ITSRLL
INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

Ingeniería Mecatrónica

PROGRAMACIÓN AVANZADA

Enero – Junio 2025

M.C. Osbaldo Aragón Banderas

UNIDAD:

1	2	3	4	5

Actividad número: 4

Nombre de actividad:

Evaluación de Métodos de Ordenamiento

Actividad realizada por:

Martin Eduardo Montiel Salazar No. Control: 21030018

Guadalupe Victoria, Durango

Fecha de entrega:

12	02	2025

INTRODUCCIÓN

En la ciencia de la computación, el ordenamiento de datos es una tarea fundamental que afecta el rendimiento de muchas aplicaciones, desde la búsqueda de información hasta la optimización de bases de datos y la mejora de procesos computacionales. La eficiencia en la organización de datos influye directamente en la velocidad de ejecución de programas y en el uso óptimo de los recursos del sistema.

Existen diversos algoritmos de ordenamiento, cada uno con características particulares que los hacen adecuados para diferentes escenarios. Algunos, como el método burbuja, son simples de entender e implementar, pero presentan una alta complejidad computacional, lo que los vuelve ineficientes para grandes volúmenes de datos. Otros, como QuickSort, utilizan estrategias más avanzadas de división y conquista, lo que les permite ofrecer un rendimiento significativamente mejor en la mayoría de los casos prácticos.

A través de la experimentación con listas aleatorias de 10, 100 y 1000 elementos, se analizará cuál de estos métodos es más eficiente y en qué circunstancias cada uno podría ser útil. Los resultados obtenidos permitirán determinar la escalabilidad y la idoneidad de estos algoritmos en situaciones donde la velocidad de procesamiento es un factor crítico

OBJETIVO

Analizar y comparar el rendimiento de los algoritmos de ordenamiento Burbuja y QuickSort en diferentes tamaños de listas de datos. Se busca evaluar su eficiencia en términos de tiempo de ejecución y determinar cuál de los dos es más adecuado para distintos volúmenes de información

DESARROLLO (PROCEDIMIENTO)

1. Implementación de los algoritmos

Se desarrolló el método burbuja, que compara y cambia elementos adyacentes en un doble ciclo anidado.

```python
#Metodo Burbuja
def metodo_burbuja(lista):
  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
```

Figura 1: Método Burbuja

Se implementó el método QuickSort, utilizando una estrategia recursiva con división y conquista, donde el pivote es el último elemento de la lista.

```python
#Metodo QuickSort
def metodo_quicksort(lista):
  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 metodo_quicksort(menores) + [pivote] + metodo_quicksort(mayores)
```

Figura 2: Método QuickSort

2. Generación de listas aleatorias

Se generaron listas de 10, 100 y 1000 elementos con valores enteros aleatorios en el rango de 1 a 10,000.

```python
#Generar Lista Aleatoria
def generar_lista_aleatoria(tamaño):
  lista = [random.randint(1, 10000) for _ in range(tamaño)]
  return lista

lista_10 = generar_lista_aleatoria(10)
lista_100 = generar_lista_aleatoria(100)
lista_1000 = generar_lista_aleatoria(1000)
```

Figura 3: Listas aleatorias

3. Medición de tiempos de ejecución

Para medir la eficiencia de los algoritmos, se utilizaron pruebas de tiempo con la librería time.time(), para registrar el tiempo antes y después de ejecutar cada algoritmo en una copia de las listas generadas.

```
##Evaluacion De Tiempo De Respuesta Burbuja
inicio_10 = time.time()
metodo_burbuja(lista_10.copy())
fin_10 = time.time()
tiempo_10 = fin_10 - inicio_10

inicio_100 = time.time()
metodo_burbuja(lista_100.copy())
fin_100 = time.time()
tiempo_100 = fin_100 - inicio_100

inicio_1000 = time.time()
metodo_burbuja(lista_1000.copy())
fin_1000 = time.time()
tiempo_1000 = fin_1000 - inicio_1000
```

Figura 4: Tiempo de respuesta burbuja

```
##Evaluacion De Tiempo De Respuesta QuickSort
inicio_q10 = time.time()
metodo_quicksort(lista_10.copy())
fin_q10 = time.time()
tiempo_q10 = fin_q10 - inicio_q10

inicio_q100 = time.time()
metodo_quicksort(lista_100.copy())
fin_q100 = time.time()
tiempo_q100 = fin_q100 - inicio_q100

inicio_q1000 = time.time()
metodo_quicksort(lista_1000.copy())
fin_q1000 = time.time()
tiempo_q1000 = fin_q1000 - inicio_q1000
```

Figura 5: Tiempo de respuesta QuickSort

4. Muestra de resultados

Se imprimieron las listas originales y ordenadas junto con los tiempos de ejecución de cada algoritmo para su comparación.

```
print("Lista Original: ", lista_10)
print("Lista Ordenada burbuja: ", metodo_burbuja(lista_10.copy()))
print("Lista Ordenada QuickSort: ", metodo_quicksort(lista_10.copy()))
print(f"Tiempo de ejecución metodo burbuja para 10 elementos: {tiempo_10} segundos")
print(f"Tiempo de ejecución metodo QuikSort para 10 elementos: {tiempo_q10} segundos")
print()
print("Lista Original: ", lista_100)
print("Lista Ordenada burbuja: ", metodo_burbuja(lista_100.copy()))
print("Lista Ordenada QuickSort: ", metodo_quicksort(lista_100.copy()))
print(f"Tiempo de ejecución metodo burbuja para 100 elementos: {tiempo_100} segundos")
print(f"Tiempo de ejecución metodo QuikSort para 100 elementos: {tiempo_q100} segundos")
print()
print("Lista Original: ", lista_1000)
print("Lista Ordenada burbuja: ", metodo_burbuja(lista_1000.copy()))
print("Lista Ordenada QuickSort: ", metodo_quicksort(lista_1000.copy()))
print(f"Tiempo de ejecución metodo burbuja para 1000 elementos: {tiempo_1000} segundos")
print(f"Tiempo de ejecución metodo QuikSort para 1000 elementos: {tiempo_q1000} segundos")
```

Figura 6: Muestra de resultados

RESULTADOS

Tamaño de lista	Tiempo Burbuja (s)	Tiempo QuickSort (s)
10 elementos	0.0001175404	0.0000813075
100 elementos	0.0007250094	0.0002067089
1000 elementos	0.0452988147	0.0014934652

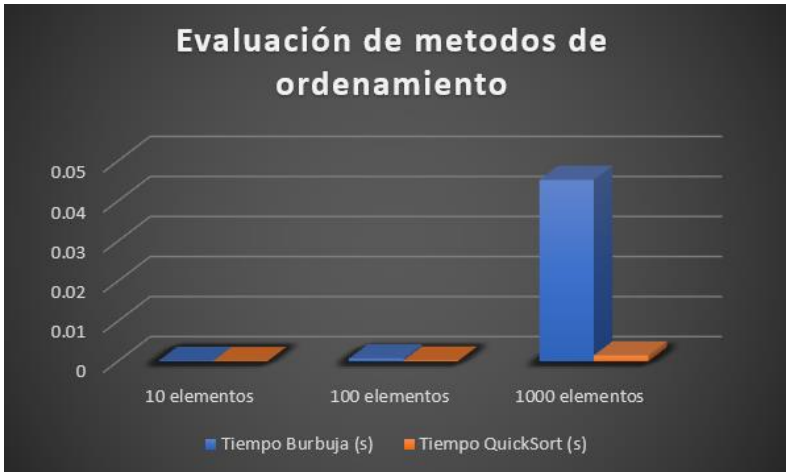

Ilustración 7: Grafico de tiempos

Observaciones:

- Para **10 elementos**, ambos métodos tienen tiempos similares, aunque QuickSort es ligeramente más rápido.
- A partir de **100 elementos**, QuickSort muestra una ventaja notable en velocidad.
- Con **1000 elementos**, QuickSort supera al método burbuja en eficiencia por un margen considerable.

ANALISIS COMPARATIVO

1. **Eficiencia temporal:**

El método burbuja tiene una complejidad de $O(n^2)$, lo que lo hace ineficiente para listas grandes. A medida que el tamaño de la lista aumenta, su tiempo de ejecución crece exponencialmente.

QuickSort tiene una complejidad promedio de $O(n \log n)$, lo que le permite manejar listas de mayor tamaño con tiempos de ejecución mucho menores en comparación con el método burbuja.

2. **Escalabilidad:**

- En listas pequeñas (10 elementos), la diferencia entre ambos algoritmos es mínima.
- En listas medianas (100 elementos), QuickSort ya es aproximadamente 3.5 veces más rápido que el método burbuja.
- En listas grandes (1000 elementos), QuickSort es 30 veces más rápido, lo que confirma su superioridad en escalabilidad.

3. **Uso de recursos:**

QuickSort usa recursividad y puede requerir más memoria en implementaciones sin optimización de pivotes. Mientras que el método burbuja, aunque consume menos memoria, es significativamente más lento, lo que lo hace poco práctico para volúmenes de datos considerables.

Repositorio en GitHub

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

https://github.com/EduMontiel19/Metodos_Ordenamiento-.git

CONCLUSION

La evaluación comparativa entre el método de burbuja y QuickSort evidencia diferencias significativas en su eficiencia y escalabilidad. Mientras que el método de burbuja presenta tiempos de ejecución considerablemente más altos debido a su complejidad. QuickSort demuestra ser una alternativa mucho más eficiente, especialmente en conjuntos de datos grandes, con una complejidad promedio. Los resultados obtenidos reflejan que QuickSort es una opción más viable para aplicaciones prácticas donde se requiere un rendimiento óptimo en el procesamiento de datos, mientras que el método de burbuja solo es útil para fines educativos o listas muy pequeñas.