Algoritmos de Búsqueda y Ordenamiento

Alumnos:

Magdalena Darchez - magdalenadarchez@gmail.com

Agustín Díaz - agusdiaz025@gmail.com

Materia: Programación I

Profesor: Sebastián Bruselario

Fecha de entrega: 9 de Junio de 2025

Índice:

Introducción

En este trabajo se abordarán los algoritmos de búsqueda y ordenamiento, conceptos clave en el desarrollo de software. Se eligió este tema por su importancia en la eficiencia y organización de datos, esenciales en cualquier programa.

La búsqueda permite localizar un elemento específico dentro de un conjunto de datos, mientras que el ordenamiento organiza esos datos según un criterio para facilitar su uso y análisis.

En programación, estos procesos son fundamentales en aplicaciones como bases de

datos, sistemas de archivos, inteligencia artificial y análisis de grandes volúmenes de datos.
Los objetivos del trabajo son analizar y aplicar algoritmos de búsqueda y ordenamiento, comprendiendo su importancia en la eficiencia y rendimiento, esenciales para optimizar y mejorar el código.

Marco teórico

Un algoritmo es un conjunto de instrucciones definidas, ordenadas y fijas para resolver un problema o desarrollar una tarea. De este modo, en programación los algoritmos son muy importantes ya que permiten agilizar los procesos al resolver un problema concreto antes de ser codificado.

La búsqueda es una operación que sirve para localizar un elemento específico en un conjunto de datos. En programación la búsqueda es sumamente útil para múltiples aplicaciones, como:

- ***Bases de datos:*** Se pueden buscar registros con un valor específico en una base de datos.
- ***Sistemas de archivos:*** Es posible encontrar un archivo con un nombre específico.
- ***Algoritmos de inteligencia artificial:*** Se evalúan datos y documentos para encontrar resultados relevantes a preguntas específicas.

Otros ejemplos comunes podrían ser *buscar palabras clave en un documento* o *buscar la ruta más corta en un gráfico*.

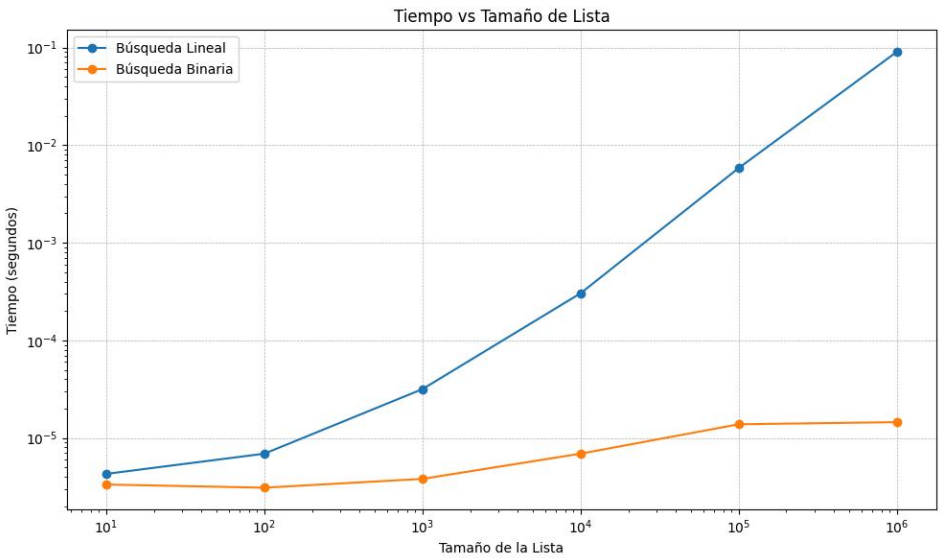
Además, hay distintos algoritmos de búsqueda, los cuales se utilizan segun las nececidades requeridas:

Búsqueda lineal	*Este algoritmo de búsqueda es el más sencillo. Recorre, de forma secuencial, cada elemento del conjunto de datos hasta llegar al objetivo.*
Búsqueda binaria	*Funciona en conjuntos de datos ordenados.*

	Bifurca el conjunto de datos y busca en la mitad correspondiente, repitiendo el proceso hasta hallar el elemento deseado.
Búsqueda de interpolación	*Este algoritmo estima la posición del elemento deseado según su valor.*
Búsqueda de hash	*Este algoritmo, permite acceder a los elementos en tiempo constante, ya que utiliza una función hash para asignar cada elemento a una ubicación única.*

*Función hash: Algoritmo que transforma una entrada de datos en una cadena de longitud fija, conocida como valor hash o resumen.

Los metodos de búsqueda más utilizados son el algoritmo de búsqueda lineal y el binario. Y un concepto fundamental que afecta al tiempo en que estos algoritmos tardan en llegar al elemento deseado, es el tamaño de la lista. Mientras más grande, más tiempo lleva encontrarlo. La búsqueda binaria reduce el tamaño del problema con cada paso, ya que como se dijo, divide en dos partes la lista. Por otro lado, la búsqueda lineal puede ser lenta si hay muchos datos, pero es más simple.



Como podemos ver, el tiempo de la búsqueda lineal aumenta más rápido que el tiempo de la búsqueda binaria a medida que aumenta el tamaño de la lista, por eso es mucho más eficiente para listas grandes.

La complejidad de los algoritmos dice cuánto tarda un algoritmo en efectuarse segun el tamaño de los datos que procesa. Se utiliza la notación *Big O (O(n))* para expresarlo.

- **O(n)** significa que si duplicas los datos, duplicas el tiempo.
- **O(Log n)** significa que "el tiempo de búsqueda aumenta logarítmicamente con el tamaño de la lista", es decir que crece mucho más lento y es perfecto para listas grandes.
- **O(n^2), O(n^3), O(n^n)** significan que el tiempo crece rápido y se vuelven lentos con muchos datos.

El ordenamiento es una manera de organizar datos, en relación a una norma específica. Es importante ya que estructura y organiza los datos de manera eficiente, permitiendo realizar otras operaciones, como búsquedas o análisis, de manera mucho más sencilla.
Además, gracias a este tipo de algoritmo se logra una búsqueda más eficiente (Debido al uso de búsqueda binaria), un análisis de datos más fácil (Ya que están ordenados) y en general, operaciones mucho más rápidas.
Se conocen varios tipos de algoritmos de ordenamiento, tales como:

- **Ordenamiento *por burbuja (*Bubble Sort*):*** Compara cada elemento de la lista, con su siguiente. Luego intercambia los elementos si están desordenados. No es muy eficiente para listas grandes.
- **Ordenamiento *por selección (*Selection Sort*):*** Encuentra el elemento más pequeño de la lista y lo intercambia con el primer elemento. Esto se repite hasta que se ordena correctamente. Es un poco más eficiente que bubble sort, pero lento para listas grandes.
- **Ordenamiento *por insercion (*Insertion Sort*):*** Inserta cada elemento de la lista en su posición correcta, en la lista ordenada. Es eficiente para listas pequeñas.
- **Ordenamiento *rápido (Quick Sort):*** Bifurca la lista y ordena, de forma recursiva, sus partes. Es mucho más rápido que Bubble Sort.
- **Ordenamiento *por mezcla (*Merge Sort*):*** Bifurca la lista, ordena ambas partes y luego las une ordenadas. Es muy eficiente, incluso en listas grandes.

La importancia de los algoritmos de búsqueda y ordenamiento:

Aspecto	Importancia
Eficiencia	Mejoran el tiempo de ejecución, en especial si hay muchos datos.
Organización	Permite presentar la información de forma clara y estructurada.
Escalabilidad	Se adaptan a diferentes tamaños de datos, desde pocos hasta millones.
Precisión	Aseguran resultados correctos y evitan errores en la búsqueda de información.
Versatilidad	Se usan en muchos contextos.

Caso Práctico

El objetivo es simular una base de datos ficticia de usuarios en Python, donde cada usuario tiene un ID, nombre, edad y correo electrónico. A partir de esta base, se implementan y comparan distintos algoritmos de búsqueda (lineal y binaria) y de ordenamiento (por ejemplo, Bubble Sort, Quick Sort y Merge Sort). Se mide el rendimiento de cada método (tiempo de ejecución) e incluso se visualizan estos tiempos mediante gráficos, lo que permite evidenciar la eficiencia de cada técnica en función del tamaño de la base de datos. Esta implementación ayuda a comprender de manera práctica cómo optimizar la búsqueda y el manejo de datos utilizando algoritmos adecuados.

A continuacion, presentamos el codigo fuente del caso practico, organizado en modulos funcionales, con comentarios explicativos.

Generación de Base de Datos

```
def generar_base_datos(n):
    """
    Crea una lista de usuarios ficticios con ID, Nombre, Edad y Email.
```

```
    """
    import random
    return [{"ID": i, "Nombre": f"Usuario{i}", "Edad": random.randint(18, 60), "Email":
f"user{i}@correo.com"} for i in range(1, n + 1)]
```

Algoritmos de búsqueda

```
def busqueda_lineal(lista, objetivo):
    """
    Busca un usuario recorriendo la lista secuencialmente.
    """
    for registro in lista:
        if registro["ID"] == objetivo:
            return registro
    return None
def busqueda_binaria(lista, objetivo):
    """
    Busca un usuario reduciendo el espacio de búsqueda en cada iteración.
    Requiere que la lista esté previamente ordenada por ID.
    """
    inicio, fin = 0, len(lista) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if lista[medio]["ID"] == objetivo:
            return lista[medio]
        elif lista[medio]["ID"] < objetivo:
            inicio = medio + 1
        else:
            fin = medio - 1
    return None
```

Ordenamiento de la Base de Datos

```
def bubble_sort(lista):
    """
```

```
    Ordena la lista usando Bubble Sort, intercambiando elementos adyacentes.
    Método simple pero poco eficiente para grandes volúmenes de datos.
    """
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j]["ID"] > lista[j+1]["ID"]:
                lista[j], lista[j+1] = lista[j+1], lista[j]
    return lista
```

Hemos seleccionado distintos algoritmos de búsqueda y ordenamiento para comparar su rendimiento en la manipulación de datos en una base ficticia de usuarios. Elegimos búsqueda binaria porque reduce el numero de comparación en listas ordenadas, mientras que la búsqueda lineal es útil cuando los datos no están ordenados o son pocos.

En cuanto a ordenamiento, incluimos bubble sort para mostrar un metido simple, pero menos eficiente en grandes volúmenes de datos, y quick sort y merge sort, que mejoran el rendimiento en listas extensas. Separamos y estructuramos el código para que sea mas fácil su reutilización y comprensión, lo que nos permite probar cada función por separado.

Hicimos pruebas con disintos tamaños de base de datos (50, 100, 200 y 300 usuarios) para evaluar el rendimiento de los algoritmos de búsqueda y ordenamiento. La ejecución del código y los resultados obtenidos serán presentados en la grabación de pantalla, mostrando la diferencia en tiempos de ejecución y análisis de los algoritmos comparados.

Metodología Utilizada

Antes de la implementación, realizamos una investigación sobre los siguientes temas:

- Fundamentos de algoritmos de búsqueda y su eficiencia
- Algoritmos de ordenamiento

- Uso de Python para la implementación de estos algoritmos

Las fuentes principales utilizadas en la investigación incluyen:

- Documentacion oficial de Python
- Material de refencia sobre estructuras de datos y algoritmos

Se ajustaron funciones para optimizar la eficiencia y corregir posibles errores en la ejecución. Con la ayuda de librerías adicionales como random, time y matplotlib, medimos tiempos y generamos graficos.

La implementación del código y pruebas, se dividieron en conjunto con mi compañera, ajustando la lógica de los algoritmos.

Tambien estrucutramos el informe y preparamos la grabación de pantalla para demostrar los resultados obtenidos.

Resultados Obtenidos

En la ejecución del código, se compararon los tiempos de búsqueda y ordenamiento en distintos tamaños de base de datos. Se realizaron pruebas con conjuntos de datos de 50, 100, 200 y 300 usuarios, aplicando búsqueda lineal, búsqueda binaria y distintos algoritmos de ordenamiento.

Observamos que la búsqueda lineal toma mas tiempo a medida que aumenta el tamaño de la base de datos, ya que debe recorrer todos los elementos secuencialmente.

La búsqueda binaria es significativamente mas rápida en listas grandes cuando los datos están ordenados previamente, reduciendo el numero de comparaciones.

Con los algoritmos de ordenamientos observamos que bubble sort es poco eficiente en grandes volúmenes de datos debido a su complejidad O (n^2).

Quic sort y merge fueron notablemente mas rapidos, con tiempos de ejecución cercanos a O (n log n), demostrando ser mejores opciones para ordenar grandes listas.

Conclusiones

En este trabajo, se han explorado distintos algoritmos de búsqueda y ordenamiento, aplicandolos a una base de datos ficticia para evaluar su rendimiento. A través de

pruebas con volúmenes de datos crecientes, se han identificado diferencias clave en eficiencia y aplicabilidad.

La búsqueda binaria es mas eficiente que la búsqueda lineal, pero requiere una lista previamente ordenada. Este ordenamiento previo influye directamente en la rapidez de la búsqueda y procesamiento.

La elección del algoritmo depende del contexto, ya que si los datos son pequeños y desordenados, el método mas simple es viable, pero en grandes conjuntos, es clave optar por algoritmos optimizados.

Con este análisis, comprendimos la importancia de la eficiencia en el desarrollo de software y demuestra como elegir el método adecuado puede mejorar significativamente el rendimiento en procesamiento de datos.

Bibliografía

- UTN - Material de Programación I. (2025). *Apuntes de algoritmos de búsqueda y ordenamiento.*
- FreeCodeCamp. (2025). *Guía de Algoritmos de Ordenamiento con ejemplos en Python.*
- UNIR México. *Algoritmos de Búsqueda y ordenamiento.*