INSTITUTO TECNOLÓGICO SUPERIOR DE LA
REGIÓN DE LOS LLANOS

INGENIERÍA MECATRÓNICA

Programación Avanzada

Actividad: U1A4. Reporte de Programa

Evaluación de Métodos de Ordenamiento

Nombre: Juan David Castruita Castañeda

Docente: Osbaldo Aragón Banderas

Fecha: 2026-02-16

TECNM
TECNOLÓGICO NACIONAL DE
MÉXICO

Objetivo de la actividad

El objetivo de la actividad es implementar y evaluar en Python dos métodos de ordenamiento (Burbuja, o Bubble Sort, y Quicksort) usando Visual Studio Code como entorno de desarrollo, comparando su rendimiento con pruebas controladas. La actividad sirve para reforzar el pensamiento algorítmico y el análisis de eficiencia, útiles para optimizar software en aplicaciones de robótica. Además, se busca que se consoliden buenas prácticas de documentación y publicación de evidencias en GitHub como parte del perfil profesional.

Desarrollo en Visual Studio Code

Primero, se configura VS Code para Python, incluyendo la extensión Python de Microsoft y realizando una selección del intérprete. Luego, se creó un proyecto organizado y se desarrolló el código en Python con dos implementaciones:

- **Bubble_sort**
- **Quicksort** (recursivo o iterativo, pero justificando la elección)

Bubble Sort es un algoritmo de ordenamiento simple que funciona comparando elementos adyacentes de una lista y los intercambia si están en el orden incorrecto. Este proceso se repite varias veces hasta que todos los elementos quedan ordenados, lo cual lo hace más lento si se trabaja con una lista de datos muy grande. *Quicksort*, por otro lado, es un algoritmo de ordenamiento eficiente que utiliza la técnica “divide y vencerás”. Esto quiere decir que funciona seleccionando un elemento llamado pivote y dividiendo la lista en dos partes: elementos menores que el pivote y elementos mayores que el pivote. Luego ordena cada parte recursivamente. Este algoritmo es el más empleado en sistemas donde se requiere un control en tiempo real.

En el caso de Quicksort para esta práctica, se eligió una implementación **recursiva** debido a que refleja de manera más directa la naturaleza del algoritmo, facilitando su comprensión conceptual y análisis teórico. En general, la implementación recursiva resulta más clara conceptualmente, representa de mejor manera la definición de Quicksort y es más limpio para el análisis académico.

En el programa de Python, se generaron conjuntos de datos de distintos tamaños mediante la función **random** (ej.: 100, 1 000, 5 000, 10 000) y se consideraron al menos dos escenarios:

- Lista aleatoria
- Lista invertida o casi ordenada.

Pruebas de rendimiento

En esta parte del programa, se midieron tiempos con **timeit**. Se ejecutaron mínimo 5 repeticiones por tamaño y por algoritmo y en base a los datos obtenidos, se organizó una tabla en Microsoft Excel: tamaño de entrada, algoritmo, promedio, y desviación estándar.

El código se desarrolló en el entorno de Visual Studio Code, programando con el lenguaje de Python. Al ejecutarlo, se obtuvieron los siguientes datos en la terminal de VS Code, los cuales se organizaron en una tabla en Excel.

Tamaño	Escenario	Algoritmo	Promedio (s)	Des. Estándar
100	Aleatoria	Bubble Sort	0.002482	0.000094
		Quicksort	0.000768	0.000017
	Invertida	Bubble Sort	0.001038	0.000014
		Quicksort	0.000252	0.000047
1000	Aleatoria	Bubble Sort	0.177535	0.127791
		Quicksort	0.003117	0.000248
	Invertida	Bubble Sort	0.114305	0.002338
		Quicksort	0.002957	0.000314
5000	Aleatoria	Bubble Sort	2.885236	0.502602
		Quicksort	0.017031	0.000975
	Invertida	Bubble Sort	3.801466	0.447591
		Quicksort	0.016396	0.000533
10000	Aleatoria	Bubble Sort	9.918634	0.734364
		Quicksort	0.035367	0.001813

	Invertida	Bubble Sort	17.150565	3.173637
		Quicksort	0.035629	0.00159

Después de obtener los datos del programa y organizarlos en una tabla, se realizó un análisis comparativo de los dos tipos de algoritmos.

¿Cuál algoritmo escala mejor y por qué?

Los resultados que se obtuvieron en VS Code demuestran que el algoritmo **Quicksort** presenta un rendimiento significativamente superior al algoritmo Bubble Sort en todos los tamaños evaluados. Esto es porque a medida que el tamaño de la lista aumenta, el tiempo de ejecución de **Bubble Sort** crece rápidamente, evidenciando un comportamiento cuadrático $O(n^2)$. Esto se debe a que el algoritmo compara repetidamente todos los elementos de la lista, incluso si ya están parcialmente ordenados. Por otro lado, Quicksort presenta un crecimiento mucho menor en el tiempo de ejecución, consistente con su complejidad promedio O(n log n). Esto se debe a su estrategia de dividir la lista en subproblemas más pequeños, lo que reduce el número total de comparaciones necesarias.

La diferencia entre ambos algoritmos se vuelve más evidente en tamaños grandes como 5000 y 10000 elementos, donde Bubble Sort tarda considerablemente más tiempo que Quicksort.

Impacto práctico en la robótica

Cuando se trata de sistemas robóticos y sistemas embebidos, el tiempo de procesamiento es un factor importante, especialmente en aplicaciones que requieren toma de decisiones en tiempo real, como lo es un robot en donde se debe de realizar el procesamiento de sensores, y una navegación autónoma y control de movimiento en base a la información proporcionada por estos mismos.

El uso de Bubble Sort, el cual demostró ser un poco lento, puede introducir retrasos significativos en el sistema, afectando su capacidad de respuesta y eficiencia operativa. Por lo tanto, el uso de Quicksort permite procesar grandes volúmenes de

datos en menor tiempo, mejorando el rendimiento general del sistema y permitiendo una respuesta más rápida ante cambios en el entorno.

Conclusión técnica

Después de la evaluación experimental mediante un programa en Python de los algoritmos Bubble Sort y Quicksort, se puede concluir que Quicksort es significativamente más eficiente, especialmente para conjuntos de datos grandes. Mientras que Bubble Sort presenta una complejidad más alta, $O(n^2)$, Quicksort presenta una complejidad promedio $O(n \log n)$, lo que resulta en un menor tiempo de ejecución. Ambos algoritmos resultan eficientes para la organización y el procesamiento de datos, sin embargo, al conocer las limitaciones respectivas de cada uno, se puede seleccionar el más adecuado según el proyecto a desarrollar.

Los resultados del programa realizado coinciden con el análisis teórico, confirmando que Quicksort es más adecuado para aplicaciones que requieren eficiencia y escalabilidad, un ejemplo siendo la operación de un robot. Por otra parte, Bubble Sort tiene características que lo hacen más eficiente para cuestiones educativas o programas más sencillos, donde no se manejan grandes cantidades de datos. Es por este motivo que la selección adecuada de algoritmos de ordenamiento es fundamental para optimizar el rendimiento en sistemas computacionales y sistemas embebidos.