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:

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

Actividad realizada por:

Carlos Diego Dominguez López - 21030006

Guadalupe Victoria, Durango

Fecha de entrega:

12	02	2025

INDICE

Introducción

El estudio y la implementación de algoritmos de ordenamiento son fundamentales en la optimización de software, especialmente en aplicaciones donde el procesamiento eficiente de datos es crucial, como en la robótica. En esta práctica, se explorarán dos algoritmos de ordenamiento ampliamente utilizados: el método de Burbuja y Quicksort.

Jupyter Notebook será la herramienta principal para desarrollar estos algoritmos en Python, permitiendo ejecutar pruebas de rendimiento y analizar su eficiencia en diferentes tamaños de datos.

Objetivo

Utilizar Jupyter Notebook para implementar y evaluar dos métodos de ordenamiento (Burbuja y Quicksort) en Python, analizando su rendimiento. Esta actividad refuerza el conocimiento de algoritmos, fundamentales para la optimización del software en aplicaciones de robótica. Además, se busca que los estudiantes practiquen la creación y organización de repositorios en GitHub, con el fin de que sirvan como apoyo en su currículo profesional.

Marco teórico

Jupyter Notebook

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto explicativo. Es ampliamente utilizado en el ámbito académico y en la industria debido a su facilidad para documentar y ejecutar código de manera interactiva.

Figura 1.1 Logo de Proyecto Jupyter

Entre sus principales características destacan:

- **Interactividad**: Permite ejecutar fragmentos de código de forma independiente, facilitando la experimentación y depuración.
- **Compatibilidad con múltiples lenguajes**: Aunque su uso más común es con Python, también soporta otros lenguajes como R y Julia.
- **Integración con bibliotecas científicas**: Facilita el análisis de datos y la visualización mediante bibliotecas como NumPy, Pandas y Matplotlib.
- **Facilidad de documentación**: Permite escribir explicaciones en formato Markdown, lo que mejora la presentación de los proyectos.

Método de Burbuja

El método de ordenamiento Burbuja es un algoritmo simple que funciona comparando y permutando elementos adyacentes hasta que la lista está completamente ordenada. Su funcionamiento se basa en múltiples pasadas por la lista, en las cuales los valores más grandes "flotan" hacia su posición final, como si fueran burbujas.

Funcionamiento del algoritmo:

1. Comparar el primer y el segundo elemento de la lista. Si están en el orden incorrecto, intercambiarlos.

2. Repetir el proceso con el segundo y tercer elemento, y así sucesivamente hasta el final de la lista.
3. En cada iteración completa, el elemento más grande se posiciona en su lugar final.
4. Repetir el proceso hasta que no haya intercambios en una pasada completa.

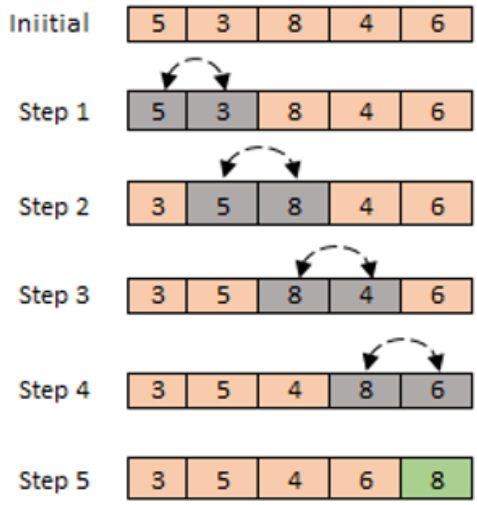

Figura 1.2 Ejemplo del método de ordenamiento burbuja

Método Quicksort

Quicksort es un algoritmo de ordenamiento eficiente y ampliamente utilizado, basado en la estrategia de **divide y vencerás**. Se selecciona un elemento de la lista llamado **pivote**, y los elementos restantes se dividen en dos subconjuntos:

- Un subconjunto con elementos menores que el pivote.
- Un subconjunto con elementos mayores que el pivote.

Luego, el proceso se repite de forma recursiva en cada subconjunto hasta que la lista queda completamente ordenada.

Quicksort es significativamente más rápido que el método de Burbuja en la mayoría de los casos, lo que lo convierte en una opción preferida para ordenamientos eficientes.

Funcionamiento del algoritmo:

1. Seleccionar un pivote (puede ser el primer elemento, el último o uno aleatorio).
2. Dividir la lista en dos partes: los menores y los mayores al pivote.
3. Aplicar Quicksort de manera recursiva a cada subconjunto.
4. Combinar las partes ordenadas.

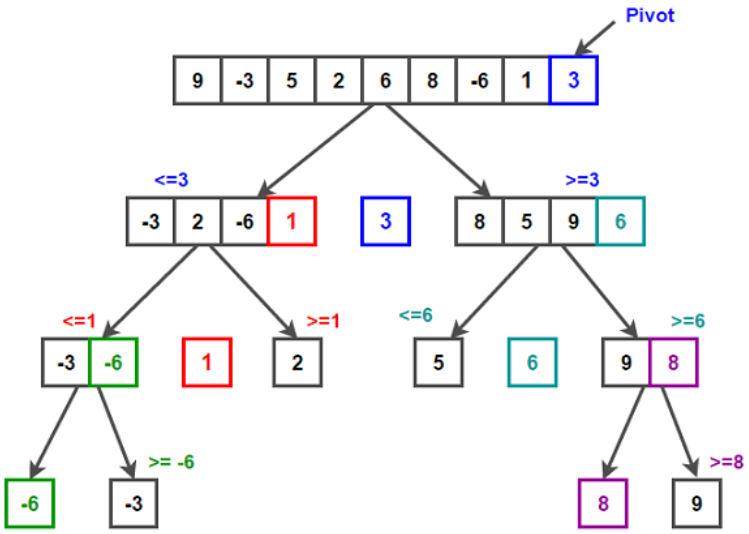

Figura 1.3 Ejemplo del método de ordenamiento Quicksort

Código Explicado

El código implementa el **método de ordenamiento Burbuja** en Python, mostrando cada paso del proceso.

1. **Funcionamiento**:
    - Recorre la lista varias veces, comparando elementos adyacentes.
    - Si un elemento es mayor que el siguiente, intercambian posiciones.
    - Este proceso se repite hasta que la lista queda ordenada.

```python
import time
def metodo_burbuja(lista):
    print("\nInicio del ordenamiento....")
    for i in range (len(lista)):
        print("\n Numero de pasada: ", i + 1)
        for j in range (0, len(lista) - i - 1):
            print("numero de elemento: ", i + 1)
            print(f"Comparacion de valor: {lista[j]} con valor: {lista[j+1]}")

            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                print(f"***cambio de elemento {j} por elemento {j+1}***")
    print("\n... Fin del ordenamiento")
    return lista
lista_ejemplo = [64, 0, 25, 12, 22, 1, 90]
lista_original = lista_ejemplo.copy()
lista_ordenada = metodo_burbuja(lista_original)

print("lista original: ", lista_ejemplo)
print("lista ordenada: ", lista_ordenada)
```

Figura 2.1 Código para el método de ordenamiento burbuja

2. **Ejecución**:

   - Se prueba con una lista de números desordenados.
   - Se muestra cada comparación y cambio realizado.
   - Finalmente, se imprime la lista original y la lista ordenada.

```
Inicio del ordenamiento....

 Numero de pasada:  1
numero de elemento:  1
Comparacion de valor: 64 con valor: 0
***cambio de elemento 0 por elemento 1***
numero de elemento:  1
Comparacion de valor: 64 con valor: 25
***cambio de elemento 1 por elemento 2***
numero de elemento:  1
Comparacion de valor: 64 con valor: 12
***cambio de elemento 2 por elemento 3***
numero de elemento:  1
Comparacion de valor: 64 con valor: 22
***cambio de elemento 3 por elemento 4***
numero de elemento:  1
Comparacion de valor: 64 con valor: 1
***cambio de elemento 4 por elemento 5***
numero de elemento:  1
Comparacion de valor: 64 con valor: 90
```

```
 Numero de pasada:  2
numero de elemento:  2
Comparacion de valor: 0 con valor: 25
numero de elemento:  2
Comparacion de valor: 25 con valor: 12
***cambio de elemento 1 por elemento 2***
numero de elemento:  2
Comparacion de valor: 25 con valor: 22
***cambio de elemento 2 por elemento 3***
numero de elemento:  2
Comparacion de valor: 25 con valor: 1
***cambio de elemento 3 por elemento 4***
numero de elemento:  2
Comparacion de valor: 25 con valor: 64

 Numero de pasada:  3
numero de elemento:  3
Comparacion de valor: 0 con valor: 12
numero de elemento:  3
Comparacion de valor: 12 con valor: 22
numero de elemento:  3
Comparacion de valor: 22 con valor: 1
***cambio de elemento 2 por elemento 3***
numero de elemento:  3
Comparacion de valor: 22 con valor: 25

 Numero de pasada:  4
numero de elemento:  4
Comparacion de valor: 0 con valor: 12
numero de elemento:  4
Comparacion de valor: 12 con valor: 1
***cambio de elemento 1 por elemento 2***
numero de elemento:  4
Comparacion de valor: 12 con valor: 22

 Numero de pasada:  5
numero de elemento:  5
Comparacion de valor: 0 con valor: 1
numero de elemento:  5
Comparacion de valor: 1 con valor: 12

 Numero de pasada:  6
numero de elemento:  6
Comparacion de valor: 0 con valor: 1

 Numero de pasada:  7

... Fin del ordenamiento
lista original:  [64, 0, 25, 12, 22, 1, 90]
lista ordenada:  [0, 1, 12, 22, 25, 64, 90]
```

Figura 2.2 Resultados del código de ordenamiento por el método burbuja

El código implementa **Quicksort** en Python de forma recursiva.

Funcionamiento

1. Si la lista tiene menos de 2 elementos, se devuelve sin cambios.
2. Se elige el **último elemento** como pivote.
3. Se dividen los elementos en **menores** y **mayores o iguales** al pivote.
4. Se ordenan recursivamente las sublistas y se combinan.

```
def quick_sort(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 quick_sort(menores) + [pivote] + quick_sort(mayores)

lista_ejemplo = [64, 0, 25, 12, 22, 1, 90]
print(lista_ejemplo[:-1])
lista_original = lista_ejemplo.copy()
lista_ordenada = quick_sort(lista_original)

print("lista original: ", lista_ejemplo)
print("lista ordenada: ", lista_ordenada)
```

Figura 2.3 Código para el método de ordenamiento Quicksort

Ejecución

- Se prueba con una lista desordenada.
- Se imprime la lista original y la lista ordenada.

Resultado esperado: La lista se ordena eficientemente dividiéndola en subconjuntos más pequeños.

```
[64, 0, 25, 12, 22, 1]
lista original:  [64, 0, 25, 12, 22, 1, 90]
lista ordenada:  [0, 1, 12, 22, 25, 64, 90]
```

Figura 2.4 Resultados del código de ordenamiento por el método Quicksort

El código mide el **tiempo de ejecución** de una función en Python usando el módulo time.

```python
import time

def funcion_ejemplo():
    suma = 0
    for i in range(1_000_000):
        suma += i
    return suma

inicio = time.time()

resultado = funcion_ejemplo()

fin = time.time()

print("El tiempo de ejecucion fue: ", fin - inicio, "segundos")
```

```
El tiempo de ejecucion fue:  0.29399967193603516 segundos
```

Figura 2.5 Código para medir la ejecución del código de ordenamiento burbuja y Quicksort

Funcionamiento

1. **funcion_ejemplo()**:
    - Suma los números del 0 al 999,999 en un bucle.
2. **Medición del tiempo**:
    - Se registra el **tiempo de inicio** (inicio = time.time()).
    - Se ejecuta la función y almacena el resultado.
    - Se registra el **tiempo de finalización** (fin = time.time()).
3. **Cálculo del tiempo transcurrido**:
    - Se imprime la diferencia fin - inicio, indicando el tiempo de ejecución en segundos.

Conclusión

El estudio y la implementación de algoritmos en Python son fundamentales para comprender la eficiencia computacional y la optimización del software. En esta práctica, se han analizado dos métodos de ordenamiento: Burbuja y Quicksort, así como la medición del tiempo de ejecución de una función.

El estudio de estos algoritmos permite comprender la importancia de elegir métodos de ordenamiento eficientes en la programación y la robótica. Mientras que el método de Burbuja es útil para entender la mecánica básica del ordenamiento, Quicksort ofrece un rendimiento mucho mejor para grandes volúmenes de datos.

El uso de Jupyter Notebook en esta práctica facilita la implementación y el análisis comparativo, permitiendo a los estudiantes desarrollar habilidades clave en algoritmia, optimización y documentación de código.