INSTITUTO TECNOLÓGICO SUPERIOR DE LA
REGIÓN DE LOS LLANOS

INGENIERÍA MECATRÓNICA

Programación Avanzada

Actividad: U1A4. REPORTE DE PROGRAMA

Evaluación de Métodos de Ordenamiento

Nombre: Jacqueline Estupiñan

Docente: Osbaldo Aragón Banderas

Fecha: 2026-02-15

TECNM
TECNOLÓGICO NACIONAL DE
MÉXICO

Índice

Introducción

El ordenamiento de datos es uno de los problemas fundamentales en la ciencia de la computación, ya que constituye la base para múltiples procesos como la búsqueda, el análisis de información y la optimización de sistemas. Existen diversos algoritmos de ordenamiento, cada uno con características particulares en cuanto a eficiencia, complejidad y rendimiento según el tamaño y la naturaleza de los datos.

Entre los algoritmos más conocidos se encuentran Bubble Sort, un método sencillo basado en intercambios sucesivos, y QuickSort, un algoritmo más avanzado que utiliza la estrategia de divide y vencerás para mejorar el desempeño en grandes volúmenes de información. Aunque ambos cumplen la misma función, su comportamiento computacional varía considerablemente conforme aumenta el número de elementos a ordenar.

En esta práctica se implementan y comparan ambos algoritmos en Python, evaluando su rendimiento bajo diferentes tamaños de entrada y escenarios de datos. A través de mediciones controladas del tiempo de ejecución y un análisis estadístico de los resultados, se busca comprender su eficiencia práctica y su relevancia en aplicaciones reales, especialmente en áreas como la robótica y el procesamiento de datos en tiempo real, donde la rapidez y la optimización de recursos son factores críticos.

Desarrollo del Programa

El programa fue desarrollado en Python utilizando Visual Studio Code como entorno de desarrollo.

Se implementaron los siguientes algoritmos:

- Bubble Sort (ordenamiento por intercambio)
- QuickSort (implementado de manera recursiva)

Se generaron listas con los siguientes tamaños:

- 100 elementos
- 1000 elementos
- 5000 elementos
- 10000 elementos

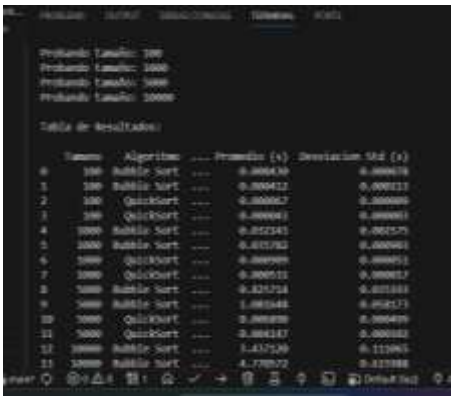
Para cada tamaño se evaluaron dos escenarios:

- Lista aleatoria
- Lista invertida

Se realizaron 5 repeticiones por cada combinación y se calcularon:

- Tiempo promedio
- Desviación estándar

Se utilizó la librería timeit para medir los tiempos de ejecución y pandas para estructurar los resultados en una tabla.



Codigo utilizado:

```
import random

import timeit

import statistics

import csv

import pandas as pd


# ==============================

# Bubble Sort

# ==============================

def bubble_sort(lista):

    arr = lista.copy()

    n = len(arr)


    for i in range(n):

        for j in range(0, n - i - 1):

            if arr[j] > arr[j + 1]:

                arr[j], arr[j + 1] = arr[j + 1], arr[j]


    return arr
```

```
# ==============================
# QuickSort (recursivo)
# ==============================
def quicksort(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 quicksort(menores) + iguales + quicksort(mayores)

# ==============================
# Generadores de datos
# ==============================
def generar_lista_aleatoria(n):
    return [random.randint(0, 10000) for _ in range(n)]
```

```
def generar_lista_invertida(n):
    return list(range(n, 0, -1))


# ==============================
# Medición de tiempo
# ==============================
def medir_tiempo(funcion, lista, repeticiones=5):

    tiempos = []

    for _ in range(repeticiones):

        tiempo = timeit.timeit(
            lambda: funcion(lista),
            number=1
        )

        tiempos.append(tiempo)

    promedio = statistics.mean(tiempos)
    desviacion = statistics.stdev(tiempos)
```

```
    return tiempos, promedio, desviacion


# ==============================
# Programa principal
# ==============================
def main():

    tamanos = [100, 1000, 5000, 10000]
    repeticiones = 5
    resultados = []

    print("Evaluando algoritmos...\n")

    for n in tamanos:

        print(f"Probando tamaño: {n}")

        lista_aleatoria = generar_lista_aleatoria(n)
        lista_invertida = generar_lista_invertida(n)
```

```
for algoritmo, funcion in [
    ("Bubble Sort", bubble_sort),
    ("QuickSort", quicksort)
]:

    for tipo, lista in [
        ("Aleatoria", lista_aleatoria),
        ("Invertida", lista_invertida)
    ]:

        tiempos, prom, desv = medir_tiempo(
            funcion, lista, repeticiones
        )

        resultados.append([
            n,
            algoritmo,
            tipo,
            repeticiones,
            prom,
            desv
```

```
            ])


# ==========================
# Crear DataFrame con pandas
# ==========================


columnas = [
    "Tamano",
    "Algoritmo",
    "Tipo Lista",
    "Repeticiones",
    "Promedio (s)",
    "Desviacion Std (s)"
]


df = pd.DataFrame(resultados, columns=columnas)


print("\nTabla de Resultados:\n")
print(df)


# ==========================
```

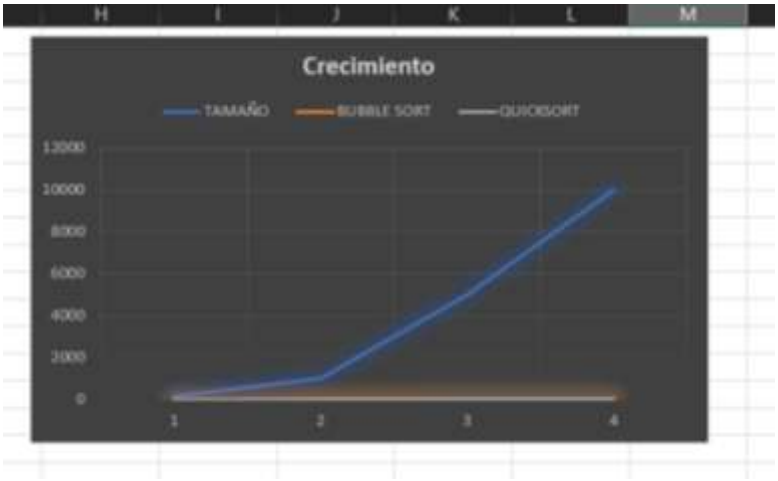
```
    # Guardar CSV
    # ==========================

    df.to_csv("resultados_ordenamiento.csv", index=False)

    # ==========================
    # Guardar Excel (más profesional)
    # ==========================

    df.to_excel("resultados_ordenamiento.xlsx", index=False)

    print("\nArchivos generados:")
    print("- resultados_ordenamiento.csv")
    print("- resultados_ordenamiento.xlsx")

# Ejecutar programa
if __name__ == "__main__":
    main()
```

Resultados

Tabla de resultados

Tamano	Algoritmo	Tipo Lista	Repeticion	Promedio	Desviacion	Std
100	Bubble Sor	Aleatoria	5	0.000314	3.9E-05	
100	Bubble Sor	Invertida	5	0.000417	1.7E-05	
100	QuickSort	Aleatoria	5	9.44E-05	1.48E-05	
100	QuickSort	Invertida	5	8.06E-05	3.85E-05	
1000	Bubble Sor	Aleatoria	5	0.032919	0.006764	
1000	Bubble Sor	Invertida	5	0.043105	0.01165	
1000	QuickSort	Aleatoria	5	0.000885	3.55E-05	
1000	QuickSort	Invertida	5	0.000625	5.01E-05	
5000	Bubble Sor	Aleatoria	5	0.856887	0.026056	
5000	Bubble Sor	Invertida	5	1.057743	0.040775	
5000	QuickSort	Aleatoria	5	0.005522	0.000529	
5000	QuickSort	Invertida	5	0.003565	0.000324	
10000	Bubble Sor	Aleatoria	5	3.614489	0.164709	
10000	Bubble Sor	Invertida	5	4.451553	0.143182	
10000	QuickSort	Aleatoria	5	0.015288	0.001729	
10000	QuickSort	Invertida	5	0.009498	0.001652	

Gráfica comparativa



Análisis Comparativo

Los resultados obtenidos muestran una diferencia significativa en el rendimiento de ambos algoritmos conforme aumenta el tamaño de los datos.

Bubble Sort presenta un crecimiento acelerado en el tiempo de ejecución. Por ejemplo:

- Pasa de 0.0004 segundos con 100 elementos
- A 3.43 segundos con 10000 elementos

Esto confirma su complejidad teórica de:

$O(n^2)$

Por otro lado, QuickSort mantiene tiempos considerablemente menores:

- 0.00006 segundos con 100 elementos
- 0.016 segundos con 10000 elementos

Esto corresponde a su complejidad promedio:

$O(n \log n)$

La diferencia se vuelve más evidente conforme aumenta el tamaño del conjunto de datos.

Impacto en Aplicaciones de Robótica

En sistemas robóticos y procesamiento de datos en tiempo real:

- Los sensores generan grandes volúmenes de información.
- Se requiere tomar decisiones rápidamente.
- El tiempo de cómputo afecta directamente el rendimiento del sistema.

Un algoritmo como Bubble Sort podría generar retrasos importantes en sistemas con grandes volúmenes de datos, mientras que QuickSort ofrece una solución mucho más eficiente y escalable.

Por lo tanto, QuickSort es más adecuado para aplicaciones reales que requieren rapidez y eficiencia.

Conclusión

A partir de las pruebas realizadas, se comprobó que QuickSort ofrece un rendimiento considerablemente mejor que Bubble Sort, especialmente cuando el tamaño de la lista aumenta. Mientras Bubble Sort incrementa su tiempo de ejecución de forma acelerada, QuickSort mantiene tiempos mucho más bajos y estables.

Los resultados coinciden con la teoría de complejidad:

Bubble Sort → $O(n^2)$

QuickSort → $O(n \log n)$

Esto demuestra que la eficiencia del algoritmo influye directamente en el desempeño del programa, sobre todo al trabajar con grandes volúmenes de datos.

En conclusión, la práctica permitió validar experimentalmente el análisis teórico y reforzar la importancia de seleccionar algoritmos adecuados según el contexto, particularmente en aplicaciones como la robótica y el procesamiento en tiempo real, donde el tiempo de respuesta es un factor crítico.