Algoritmos de Búsqueda y Ordenamiento

Lucio Rivas – Lucior0900@gmail.com

Maximo Ponce - maximoponce31@gmail.com

Materia: Programación I

Fecha de Entrega: 9 de junio de 2025

Índice

1- Introducción

2- Marco Teórico

3- Caso Práctico

4- Metodología Utilizada

5- Resultados Obtenidos

6- Conclusiones

7- Bibliografía

8- Anexos

Introducción

Los algoritmos de búsqueda y ordenamiento son herramientas fundamentales en la programación, ya que permiten organizar y acceder a la información de forma eficiente. Comprender cómo funcionan y cuándo aplicarlos es clave para resolver problemas cotidianos del desarrollo de software.

En este trabajo se investigan algunos de los algoritmos más utilizados en Python, como el ordenamiento por burbujeo (Bubble Sort) y la búsqueda binaria. Se analiza su funcionamiento, se desarrollan ejemplos prácticos y se reflexiona sobre su importancia en la optimización de programas.

Marco Teórico

Algoritmos de Ordenamiento

Los algoritmos de ordenamiento son procesos que permiten organizar una colección de datos (por ejemplo, números o palabras) según un criterio definido, como de menor a mayor o alfabéticamente.

Uno de los más conocidos es el Bubble Sort (ordenamiento por burbujeo), que compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. El proceso se repite varias veces hasta que la lista está completamente ordenada. Es un algoritmo simple, ideal para entender la lógica de ordenamiento, aunque no es eficiente con grandes volúmenes de datos.

Algoritmos de Búsqueda

La búsqueda de datos consiste en encontrar un valor específico dentro de una estructura. Existen distintos tipos de búsqueda, entre los cuales se destacan:

Búsqueda Lineal: recorre los elementos uno por uno hasta encontrar el deseado. Es sencilla, pero lenta en listas largas.

Búsqueda Binaria: solo se puede aplicar en listas ordenadas. Consiste en dividir el rango de búsqueda a la mitad en cada paso, lo que permite encontrar un valor mucho más rápido.

Comparación de Complejidad

Bubble Sort: tiene una complejidad temporal de $O(n^2)$, lo que significa que su rendimiento empeora mucho cuando hay muchos datos.

Búsqueda Binaria: tiene una complejidad O(log n), mucho más eficiente que la búsqueda lineal, pero requiere que los datos estén previamente ordenados.

Caso Práctico

Para ejemplificar el funcionamiento de los algoritmos estudiados, se desarrolló un programa en Python que:

Ordena una lista de números utilizando el algoritmo Bubble Sort.

Busca un número específico utilizando Búsqueda Binaria.

Código:

Metodología Utilizada

El trabajo se realizó siguiendo una serie de etapas ordenadas:

1- Selección del tema: Se eligieron los algoritmos de búsqueda y ordenamiento por ser conceptos fundamentales en programación y de fácil implementación en Python.

2- Investigación teórica: Se consultaron libros, documentación oficial y recursos educativos en línea para comprender el funcionamiento de los algoritmos seleccionados.

3- Desarrollo práctico: Se programaron en Python los algoritmos Bubble Sort y Búsqueda Binaria, y se probaron con distintos conjuntos de datos.

4-Evaluación: Se analizaron los resultados para comprobar la eficiencia y la lógica del código.

5-Documentación: Se elaboró este informe, acompañado de capturas, código comentado y reflexiones.

Resultados Obtenidos

La función bubble_sort() ordenó correctamente la lista de números de menor a mayor.

La función binary_search() permitió encontrar de forma eficiente el número buscado dentro de la lista ordenada.

Se comprobó que, si bien el Bubble Sort es fácil de implementar, no es el más rápido para grandes volumenes de datos.

La búsqueda binaria demostró su rapidez, pero depende de que los datos estén previamente ordenados.

Se logró comprender el funcionamiento interno de ambos algoritmos mediante pruebas y observación directa del código.

Conclusiones

El desarrollo de este trabajo permitió comprender de forma práctica la utilidad de los algoritmos de búsqueda y ordenamiento en la programación. Se aprendió que elegir el algoritmo correcto depende del tipo de datos y del objetivo que se quiere alcanzar.

El Bubble Sort, aunque poco eficiente con grandes cantidades de datos, es ideal para introducirse en el ordenamiento por su sencillez. Por otro lado, la búsqueda binaria demostró ser muy rápida y eficaz, aunque requiere que la lista esté previamente ordenada.

Estos conceptos forman la base de problemas más complejos que se pueden resolver en programación, por lo que dominarlos es clave para seguir avanzando en el estudio del desarrollo de software.

Bibliografía

Python Software Foundation. (2024). Documentación oficial de Python. https://docs.python.org/3/

Khan Academy. Algoritmos de ordenamiento y búsqueda. https://es.khanacademy.org/computing/computer-science/algorithms

Anexos

Captura de pantalla del programa funcionando:

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Intercambio de elementos si están en el orden incorrecto
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

```python
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid  # Devuelve la posición
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1  # Si no se encuentra el valor
```

```python
# Datos de prueba
datos = [45, 12, 78, 4, 33, 56, 9]
print("Lista original:", datos)

# Ordenamiento
bubble_sort(datos)
print("Lista ordenada:", datos)

# Búsqueda de un número
valor_buscado = 33
pos = binary_search(datos, valor_buscado)
if pos != -1:
    print(f"El número {valor_buscado} fue encontrado en la posición {pos}.")
else:
    print(f"El número {valor_buscado} no fue encontrado.")
```

```
Lista original: [45, 12, 78, 4, 33, 56, 9]
Lista ordenada: [4, 9, 12, 33, 45, 56, 78]
El número 33 fue encontrado en la posición 3.
```

Repositorio en GitHub: https://github.com/RivasLucio/tp-integradot-utn

Video explicativo :https://youtu.be/ygHZVEeC8WE