INSTITUTO TECNOLÓGICO SUPERIOR DE LA
REGIÓN DE LOS LLANOS

INGENIERÍA MECATRÓNICA

PROGRAMACION AVANZADA

Actividad: U1A4 REPORTE DE PROGRAMA

Microcomputadora, microprocesador y microcontrolador

Nombre: Jean Paul Acosta Navarro

Docente: Osbaldo Aragón Banderas

Fecha: 2026-02-16

TECNM
TECNOLÓGICO NACIONAL DE MÉXICO

Objetivo

Implementar y evaluar en **Python** dos métodos de ordenamiento (**Burbuja** y **Quicksort**) usando **Visual Studio Code** como entorno de desarrollo, comparando su rendimiento con pruebas controladas. La actividad refuerza el pensamiento algorítmico y el análisis de eficiencia, útiles para optimizar software en aplicaciones de robótica. Además, se busca que el estudiante consolide buenas prácticas de documentación y publicación de evidencias en **GitHub** como parte de su perfil profesional.

Instrucciones

1) Desarrollo en Visual Studio Code

1. Configura VS Code para Python (extensión **Python** de Microsoft y selección del intérprete).
2. Crea un proyecto organizado y desarrolla el código en Python con **dos implementaciones**:
    - bubble_sort(lista)
    - quicksort(lista) (recursivo o iterativo, pero justificando tu elección)
3. Genera conjuntos de datos de distintos tamaños (ej.: 100, 1 000, 5 000, 10 000) y considera al menos **dos escenarios**:
    - Lista aleatoria
    - Lista invertida o casi ordenada

2) Pruebas de rendimiento (obligatorio)

1. Mide tiempos con timeit (recomendado).
2. Ejecuta **mínimo 5 repeticiones** por tamaño y por algoritmo.
3. Reporta en tabla: tamaño de entrada, algoritmo, repeticiones, **promedio** (y desviación estándar si puedes).

4. Realiza un análisis comparativo:
    - ¿Cuál algoritmo escala mejor y por qué?
    - Relación con la complejidad esperada (**O(n²)** vs **O(n log n)**)
    - Impacto práctico en robótica (tiempo de cómputo, toma de decisiones, procesamiento de datos/sensores).

Código de Python en VS Code.

```python
import random
import statistics as stats
import timeit
import csv
from typing import List, Dict, Tuple

# ----------------------------
# 1) Implementaciones de ordenamiento
# ----------------------------
def bubble_sort(arr: List[int]) -> List[int]:
    """Ordena una lista usando Bubble Sort (O(n^2)). Devuelve una NUEVA lista."""
    a = arr.copy()
    n = len(a)

    for i in range(n):
        swapped = False
        for j in range(0, n - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                swapped = True
        if not swapped:
            break

    return a


def quicksort(arr: List[int]) -> List[int]:
    """
    Quicksort recursivo (promedio O(n log n)).
    Devuelve una NUEVA lista.
    """
    if len(arr) <= 1:
        return arr.copy()

    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    mid = [x for x in arr if x == pivot]
```

Ilustración 1 codigo python parte 1

```python
    right = [x for x in arr if x > pivot]

    return quicksort(left) + mid + quicksort(right)


# ----------------------------
# 2) Generación de datos (tamaños + escenarios)
# ----------------------------
def generate_data(n: int, scenario: str, seed: int = 42) -> List[int]:
    """
    scenario:
      - "random": lista aleatoria
      - "reversed": lista invertida
    """
    rnd = random.Random(seed + n)  # semilla estable por tamaño

    if scenario == "random":
        a = list(range(n))
        rnd.shuffle(a)
        return a

    if scenario == "reversed":
        return list(range(n, 0, -1))

    raise ValueError("Escenario no válido. Usa: random, reversed")


# ----------------------------
# 3) Medición con timeit + repeticiones
# ----------------------------
def measure_sort_time(sort_fn, data: List[int], repeats: int = 5) -> Tuple[float, float, List[float]]:
    """
    Mide tiempos (segundos) usando timeit con múltiples repeticiones.
    Devuelve: (promedio, desviación_estándar, lista_de_tiempos)
```

Ilustración 2 codigo python parte 2

```python
    """
    timer = timeit.Timer(lambda: sort_fn(data))
    times = timer.repeat(repeat=repeats, number=1)

    mean_t = stats.mean(times)
    stdev_t = stats.stdev(times) if repeats > 1 else 0.0

    return mean_t, stdev_t, times


# ----------------------------
# 4) Ejecución del experimento
# ----------------------------
def run_experiment(
    sizes=(100, 1000, 5000, 10000),
    scenarios=("random", "reversed"),
    repeats=5
) -> List[Dict]:
    algorithms = [
        ("Burbuja", bubble_sort),
        ("Quicksort", quicksort),
    ]

    results = []

    for scenario in scenarios:
        for n in sizes:
            data = generate_data(n, scenario=scenario)

            for alg_name, alg_fn in algorithms:
                mean_t, stdev_t, _ = measure_sort_time(alg_fn, data, repeats=repeats)

                results.append({
                    "escenario": scenario,
```

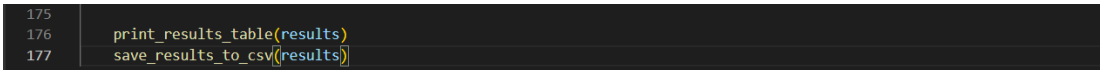
Ilustración 3 codigo de python parte 3

```
                    "n": n,
                    "algoritmo": alg_name,
                    "repeticiones": repeats,
                    "promedio_s": mean_t,
                    "desv_std_s": stdev_t
                })

    return results


# ----------------------------
# 5) Imprimir tabla en consola
# ----------------------------
def print_results_table(results: List[Dict]) -> None:
    header = f"{'Escenario':<12} {'n':>8} {'Algoritmo':<10} {'Rep':>4} {'Promedio (s)':>14} {'DesvStd (s)':>12}"
    print(header)
    print("-" * len(header))

    for r in results:
        print(
            f"{r['escenario']:<12} "
            f"{r['n']:>8} "
            f"{r['algoritmo']:<10} "
            f"{r['repeticiones']:>4} "
            f"{r['promedio_s']:>14.6f} "
            f"{r['desv_std_s']:>12.6f}"
        )


# ----------------------------
# 6) Guardar resultados en CSV
# ----------------------------
def save_results_to_csv(results: List[Dict], filename: str = "resultados_ordenamiento.csv") -> None:
    with open(filename, mode="w", newline="", encoding="utf-8") as file:
```

Ilustración 4 codigo de python parte 4

```
        writer = csv.writer(file)

        # Encabezados
        writer.writerow([
            "escenario",
            "n",
            "algoritmo",
            "repeticiones",
            "promedio_s",
            "desv_std_s"
        ])

        # Filas
        for r in results:
            writer.writerow([
                r["escenario"],
                r["n"],
                r["algoritmo"],
                r["repeticiones"],
                r["promedio_s"],
                r["desv_std_s"]
            ])

    print(f"\n✅ CSV guardado como: {filename}")


# ----------------------------
# 7) MAIN
# ----------------------------
if __name__ == "__main__":
    results = run_experiment(
        sizes=(100, 1000, 5000, 10000),
        scenarios=("random", "reversed"),
        repeats=5
    )
```

Ilustración 5 codigo de python parte 5



```

    print_results_table(results)
    save_results_to_csv(results)
```

Ilustración 6 codigo de python parte 6

Resultados generados en VS Studio Code y en un archivo .csv

resultados_ordenamiento.csv > data

```
escenario,n,algoritmo,repeticiones,promedio_s,desv_std_s
random,100,Burbuja,5,0.00042382000829093157,0.00014523527799309994
random,100,Quicksort,5,9.706000564619899e-05,9.010166842196505e-06
random,1000,Burbuja,5,0.042292240000097084,0.0020284092132869247
random,1000,Quicksort,5,0.0015300200029741973,0.000187005627160312
random,5000,Burbuja,5,1.1737225399992894,0.123060607885174
random,5000,Quicksort,5,0.008872099994914607,0.000515924432563203
random,10000,Burbuja,5,5.260979759990005,0.6305999556838033
random,10000,Quicksort,5,0.023723700002301486,0.0020248642898129023
reversed,100,Burbuja,5,0.0007254399941302836,0.00010809145144911226
reversed,100,Quicksort,5,8.220000891014933e-05,1.9184892878076 63e-05
reversed,1000,Burbuja,5,0.06318869999959134,0.0049162530675984605
reversed,1000,Quicksort,5,0.0010641400062013418,2.9891437171246946e-05
reversed,5000,Burbuja,5,1.6413929399976042,0.10070128354199918
reversed,5000,Quicksort,5,0.00599037999054417,0.00019150686013595387
reversed,10000,Burbuja,5,6.643781740003033,0.11395252450628048
reversed,10000,Quicksort,5,0.013607999996747822,0.0006160872196192832
```

Ilustración 7 resultados en VS Studio Code.

	escenario	n	algoritmo	repeticiones	promedio_s	desv_std_s
2	random	100	Burbuja	5	0.00042382000829093157	0.00014523527799309994
3	random	100	Quicksort	5	9.706000564619899e-05	9.010166842196505e-06
4	random	1000	Burbuja	5	0.042292240000097084	0.0020284092132869247
5	random	1000	Quicksort	5	0.0015300200029741973	0.000187005627160312
6	random	5000	Burbuja	5	1.1737225399992894	0.123060607885174
7	random	5000	Quicksort	5	0.008872099994914607	0.000515924432563203
8	random	10000	Burbuja	5	5.260979759990005	0.6305999556838033
9	random	10000	Quicksort	5	0.023723700002301486	0.0020248642898129023
10	reversed	100	Burbuja	5	0.0007254399941302836	0.00010809145144911226
11	reversed	100	Quicksort	5	8.220000891014933e-05	1.918489287807663e-05
12	reversed	1000	Burbuja	5	0.06318869999959134	0.0049162530675984605
13	reversed	1000	Quicksort	5	0.0010641400062013418	2.9891437171246946e-05
14	reversed	5000	Burbuja	5	1.6413929399976042	0.10070128354199918
15	reversed	5000	Quicksort	5	0.00599037999054417	0.00019150686013595387
16	reversed	10000	Burbuja	5	6.643781740003033	0.11395252450628048
17	reversed	10000	Quicksort	5	0.013607999996747822	0.0006160872196192832

Ilustración 8 resultados obtenidos en el archivo .csv (github).

Relación con la complejidad computacional

Los resultados experimentales obtenidos mediante timeit corroboran el comportamiento asintótico esperado de ambos algoritmos conforme a su análisis de complejidad temporal.

Bubble Sort – O(n²)

Bubble Sort presenta una complejidad temporal cuadrática:

$$T(n) = O(n^2)$$

Esto se debe a su estructura de doble ciclo anidado, donde cada elemento se compara repetidamente con el resto de la lista. En el peor y promedio de los casos, el número de comparaciones y swaps crece proporcionalmente a n^2

Experimentalmente, al aumentar el tamaño de entrada de 1,000 a 10,000 elementos (factor 10), el tiempo de ejecución aumenta aproximadamente por un factor cercano a 100, lo cual es consistente con el crecimiento cuadrático.

Este comportamiento evidencia una escalabilidad limitada para tamaños de entrada grandes.

Quicksort – O(n log n)

Quicksort presenta una complejidad promedio:

$$T(n) = O(n \log n)$$

Su eficiencia se basa en la estrategia de divide y vencerás, donde el problema se particiona recursivamente alrededor de un pivote, reduciendo progresivamente el tamaño de los subproblemas.

El crecimiento observado en los tiempos experimentales muestra una tendencia subcuadrática, alineada con el término $n \log n$. Al multiplicar el tamaño de entrada por 10, el tiempo de ejecución aumenta en un factor mucho menor que en Bubble Sort, evidenciando mejor escalabilidad.

Impacto práctico en sistemas robóticos

En sistemas robóticos y embebidos, la eficiencia algorítmica impacta directamente en:

Latencia del sistema

Los sistemas de control en tiempo real requieren tiempos de respuesta determinísticos. Un algoritmo con complejidad O(n²) puede introducir latencias significativas cuando el volumen de datos aumenta, afectando la estabilidad del sistema de control.

En contraste, un algoritmo $O(n \log n)$ mantiene tiempos de ejecución más predecibles y escalables.

Procesamiento de datos sensoriales

En aplicaciones robóticas es común:

- Ordenar mediciones por prioridad o magnitud.
- Procesar datos provenientes de sensores LiDAR, visión o ultrasonido.
- Clasificar eventos en sistemas de navegación o planificación.

Un algoritmo ineficiente puede generar cuellos de botella en la cadena de procesamiento (sensor → procesamiento → decisión → actuador), incrementando el tiempo total del ciclo de control.

Consumo energético y recursos computacionales

En microcontroladores y sistemas embebidos, el uso intensivo de CPU implica:

- Mayor consumo energético.
- Reducción de autonomía en sistemas móviles.
- Posible saturación del procesador.

Un algoritmo $O(n \log n)$ reduce significativamente la carga computacional, optimizando el uso de recursos limitados.

Enlace del repositorio en github.

https://github.com/jeanacad2004-wq/U1A4-Comparacion-Ordenamientos