Título del trabajo: “Algoritmos de Búsqueda y Ordenamiento en Python”

Alumnos:

Bruno Croppi - croppibruno@gmail.com

Ignacio Figueroa - ignaciofigueroadev@gmail.com

- **Materia**: Programación I
- **Profesor/a**: Sebastian Bruselario
- **Fecha de Entrega**: (Formato: día/mes/año)

Índice

---

1. Introducción

Se abordará el tema de Algoritmos de Búsqueda y Ordenamiento por su utilidad en la programación para organizar y encontrar datos de manera eficiente. Es un concepto clave en el desarrollo de software.

El objetivo es analizar e implementar algunos de estos algoritmos y comprender su funcionamiento.

---

2. Marco Teórico

Los algoritmos de búsqueda y ordenamiento son esenciales en el mundo de la programación, ya que permiten organizar y acceder a la información de manera eficiente. Su estudio es clave para optimizar el rendimiento de cualquier sistema que procese datos.

Búsqueda:

Los algoritmos de búsqueda tienen como objetivo localizar un valor específico dentro de

una estructura de datos, como una lista o un array.

- La **búsqueda lineal** recorre los elementos uno por uno hasta encontrar el valor o llegar al final.
- La **búsqueda binaria**, en cambio, es más eficiente, pero requiere que la lista esté previamente ordenada. Divide el conjunto en mitades sucesivas para encontrar el elemento.

Ordenamiento:

Los algoritmos de ordenamiento se utilizan para **organizar los datos en un orden determinado**, ya sea ascendente o descendente.
Algunos de los más conocidos son:

- **Burbuja**: compara elementos adyacentes e intercambian sus posiciones si están en el orden incorrecto.

- **Inserción**: toma cada elemento y lo coloca en la posición adecuada respecto a los anteriores.

- **Selección**: busca el elemento mínimo y lo posiciona en el lugar correspondiente, repitiendo el proceso.

Estos algoritmos, si bien son simples, permiten comprender los principios de eficiencia algorítmica y sirven como base para técnicas más avanzadas.

Estructuras

Se trabaja principalmente con **listas o arrays**, estructuras lineales que permiten almacenar elementos en posiciones consecutivas y acceder a ellos mediante índices.

3. Caso Práctico

Para este trabajo implementamos tres algoritmos clásicos en Python: ordenamiento por inserción, búsqueda lineal y búsqueda binaria.

El objetivo fue ordenar una lista de números y luego buscar un valor específico. Primero se usó el algoritmo de **inserción** para ordenar la lista. Luego se aplicaron dos métodos de búsqueda:

- La **búsqueda lineal**, que recorre todos los elementos.
- La **búsqueda binaria**, más eficiente pero que requiere la lista ordenada.

```
def ordenamiento_insercion(lista):
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        while j >= 0 and clave < lista[j]:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
    return lista
```

```
def busqueda_lineal(lista, objetivo):
    for i in range(len(lista)):
        if lista[i] == objetivo:
            return i
    return -1
```

```
def busqueda_binaria(lista, objetivo):
    inicio, fin = 0, len(lista) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if lista[medio] == objetivo:
            return medio
        elif lista[medio] < objetivo:
            inicio = medio + 1
        else:
            fin = medio - 1
    return -1

# Prueba
datos = [42, 7, 13, 29, 18]
ordenada = ordenamiento_insercion(datos.copy())
print("Lista ordenada:", ordenada)
print("Lineal (29):", busqueda_lineal(ordenada, 29))
print("Binaria (29):", busqueda_binaria(ordenada, 29))
```

El resultado fue correcto: la lista quedó ordenada y ambas búsquedas encontraron el valor 29. Se validó que el código funciona como se esperaba y permite comparar la lógica de cada algoritmo.

---

4. Metodología Utilizada

Para hacer este trabajo empezamos investigando un poco sobre los algoritmos más comunes de búsqueda y ordenamiento en Python. Usamos apuntes de la materia, documentación oficial y algunos ejemplos simples que encontramos en internet.

Todo el código lo hicimos en **Python** usando **Visual Studio Code**. No usamos librerías

externas, ya que la idea era entender bien cómo funcionan estos algoritmos escribiéndonos desde cero.

En cuanto a la división de tareas, nos organizamos más o menos así:

- **Bruno** se encargó de armar los algoritmos de ordenamiento y revisar que funcionen bien.

- **Ignacio** programó las búsquedas y probó el código con distintos datos. Después nos juntamos a repasar todo, agregar comentarios y asegurarnos de que corriera bien.

---

5. Resultados Obtenidos

Al ejecutar el programa, los tres algoritmos funcionaron correctamente. La lista se ordenó bien usando el algoritmo de inserción, y tanto la búsqueda lineal como la binaria localizaron el número buscado sin errores.

Se hicieron varias pruebas con diferentes listas y valores, y en todos los casos los resultados fueron los esperados. Solo tuvimos que corregir un pequeño error en la lógica de la búsqueda binaria al principio, pero se solucionó rápido.

También notamos que, aunque ambos métodos de búsqueda dieron el mismo resultado, la búsqueda binaria es más eficiente cuando la lista es larga, siempre y cuando esté ordenada.

En resumen, el código quedó funcionando correctamente, fue fácil de entender y cumplió con los objetivos planteados

6. Conclusiones

Al realizar este trabajo, aprendimos a implementar desde cero algoritmos básicos de ordenamiento y búsqueda en Python, entendiendo mejor su lógica y funcionamiento. Esto nos ayudó a valorar la importancia de organizar y buscar datos de forma eficiente, un concepto fundamental en la programación.

El tema es muy útil porque estos algoritmos son la base para muchas aplicaciones que manejan grandes cantidades de información, y conocerlos facilita abordar problemas más complejos en proyectos futuros.

Como posibles mejoras, podríamos incorporar algoritmos más avanzados o comparar el rendimiento con listas mucho más grandes, usando herramientas para medir tiempos de ejecución.

Durante el desarrollo, nos enfrentamos a algunos errores lógicos en la búsqueda binaria, pero los resolvimos revisando paso a paso la implementación y probando con distintos casos.

En definitiva, el trabajo fue una buena práctica para fortalecer nuestros conocimientos y mejorar nuestras habilidades en programación.

7. Bibliografía

–'–µl_H µ_‘ Listado de fuentes consultadas, utilizando normas básicas APA u otro formato consistente. Se sugiere incluir:

- Material de clase de la materia Programación I, Profesor Sebastián Bruselario.
- Recursos en línea consultados para la implementación y comprensión de los algoritmos.
- OpenAI. (2025). *ChatGPT* [Modelo de lenguaje]. Disponible en https://chat.openai.com/

8. Anexo

- Enlace a video en Youtube
- Enlace a repositorio en GitHub
