Informe de Ordenamiento de 3 Métodos

UNIVERSIDAD NACIONAL DE ENTRE RÍOS
Facultad de Ingeniería
Carrera: T.U.P.E.D

Materia: Fundamentos de Algoritmos y Estructuras de Datos

Estudiante: Franco Tomiozzo

Profesor: Dr. Javier E. Diaz Zamboni

Bioing. Jordán F. Insfrán

Esp. Bioing. Francisco Rizzato

Comisión: 1

Fecha: Septiembre 2025

Introducción

El objetivo de este trabajo es implementar y comparar tres algoritmos de ordenamiento en Python, como: Burbuja, Quicksort y Radix Sort. Asimismo, también se va a ver el desempeño de la función sorted() que está incorporada en Python, que implementa el algoritmo llamado Timsort.
Para validar los métodos, se realizaron varias pruebas tanto individualmente como experimentales. Por un lado, las pruebas individuales verificaron la correcta funcionalidad de cada algoritmo.
Las pruebas de rendimiento midieron tiempos de ejecución sobre unas listas de entre 1 y 1000 elementos, con el objetivo de analizar la eficiencia y contraste en el análisis.

Desarrollo

Ordenamiento con Burbuja

El algoritmo de ordenamiento con burbuja funciona para comparar pares de elementos adyacentes e intercambiar cuando están en orden incorrecto. El proceso se repite hasta que no se necesitan más intercambios.

Características principales:

- Muy fácil de implementar.
- Es más eficiente que otros ordenamientos, porque tiene una gran cantidad de comparaciones e intercambios necesarios, por eso es elegido en el mejor de los casos.
- Los números de comparaciones disminuyen en cada pasada, ya que los elementos mayores "suben" al final de la lista.

Ordenamiento Quicksort

El Quicksort es un algoritmo que se dice que “divide y vencerás”. Se selecciona un pivote y divide la lista en dos sublistas: una con menor o igual y la otra con mayor valor. Y luego se ordenan recursivamente ambas partes.

Características principales:

- Es muy eficiente en la práctica.
- Se utiliza recursivamente.
- Solo depende de elegir un buen pivote para evitar una degradación.

Ordenamiento Radix Sort

El Radix Sort no compara elementos entre sí, sino que distribuye los números en función de sus dígitos. Este algoritmo los procesa desde el número menor hasta el más grande, agrupando repetidas veces hasta que la lista queda ordenada.

Características principales:

- Es muy eficiente para enteros y claves de longitud fija.
- Se requiere espacio adicional para las listas auxiliares.

Sorted() en Python — Timsort

La función sorted() que se implementa en Timsort, es un algoritmo híbrido entre Merge Sort e Insertion Sort especialmente optimizando las listas de Python. Está diseñado para reconocer patrones de orden parcial y se aprovechan, logrando gran eficacia en casos reales.

Pruebas de Rendimiento

Se hizo una medición de los tiempos de ejecución para unas listas de tamaño entre 1 y 1000, generadas con enteros aleatorios y se graficaron los resultados para comparar performancia.

Resultados observados:

- **Burbuja**: Es extremadamente lento y su tiempo de ejecución crece cuadráticamente con el tamaño de la lista.
- **Quicksort**: Es rápido incluso en listas grandes, confirmando su orden O(n log n) O(n\log n) O(n log n).
- **Radix Sort**: Es muy eficiente para este tipo de entrada con un crecimiento cercano a lineal.
- **Sorted()**: Este método resultó ser el más consistente y rápido, gracias a la optimización del Timsort.

Conclusiones

- El Ordenamiento Burbuja es el menos eficiente solo es útil para fines pedagógicos.
- El Quicksort es altamente eficiente, ya que su rendimiento teórico en la práctica es efectivo.
- El Radix Sort demostró ser excelente en eficiencia en listas de números enteros, aunque requiere más memoria auxiliar que Quicksort.