Algoritmos de Búsqueda y Ordenamiento

●**Alumnos:**

● Vazquez Franco– fraan_v1@hotmail.com

● Larrosa Sebastián – 11sclarrosa11@gmail.com

●**Materia**: Programación I

●**Profesor**: Cinthia Rigoni

●**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

1 - Introducción

Los algoritmos de búsqueda y ordenamiento son fundamentales en programación, ya que optimizan el procesamiento de datos y mejoran la eficiencia de aplicaciones. Este trabajo explora su implementación en Python, comparando su rendimiento y aplicaciones en casos reales como bases de datos o sistemas de recomendación.
Los objetivos incluyen:

1. Analizar métodos clásicos (QuickSort, Búsqueda Binaria),
2. Evaluar su complejidad algorítmica
3. Demostrar su utilidad en un proyecto simulado.

¿Por qué se eligió este tema?

El presente trabajo aborda los algoritmos de búsqueda y ordenamiento, conceptos fundamentales en la programación y el tratamiento eficiente de datos. Se eligió este tema debido a su gran relevancia en múltiples áreas de la informática, desde la inteligencia artificial hasta el desarrollo web y el análisis de grandes volúmenes de información.

¿Qué importancia tiene en la programación?

Los algoritmos de búsqueda permiten localizar datos dentro de estructuras como listas o árboles, mientras que los de ordenamiento reorganizan esos datos de acuerdo con ciertos criterios. Comprender su funcionamiento no solo mejora el rendimiento de los programas, sino que también permite tomar decisiones informadas al diseñar soluciones.

¿Qué objetivos se propone alcanzar con el desarrollo del trabajo?
Este trabajo tiene como objetivo:

1. Comprender los distintos tipos de algoritmos de búsqueda y ordenamiento.
2. Comparar su eficiencia y aplicabilidad.
3. Implementar un caso práctico para validar su funcionamiento y rendimiento

2 - Marco Teórico

Algoritmos de ordenamiento

Los algoritmos de ordenamiento permiten reorganizar una colección de elementos según un criterio específico, como de mayor a menor o alfabéticamente (Cormen et al., 2009).

Entre los más comunes se encuentran:

- Bubble Sort: compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto (Knuth, 1998).

- Insertion Sort: construye una lista ordenada tomando los elementos uno a uno e insertándolos en su posición adecuada (Sedgewick & Wayne, 2011).

- Selection Sort: selecciona el elemento más pequeño del arreglo y lo coloca en su posición final (Cormen et al., 2009).

Algoritmos de búsqueda

Los algoritmos de búsqueda son procedimientos estructurados diseñados para encontrar un elemento específico dentro de un conjunto de datos, como listas, arreglos, árboles o grafos.

Su principal objetivo es reducir el tiempo y los recursos
necesarios para localizar información, por lo que son esenciales en áreas como la computación, la inteligencia artificial, las bases de datos y la ciberseguridad.

Estos algoritmos tienen aplicaciones fundamentales en múltiples contextos:

- Bases de datos: permiten encontrar registros de manera eficiente.

- Motores de búsqueda: indexan y recuperan información rápidamente (por ejemplo, Google).

- Sistemas de recomendación: filtran productos, películas o música según las preferencias del usuario.

- Videojuegos: calculan caminos óptimos (pathfinding) utilizando algoritmos como A* (Russell & Norvig, 2021).

- Ciberseguridad: detectan patrones o intrusiones en grandes volúmenes de datos.

Tipos principales de algoritmos de búsqueda

- Búsqueda secuencial (lineal): revisa elemento por elemento. Es simple, pero poco eficiente en grandes volúmenes de datos (Knuth, 1998).

- Búsqueda binaria: divide repetidamente el conjunto de datos en mitades. Requiere que los datos estén previamente ordenados (Cormen et al., 2009).

Búsqueda en árboles:

- Búsqueda en profundidad (DFS) y en amplitud (BFS): exploran estructuras jerárquicas o grafos, útiles en inteligencia artificial (Russell & Norvig, 2021).

- Búsqueda heurística:
  Algoritmos como A* y Dijkstra se aplican para encontrar rutas óptimas en grafos ponderados (Pearl, 1984). En lugar de explorar exhaustivamente todas las posibilidades, la heurística se basa en reglas o funciones que estiman la cercanía a una solución óptima

- Búsqueda por hash: permite un acceso casi instantáneo a los datos mediante funciones de mapeo (Sedgewick & Wayne, 2011).

3 - Caso Práctico

Para garantizar la máxima seguridad en un garaje privado, se implementará un sistema de control de acceso que permitirá al personal autorizado ingresar la matrícula del vehículo, mostrará automáticamente los datos relevantes del conductor y determinará si se concede o deniega el acceso.

El guardia de seguridad, ubicado en el ingreso al garaje, introducirá la patente del automóvil, entonces el sistema consultará la base de datos de vehículos autorizados y le mostrará en pantalla la información del conductor (vinculando patente con DNI). Cotejando el número de DNI con patente el guardia le indicará al conductor si el acceso está permitido o denegado.

Este mecanismo asegura un control estricto de los ingresos al garaje, permitiendo verificar en tiempo real la identidad de cada conductor y su autorización para acceder al recinto privado.

Patentes:

Número	Patente	DNI
1	ABC123	12123456
2	ABC234	18128458
3	ABC345	16424486
4	ABC456	23423456
5	ABC567	30123445

6	DFG123	11234868
7	DFG234	36623666
8	DFG345	81284861
9	DFG456	12123456
10	DFG567	44483456
11	HIJ567	23123456
12	KLM123	21923456
13	KLM234	86123456
14	KLM345	62123456
15	KLM456	92219456

> El guardia, cuando se presente un vehículo, deberá ingresar el número de patente, el pedido de información será cumplido con el algoritmo diseñado y éste buscará en su pequeña base de datos, mostrándola en pantalla junto con el número de DNI del conductor.

Utilizaremos un **algoritmo de búsqueda binario** para lograrlo, que será codificado en Python, este podrá ser encontrado en el Anexo de este documento.

Como hemos señalado anteriormente, la selección de este algoritmo en particular se debe a dos características clave del conjunto de datos:

1. Tamaño manejable: La lista de patentes no es excesivamente extensa, lo que hace viable su ordenación previa y mantenimiento.
2. Datos ordenados: La naturaleza previamente ordenada de nuestra base de datos de patentes hace que la búsqueda binaria sea ideal para este caso.

Si bien podríamos haber optado por un algoritmo de búsqueda secuencial, descartamos esta alternativa porque:

- La búsqueda binaria ofrece mayor eficiencia (O (log n) vs O(n) de la secuencial)
- Está específicamente diseñada para trabajar con listas ordenadas
- Proporciona mejor rendimiento incluso con pequeñas cantidades de datos
- Reduce el tiempo promedio de respuesta del sistema

Por estas razones, consideramos que la búsqueda binaria representa la solución óptima para nuestro sistema de control de acceso vehicular, garantizando un equilibrio perfecto entre eficiencia computacional y simplicidad de implementación

En principio se discutió en forma grupal un problema ,simple, que se adapte al caso práctico requerido. La primera opción que contemplamos fue la realización de un código que pueda realizar

una búsqueda de raíces por medio del método Newton, sin embargo se abandonó la idea dada la complejidad de la función, que requiere conocimientos de derivadas.

Si bien podría haber sido logrado al contar con más tiempo, y por supuesto, contar con una base de datos lo suficientemente amplia para poder poner en funcionamiento dicho algoritmo. Entendimos que las herramientas necesarias nos exigían otro nivel de conocimiento, superior. Por lo tanto comenzamos a buscar alternativas a nuestro alcance y, si bien de cierta complejidad, posible de ser llevadas a cabo. Inmediatamente se nos ocurrió que con una base de datos pequeña y ordenada podríamos escribir un código preciso y que cumpliera con los requerimientos que exige la materia. Entonces pensamos escenarios posibles en donde podrían caber nuestros requisitos ya mencionados.

El escenario, entonces, fue un garage con una cantidad de autos limitada (base de datos pequeña) para que alguien deba hacer una búsqueda (un portero que permita o impida el acceso) y que luego de la búsqueda se coteje información (comparar patente del vehículo y DNI correcto). Es decir, en esta oportunidad en primera instancia pusimos las exigencias y luego buscamos un tema que se ajuste.

4 - Metodología Utilizada

La realización del trabajo se llevó a cabo en las siguientes etapas:

1. Búsqueda y recopilación de información teórica a partir de fuentes confiables.
2. Implementación de los algoritmos estudiados utilizando el lenguaje Python.
3. Ejecución de pruebas con distintos conjuntos de datos.
4. Registro de los resultados obtenidos y verificación de la funcionalidad.
5. Apoyo en herramientas de inteligencia artificial (ChatGPT, Gemini) para la validación y corrección de errores.
6. Publicación en Git del proyecto
7. Redacción del presente informe y preparación de los anexos correspondientes.
8. Reflexión grupal y redacción de conclusiones.
9. Producción del video tutorial

5 - Resultados Obtenidos

- El programa ordenó correctamente la base de datos según las patentes, permitiendo el uso adecuado del algoritmo de búsqueda binaria.

- La búsqueda binaria localizó de forma eficiente las matrículas registradas, entregando en pantalla el DNI correspondiente.

- En casos de matrículas no registradas, el sistema respondió de manera clara, indicando la denegación de acceso.

- Se comprendió la importancia de mantener los datos ordenados para aprovechar las ventajas de la búsqueda binaria.

- Se observó la diferencia de rendimiento y estructura respecto a otros algoritmos de

búsqueda más simples, como la búsqueda lineal.

- Se validó la funcionalidad del sistema mediante pruebas con múltiples entradas y se garantizó su correcto funcionamiento ante distintos escenarios.

6 - Conclusiones

Conclusión del caso práctico

1. Nuestro caso práctico se trata de un pequeño código que ordena las patentes de vehículos que deben ingresar a garaje que realiza una búsqueda binaria para verificar el acceso.

2. La lista **base_de_datos** debe estar ordenada alfabéticamente (base_datos.sort() - **Bubble Sort**), condición para el funcionamiento del algoritmo de búsqueda tipo binario. El bucle while divide repetidamente el espacio de búsqueda a la mitad: si la patente buscada es mayor que la del medio busca en la mitad derecha, si es menor busca en la mitad izquierda, si es igual retorna el DNI asociado.

3. El bucle termina cuando izquierda es mayor a derecha. Retorna **None** si la patente no existe.

4. Es muy importante destacar que cada iteración descarta la mitad de los elementos restantes, siendo el accionar típico de un algoritmo de búsqueda binario.

Conclusión de los algoritmos de búsqueda y ordenamiento

1. Los algoritmos de búsqueda y ordenamiento constituyen fundamentos esenciales en el campo de la informática. A través del desarrollo de este trabajo, se comprendió que su aplicación no solo mejora el rendimiento de los programas, sino que también permite optimizar significativamente los procesos de búsqueda y organización de datos.

2. La implementación de la búsqueda binaria demostró ser altamente eficiente en comparación con otros métodos más simples, como la búsqueda secuencial. Sin embargo, también se evidenció una de sus principales limitaciones: requiere que los datos estén previamente ordenados para funcionar correctamente.

3. Este proyecto permitió reforzar la importancia de seleccionar el algoritmo más adecuado en función del problema a resolver y de las características del conjunto de datos.

7 - Bibliografía

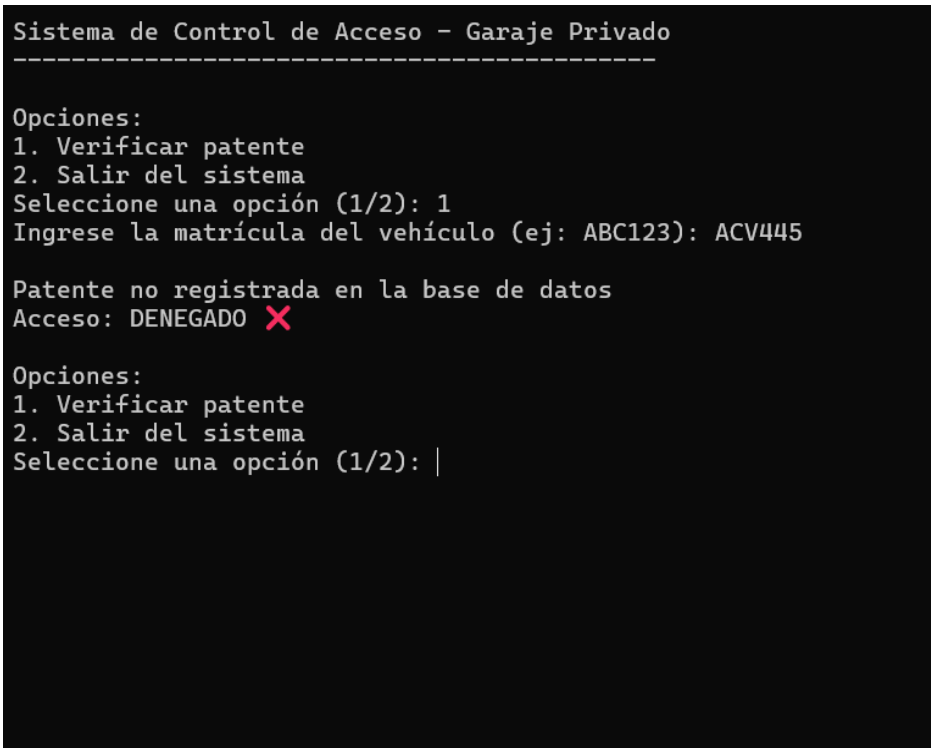
- ***Búsqueda binaria (artículo) | Algoritmos -*** ***https://es.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search***

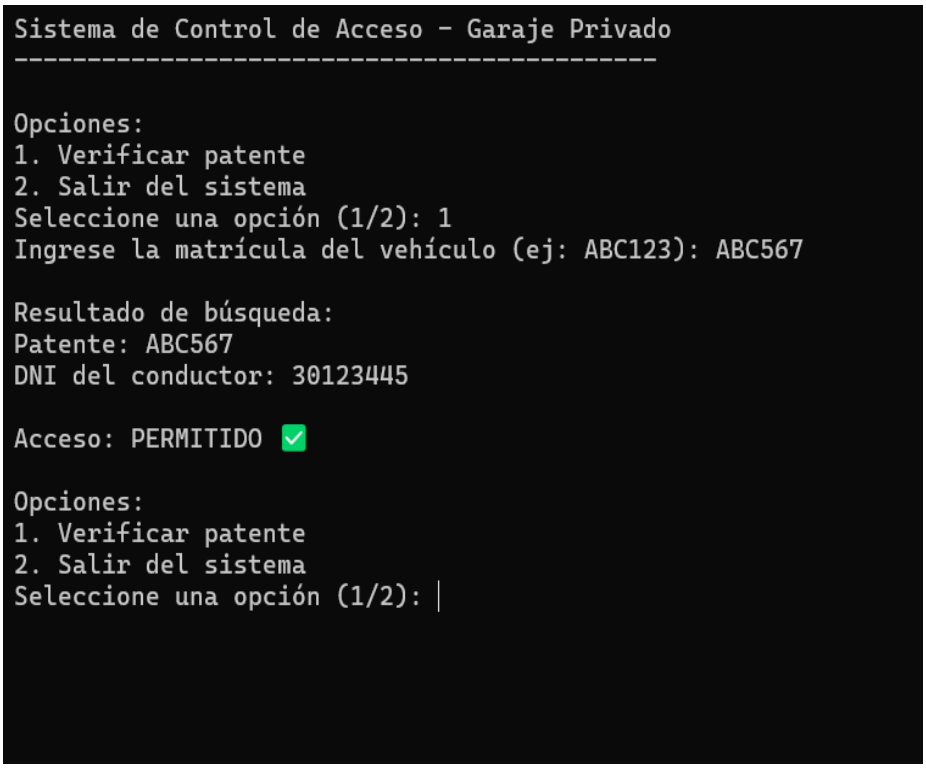
- ***Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.). Addison-Wesley. -*** ***https://seriouscomputerist.atariverse.com/media/pdf/book/Art%20of%20Computer%20Programming%20-%20Volume%203%20(Sorting%20&%20Searching).pdf***

- ***Python 1.13.4 documentation - https://docs.python.org/es/3.13/***

8 - Anexos

Imágenes del programa en funcionamiento:





Código del programa en python:

```
def buscar_patente(patentes_ordenadas, patente_buscada):
```

```
    """Implementa búsqueda binaria para encontrar una patente"""
    izquierda, derecha = 0, len(patentes_ordenadas) - 1

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        patente_actual, dni = patentes_ordenadas[medio]

        if patente_actual == patente_buscada:
            return dni
        elif patente_actual < patente_buscada:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    return None
def busqueda_binaria():
    # Base de datos ordenada por patente
    base_datos = [
        ("ABC123", "12123456"),
        ("ABC234", "18128458"),
        ("ABC345", "16424486"),
        ("ABC456", "23423456"),
        ("ABC567", "30123445"),
        ("DFG123", "11234868"),
        ("DFG234", "36623666"),
        ("DFG345", "81284861"),
        ("DFG456", "12123456"),
        ("DFG567", "44483456"),
        ("HIJ567", "23123456"),
        ("KLM123", "21923456"),
        ("KLM234", "86123456"),
        ("KLM345", "62123456"),
        ("KLM456", "92219456")
    ]

    # Ordenamos por patente (requisito para búsqueda binaria)
    base_datos.sort()

    print("Sistema de Control de Acceso - Garaje Privado")
    print("--------------------------------------------")

    while True:
        print("\nOpciones:")
        print("1. Verificar patente")
```

```
        print("2. Salir del sistema")

        opcion = input("Seleccione una opción (1/2): ")

        if opcion == "1":
            patente = input("Ingrese la matrícula del vehículo (ej: ABC123): ").strip().upper()

            dni = buscar_patente(base_datos, patente)

            if dni:
                print("\nResultado de búsqueda:")
                print(f"Patente: {patente}")
                print(f"DNI del conductor: {dni}")
                print("\nAcceso: PERMITIDO ✅")
            else:
                print("\nPatente no registrada en la base de datos")
                print("Acceso: DENEGADO ❌")

        elif opcion == "2":
            print("Saliendo del sistema...")
            break

        else:
            print("Opción no válida. Por favor ingrese 1 o 2.")
if __name__ == "__main__":
    busqueda_binaria()
```