dígitos (no menos de 500 números), se midieron los tiempos de ejecución de cada ordenamiento y se generó una gráfica con dichos tiempos obtenidos. Finalmente, se comparan dichos rendimientos de los algoritmos implementados con la función sorted(); y se estudia y comparan los órdenes de complejidad de cada uno.

2. Algoritmos a implementar

Se desarrollan tres algoritmos clásicos de ordenamiento:

Ordenamiento Burbuja (Bubble Sort).
Consiste en recorrer la lista comparando pares de elementos y realizando intercambios cuando están ordenados incorrectamente. Este proceso se repite hasta que toda la lista queda ordenada.
Complejidad temporal:

- Mejor caso: O(n)
- Promedio y peor caso: O(n^2)

Ordenamiento Quicksort.
Elige un elemento “pivote” y divide la lista en dos sublistas: una con los elementos menores y otra con los mayores al pivote. El proceso ordena cada sublista.
Complejidad temporal:

- Promedio: O(n log n)
- Peor caso: O(n^2)

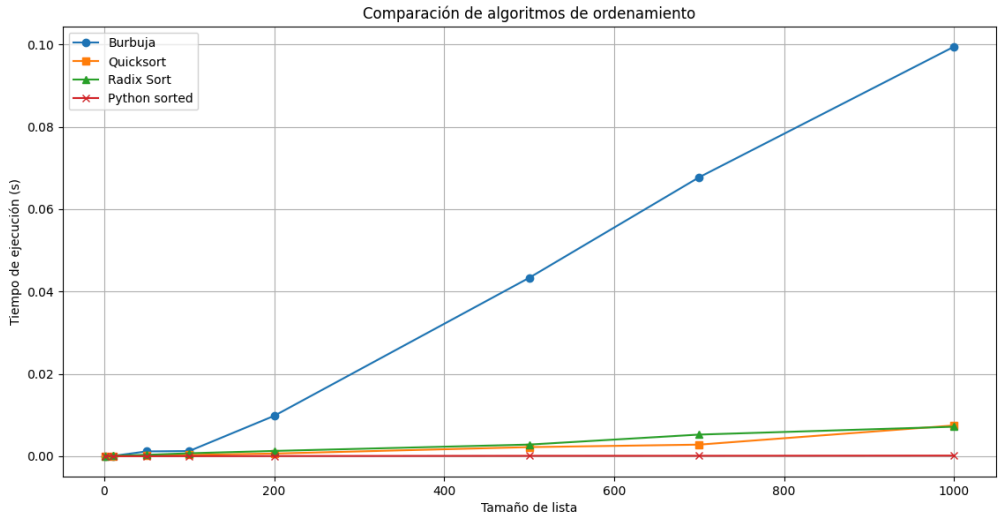
Ordenamiento por residuos (Radix Sort).
Ordena los números por dígitos, comenzando por el que menos cifras significativas tiene hasta el que más presenta. Para ello utiliza un proceso de distribución en cubetas (listas temporales donde se coloca el dígito que se está mirando) según el valor del dígito actual.
Además no realiza comparaciones directas entre números.

Complejidad temporal: $O(n \cdot k)$, donde k es la cantidad de dígitos de los números

3. Experimentos con listas aleatorias.

Se generan listas de números enteros de cinco dígitos, con una longitud mínima de 500 elementos, y sobre estas listas se aplican los tres algoritmos para poder se verificar los tiempos de ejecución de cada uno por medio de time.perf_counter(). Luego se genera un gráfico con los tiempos obtenidos.

4. Resultados.



5. Conclusión.

Para listas grandes, la función sorted() es la opción más eficiente, para números de dígitos pequeños, Radix Sort es muy eficiente, Bubble Sort es útil sólo en casos específicos (no es el mejor para implementar con una gran cantidad de datos) y Quicksort ofrece equilibrio entre eficiencia y complejidad en promedio, pero puede sufrir en el peor caso debido a la elección del pivote.