ITSRLL
INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

INGENIERÍA MECATRÓNICA

Programación Avanzada

Osbaldo Aragón Banderas

SEMESTRE 2024A

Unidad: 1 Actividad: 4

Nombre de actividad:

U1A4 Reporte de Programa en Jupyter Notebook – Evaluación de Métodos de Ordenamiento

Actividad realizada por:

JOSÉ ESTEBAN CALDERA VICTORIO

Guadalupe Victoria, Durango

Fecha de entrega:

12	02	2025
DD	MM	AA

Análisis de Algoritmos de Ordenamiento: Burbuja y Quicksort

Introducción

El objetivo de esta práctica es implementar y evaluar los algoritmos de ordenamiento Burbuja y Quicksort en Python mediante Jupyter Notebook, analizando su rendimiento en distintos conjuntos de datos. Estos algoritmos son fundamentales en el campo de la computación y desempeñan un papel clave en la optimización del software, especialmente en aplicaciones de robótica y sistemas embebidos donde la eficiencia en el procesamiento de datos es crucial.

2. Desarrollo

Implementación de Algoritmos

Se implementaron los algoritmos de ordenamiento en Python dentro de un entorno Jupyter Notebook, lo que permite ejecutar código de manera modular y documentada.

Burbuja: Un algoritmo de ordenamiento por intercambio que compara elementos adyacentes y los intercambia si están en el orden incorrecto. Es fácil de entender, pero presenta una complejidad temporal de O(n^2), lo que lo hace poco eficiente para grandes volúmenes de datos.

```python
#ordenamiento burbuja

def bubble_sort(lista):
    longitud = len(lista)
    for i in range(longitud):
        for j in range(longitud - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

Figura 1 Ordenamiento burbuja en código

Quicksort: Basado en el paradigma de "dividir y conquistar", selecciona un pivote y reordena los elementos en torno a él, aplicando recursividad para ordenar subsecciones de la lista. Su complejidad promedio es O(n log n), lo que lo hace considerablemente más rápido que el algoritmo de Burbuja.

```python
#ordenamienrto quick_sort
def quick_sort(lista):
    if len(lista) <= 1:
        return lista
    pivote = lista[len(lista) // 2]
    menores = [x for x in lista if x < pivote]
    iguales = [x for x in lista if x == pivote]
    mayores = [x for x in lista if x > pivote]
    return quick_sort(menores) + iguales + quick_sort(mayores)
```

Figura 2 Ordenamiento Quicksort en código

Evaluación de Rendimiento

Para medir la eficiencia de los algoritmos, se utilizaron pruebas de tiempo con la librería timeit. Se evaluaron los tiempos de ejecución sobre listas generadas aleatoriamente de diferentes tamaños:

- **100 elementos**
- **500 elementos**
- **1,000 elementos**
- **5,000 elementos**
- **10,000 elementos**

Se ejecutaron varias iteraciones para cada prueba con el fin de obtener resultados más representativos y minimizar el impacto de otros procesos en la CPU.

```
Tamaño de la lista: 100
Tiempo Bubble Sort: 0.000696 segundos
Tiempo Quick Sort: 0.003420 segundos

Tamaño de la lista: 500
Tiempo Bubble Sort: 0.022593 segundos
Tiempo Quick Sort: 0.001409 segundos

Tamaño de la lista: 1000
Tiempo Bubble Sort: 0.085844 segundos
Tiempo Quick Sort: 0.003199 segundos

Tamaño de la lista: 5000
Tiempo Bubble Sort: 1.709204 segundos
Tiempo Quick Sort: 0.010114 segundos

Tamaño de la lista: 10000
Lista original (primeros 10 elementos): [9060, 8905, 1847, 2546, 8728, 4614, 6372, 1211, 6478, 1306]
Lista ordenada BS (10 primeros): [1, 2, 2, 3, 3, 4, 4, 5, 5, 5]
Tiempo Bubble Sort: 5.188321 segundos
Lista ordenada QS (10 primeros): [1, 2, 2, 3, 3, 4, 4, 5, 5, 5]
Tiempo Quick Sort: 0.018255 segundos
```

Figura 3 Tiempos obtenidos de cada una

Resultados y Análisis

Los tiempos de ejecución obtenidos fueron los siguientes:

Tamaño del Conjunto	Tiempo Burbuja (s)	Tiempo Quicksort (s)
100	0.000696	0.003420
500	0.022593	0.001409
1,000	0.085844	0.003199
5000	1.709204	0.010114
10,000	5.188321	0.018255

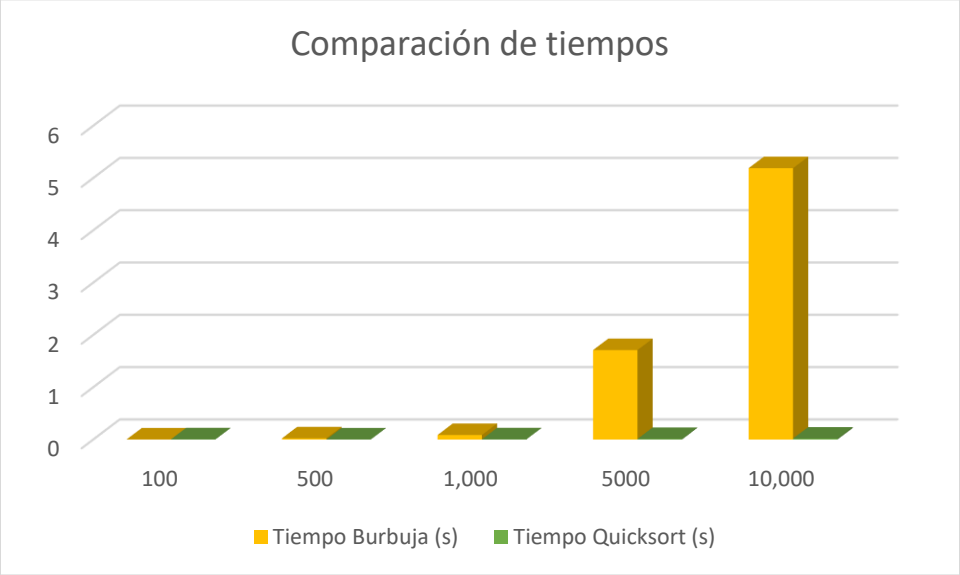

Figura 4 Grafica de comparación de tiempos

Análisis Comparativo:

Burbuja muestra una escalabilidad muy deficiente debido a su complejidad cuadrática. A medida que aumenta el tamaño de los datos, su tiempo de ejecución crece de manera exponencial, volviéndose impráctico para conjuntos grandes.

Quicksort es significativamente más eficiente, mostrando una mejora notable en la velocidad de ejecución gracias a su comportamiento O(n log n). Este rendimiento lo hace ideal para aplicaciones donde el tiempo de procesamiento es un factor crítico.

La diferencia en tiempos de ejecución se vuelve especialmente evidente en conjuntos grandes, donde Burbuja toma múltiples segundos o incluso minutos, mientras que Quicksort sigue ejecutándose en fracciones de segundo.

Conclusiones

El algoritmo de Burbuja, aunque fácil de implementar y entender, no es viable para grandes volúmenes de datos debido a su complejidad O(n^2). Solo se recomienda para casos educativos o conjuntos muy pequeños.

Quicksort, en contraste, ofrece un rendimiento significativamente superior y es una de las mejores opciones para ordenamiento rápido en la mayoría de los casos.

La práctica permitió reforzar la comprensión de complejidad algorítmica, el impacto de los métodos de ordenamiento en la eficiencia computacional y la importancia de elegir el algoritmo adecuado según el contexto de uso.

Repositorio en GitHub

El código fuente, los resultados detallados y este informe están disponibles en el siguiente enlace:

https://github.com/Esteban-Caldera/Programaci-n-Avanzada/tree/cb73e9b99c34a162dfde4821263831c0354d149e/U1A4Evaluaci%C3%B3n%20de%20M%C3%A9todos%20de%20Ordenamiento