Introducción

El ordenamiento de datos es una operación fundamental en el ámbito de la informática, utilizada en una gran variedad de aplicaciones, desde bases de datos hasta algoritmos de búsqueda. En esta evaluación, se analizarán dos de los algoritmos de ordenamiento más conocidos: el Método Burbuja y QuickSort.

El objetivo de esta evaluación es comparar su eficiencia midiendo los tiempos de ejecución al ordenar listas de diferentes tamaños. Se busca entender cómo la elección del algoritmo adecuado puede impactar el rendimiento, especialmente en entornos donde la eficiencia es crucial, como sistemas de robótica e inteligencia artificial.

Algoritmos Evaluados

1. Método Burbuja

El algoritmo de ordenamiento burbuja es sencillo, pero ineficiente, especialmente cuando se trabaja con grandes volúmenes de datos. Su peor caso requiere numerosas comparaciones y reasignaciones, lo que genera un tiempo de ejecución cuadrático **$O(n^2)$**, donde *n* es el número de elementos a ordenar.

2. QuickSort

QuickSort es un algoritmo de ordenamiento mucho más eficiente, ya que divide el conjunto de datos en partes más pequeñas y las ordena de manera recursiva. Su complejidad promedio es **$O(n \log n)$**, lo que lo hace mucho más adecuado para grandes volúmenes de datos.

Implementación en Python

A continuación, se presenta la implementación en Python de ambos algoritmos y la medición de tiempos de ejecución:

```python
import time
import random

def metodo_burbuja(lista):
    """Implementación del método de ordenamiento Burbuja."""
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista

def quick_sort(lista):
    """Implementación del método de ordenamiento QuickSort."""
    if len(lista) < 2:
        return lista
    else:
        pivote = lista[-1]
        menores = [x for x in lista[:-1] if x < pivote]
        mayores = [x for x in lista[:-1] if x >= pivote]
        return quick_sort(menores) + [pivote] + quick_sort(mayor

def medir_tiempo(algoritmo, lista):
    """Mide el tiempo de ejecución de un algoritmo de ordenamier
    inicio = time.time()
    resultado = algoritmo(lista.copy())  # Copia la lista para e
    fin = time.time()
    return fin - inicio

# Generar listas de prueba de diferentes tamaños
tamaños = [1000, 5000, 10000]
resultados = {}

for n in tamaños:
    lista_prueba = [random.randint(0, 10000) for _ in range(n)]
    tiempo_burbuja = medir_tiempo(metodo_burbuja, lista_prueba)
    tiempo_quick = medir_tiempo(quick_sort, lista_prueba)

    resultados[n] = {
        "Burbuja": tiempo_burbuja,
        "QuickSort": tiempo_quick
    }
```

```python
# Imprimir resultados
print("Comparación de tiempos de ejecución:")
for tamaño, tiempos in resultados.items():
    print(f"\nTamaño de la lista: {tamaño}")
    print(f"Método Burbuja: {tiempos['Burbuja']:.6f} segundos")
    print(f"QuickSort: {tiempos['QuickSort']:.6f} segundos")
```

Resultados y Análisis

Los tiempos de ejecución obtenidos fueron los siguientes:

Tamaño de la Lista	Método Burbuja (segundos)	Quicksort (segundos)
1000 elementos	0.2478	0.0029
5000 elementos	3.8806	0.0189
10,000 elementos	13.5996	0.0459

Análisis

- **Burbuja es ineficiente:** A medida que aumenta el tamaño de la lista, el tiempo de ejecución crece de forma cuadrática.
- **QuickSort es mucho más rápido,** incluso con 10,000 elementos.
- **QuickSort es la mejor opción para grandes volúmenes de datos.**

Conclusiones

QuickSort es significativamente más eficiente que el método Burbuja, especialmente cuando se trabaja con listas grandes, mientras que el método Burbuja solo resulta útil para listas pequeñas o con fines educativos. Este análisis destaca la importancia de elegir el algoritmo adecuado según el tamaño y la complejidad del problema, asegurando un rendimiento óptimo. Además, el uso de herramientas como Jupyter Notebook y GitHub se vuelve fundamental para documentar, organizar y compartir código de manera eficiente.