Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Trabajo Práctico Integrador – Programación 1

“Implementación de Algoritmos de Búsqueda y Ordenamiento en un sistema de inventario”

Alumnos

Florencia Lucía Eyo Bartl

Lucas Daniel Fredes

Docente Titular

Ariel Enferrel

Docente Tutor

Franco Gonzalez

9 de Junio de 2025

Índice:

1. Introducción

Los algoritmos de búsqueda y ordenamiento son fundamentales en la programación por su utilidad en la manipulación de datos. En este trabajo integrador de índole descriptivo y explicativo abordaremos los siguientes temas, por un lado, los algoritmos de búsqueda (búsqueda lineal y búsqueda binaria) y, por otro lado, los algoritmos de ordenamiento (bubble sort y quick sort).

Comenzando con la descripción de estas categorías, para luego detallar un caso práctico. El mismo se compone de un ensayo de un negocio de artículos con una pequeña base de datos de artículos para poder trabajar con dos funciones de ordenamiento y dos de búsqueda. Luego relataremos sobre la metodología utilizada y los resultados obtenidos. Para finalizar con una conclusión personal sobre el desarrollo de este trabajo Integrador.

2. Marco Teórico

Algoritmos de Búsqueda

Los métodos de búsqueda constituyen una operación esencial en la programación, empleada para localizar un elemento específico dentro de una colección de datos, tarea frecuente en numerosas aplicaciones, tales como bases de datos, sistemas de archivos y algoritmos de inteligencia artificial.

Existen varios algoritmos de búsqueda diferentes, pero en este trabajo integrador nos vamos a centrar en dos: búsqueda lineal y búsqueda binaria, reconociendo que uno es útil para buscar sobre pocos datos (búsqueda lineal) y el otro para buscar sobre muchos datos (búsqueda binaria).

Por otro lado, existe una forma de medir la eficiencia de un algoritmo es el tiempo que tarda el mismo en ejecutarse en función del tamaño (n) de la lista que tiene que recorrer. La valoración del rendimiento de un algoritmo es lo que técnicamente se conoce como orden de complejidad, y se expresa a través de la notación de la cota superior (Big O). Donde puede ir de:

O(1) Constante: El tiempo es fijo, no depende de n;

O(log n) Logaritmo de n: El tiempo crece logarítmicamente con el tamaño de n;

O(n) Lineal: El Tiempo crece en relación directa con n;

O(n log n) Log-Lineal: Es más rápido que cuadrático, menos que lineal.

O(n**2) Cuadrático: El tiempo crece muy rápidamente.

La búsqueda lineal se basa en inspeccionar un elemento desde el inicio de la lista. En el caso de no encontrarse, se avanza al siguiente componente y se continúa de esta manera hasta el final del conjunto. Una condición vital para su ejecución es que los datos estén organizados de forma consecutiva. Es más fácil de implementar, pero puede ser más lento para conjuntos de datos grandes. El algoritmo de búsqueda secuencial presenta un rendimiento de O(n), lo cual implica que el tiempo de ejecución es directamente proporcional a la dimensión de la lista.

La búsqueda binaria es un algoritmo más eficiente que la búsqueda lineal, sobre todo cuando se trabaja con grandes volúmenes de datos. Sin embargo, requiere para su correcto funcionamiento, que la lista sobre la cual se aplica esté previamente ordenada en forma ascendente o descendente, según el criterio de búsqueda.

En cada paso, el algoritmo compara el valor buscado con el elemento que se encuentra en la posición central del rango actual. Si coincide, la búsqueda finaliza exitosamente. Si no, se evalúa si el valor que se desea encontrar es menor o mayor que el elemento del centro. Con base en esta comparación, se descarta la mitad de los elementos que no pueden contener el valor buscado y se continúa la búsqueda en la mitad restante.

Este proceso se repite de forma sucesiva, dividiendo el rango en mitades cada vez más pequeñas, hasta que el valor se encuentra o se determina que no está presente en la lista.

Desde el punto de vista computacional, la búsqueda binaria tiene una complejidad algorítmica de O(log n), lo que significa que el tiempo de ejecución crece logarítmicamente a medida que aumenta el tamaño de la lista. En la práctica, esto implica que incluso en listas muy extensas, el número de comparaciones necesarias para encontrar un valor es relativamente bajo, lo que convierte a este algoritmo en una buena opción para búsquedas en listas grandes pero ordenadas.

Algoritmos de Ordenamiento

Los algoritmos de ordenamiento son procedimientos diseñados para reorganizar una colección de elementos de acuerdo con un determinado criterio, como el valor numérico o alfabético, en forma ascendente o descendente. El propósito principal de ordenar datos es facilitar su búsqueda, análisis y visualización, ya que un conjunto organizado permite aplicar estrategias más eficientes de acceso, como la búsqueda binaria, que no sería viable en listas desordenadas.

Ordenar una lista consiste en reposicionar los elementos o referencias de modo que respeten una lógica de clasificación, según una propiedad particular de los datos. En programación, esto es fundamental cuando se trabaja con estructuras como arreglos, listas o bases de datos.

En este trabajo práctico se aplicaron dos algoritmos de ordenamiento de diferentes características: Bubble Sort y Quick Sort. Bubble Sort puede considerarse un método sencillo y accesible, ideal para listas pequeñas, frente a un algoritmo más eficiente y sofisticado, apto para listas de mayor tamaño como lo es Quick Sort. Esta diferencia evidencia la relación entre la eficiencia del algoritmo y el contexto en que se aplica.

El Bubble Sort o "ordenamiento burbuja" es uno de los algoritmos más conocidos y sencillos de implementar. Su funcionamiento se basa en comparar pares de elementos adyacentes e intercambiarlos si están en el orden incorrecto, repitiendo este proceso en múltiples pasadas hasta que toda la lista esté ordenada.

Durante cada pasada, el valor más grande va "flotando" hacia el final de la lista, de allí su nombre. Su eficiencia es limitada: en el peor de los casos, su tiempo de ejecución crece de manera cuadrática con respecto al número de elementos (complejidad temporal $O(n^2)$), lo que lo hace inadecuado para listas largas. Aun así, es útil en situaciones donde la lista es muy pequeña o casi ordenada.

Por el contrario, el *Quick Sort* es un algoritmo muy eficiente que se basa en el principio de divide y vencerás. Su estrategia consiste en elegir un elemento como pivote, y luego dividir la lista en dos sablistas:

-Una que contiene los elementos menores al pivote.

-Otra que contiene los elementos mayores o iguales.

A continuación, el algoritmo se llama recursivamente sobre cada una de las sublistas. Este proceso de división continúa hasta que todas las sablistas tienen 0 o 1 elemento (caso base), momento en el cual se combinan hacia atrás para reconstruir la lista ordenada completa. La eficiencia del algoritmo depende de qué tan bien se elija el pivote, pero en la mayoría de los casos su rendimiento es excelente.

Quicksort tiene una complejidad promedio de O(n log n), lo que lo convierte en uno de los algoritmos de ordenamiento más rápidos en la práctica. Solo en situaciones muy desfavorables (por ejemplo, si todos los elementos ya están ordenados y se elige mal el pivote), puede degradarse a $O(n^2)$.

3. Caso Práctico

Definición del Problema

Se desarrolló un sistema que simula un inventario de productos. Permitiéndole al usuario realizar búsquedas de productos por ID o nombre, y ordenar el inventario por distintos criterios: ID, precio, nombre o stock. La base del sistema es una lista de productos (cada uno representado como un diccionario de datos) sobre la que se aplican los algoritmos mencionados de búsqueda y ordenamiento.

La situación hipotética parte de una necesidad común en sistemas comerciales: almacenar, ordenar y acceder rápidamente a información sobre productos. Para ello, se creó una lista denominada inventario, compuesta por una serie de diccionarios. Cada diccionario representa un producto individual, el cual incluye los siguientes campos:

id_producto: número único que identifica al producto (clave principal).

Nombre: descripción o denominación del producto.

precio: valor monetario unitario.

cantidad_en_stock: número de unidades disponibles en inventario.

El objetivo fue simular una funcionalidad básica que permitiera buscar un producto por su ID utilizando distintos algoritmos y ordenar la lista completa de productos según el ID.

El inventario contiene 15 productos diversos. Este conjunto de datos fue diseñado para ser lo bastante simple como para seguir el proceso paso a paso y comprenderlo en detalle.

El inventario se representó con una lista de diccionarios para aprovechar la estructura flexible de Python y facilitar el acceso a los atributos de cada producto.

Implementación de funciones

Se programaron cuatro funciones principales:

- *buscar_producto_lineal(inventario, id_buscado)*

Esta función recorre el inventario producto por producto, comparando el id_producto de cada uno con el valor buscado. Si encuentra coincidencia, devuelve el diccionario del producto. Si no lo encuentra después de revisar toda la lista, devuelve None.

Este método permite realizar búsquedas sin necesidad de ordenar previamente el inventario, pero puede ser ineficiente en listas grandes, ya que compara todos los elementos uno por uno.

- *bubble_sort_por_id(inventario)*

Implementa el algoritmo Bubble Sort para ordenar los productos por su id_producto. Mediante ciclos anidados, va comparando elementos adyacentes e intercambiándolos si están fuera de orden, realizando múltiples pasadas hasta que toda la lista esté en orden ascendente.

- *busqueda_binaria_por_id(inventario_ordenado, id_buscado)*

Esta función realiza una búsqueda binaria sobre una lista previamente ordenada. Compara el valor central con el valor buscado, y según el resultado, descarta la mitad de la lista y repite el procedimiento sobre la mitad restante. Es una técnica muy eficaz, especialmente cuando se trabaja con grandes volúmenes de datos.

Para garantizar su correcto funcionamiento, se utilizó previamente la función bubble_sort_por_id y se almacenó su resultado en una nueva variable llamada inventario_ordenado.

- *quick_sort_por_id(inventario)*

Esta función aplica el algoritmo Quick Sort, utilizando recursividad. Selecciona un producto como pivote y lo compara con el resto de los elementos. Luego divide la lista en dos sablistas: menores y mayores al pivote, y ordena recursivamente cada una de ellas.

Al llegar a listas de tamaño 0 o 1 (caso base), la recursividad se detiene, y se comienza a reconstruir la lista combinando hacia atrás cada sublista ordenada. El resultado final es una lista completamente ordenada por ID.

Ejecución y pruebas del sistema

Al final del código se implementaron varios ejemplos de uso que permiten comprobar el funcionamiento de cada algoritmo imprimiendo por consola sus resultados.

-Se realizó una búsqueda lineal sobre el inventario original para buscar un producto por ID.

-Se aplicó Bubble Sort al inventario y se imprimió la lista ordenada.

-Se ejecutó una búsqueda binaria sobre esa lista ordenada.

-Finalmente, se ordenó el inventario nuevamente usando Quick Sort.

El código permite observar cómo se comporta cada algoritmo ante una misma lista de datos. Las ventajas de la recursividad en el ordenamiento, la diferencia entre los métodos de búsqueda y el impacto del orden previo en la eficiencia.

4. Metodología Utilizada

El desarrollo del trabajo se dividió en varias etapas.:

En primer lugar, se realizó una investigación teórica sobre los algoritmos seleccionados, consultando libros, documentación académica y recursos dados por la cátedra.

Luego se diseñó un caso práctico simulando un inventario comercial con estructura de datos pensada para permitir formas de búsqueda y ordenamiento, para comparar distintos algoritmos sobre la misma base.

Se programaron los algoritmos en lenguaje Python desarrollando las funciones ya mencionadas.

Y por último se implementaron las funciones a modo de prueba con la lista de productos como ejemplo de inventario.

También se utilizaron herramientas como GitHub y Visual Studio Code como entorno de desarrollo.

5. Resultados Obtenidos

Implementando un sistema de gestión de inventario pudimos observar los resultados de la aplicación de los algoritmos y sus desempeños en la simulación.

Todos los algoritmos funcionaron correctamente según su diseño y de ellos se obtuvieron las siguientes conclusiones:

Búsqueda Lineal se pudo comprobar que este método no requiere que los datos estén ordenados (útil en listas pequeñas o con datos dispersos). Pero al recorrer toda la lista hasta encontrar el elemento buscado (o determinar su ausencia) demuestra ser ineficiente a medida que aumenta el tamaño del inventario.

El algoritmo *Bubble Sort* logró ordenar correctamente el inventario por ID, lo que permitió realizar búsquedas binarias posteriormente. Su desempeño es lento ya que realiza múltiples comparaciones e intercambios, no es una opción viable para listas extensas.

La búsqueda binaria: mostró un rendimiento superior, pero solo es posible aplicarla una vez ordenado el inventario por ID. Permitió encontrar productos específicos en menos pasos y con menor esfuerzo computacional, reduciendo considerablemente el número de comparaciones necesarias.

Quick Sort fue el algoritmo de ordenamiento que ofreció mejor rendimiento general, con una notoria reducción en la cantidad de pasos y repeticiones respecto a Bubble Sort. Cabe mencionar que incluso con listas más grandes, Quick Sort mantiene su rendimiento estable, lo que lo hace ideal para proyectos con grandes volúmenes de datos.

Las pruebas permitieron comprobar la importancia de elegir el algoritmo adecuado según el tamaño del problema.

Se comprendió que la eficiencia mejora sustancialmente cuando se aplican técnicas como Quick Sort y búsqueda binaria en conjunto para grandes volúmenes de datos a analizar.

Las salidas por consola mostraron los productos esperados, tanto al buscar como al ordenar.

6. Conclusiones

El desarrollo de este trabajo integrador permitió consolidar conocimientos fundamentales en programación, referidos al diseño e implementación de algoritmos de búsqueda y ordenamiento, así como a la eficiencia computacional asociada a cada uno de ellos. A lo largo del proceso, también se

fortalecieron otras competencias trabajadas durante la cursada, como la declaración y reutilización de funciones, el uso correcto de estructuras de control (bucles for, condicionales if, bucles anidados), y la aplicación concreta del concepto de recursividad, especialmente en la implementación del algoritmo Quick Sort.

Creemos que es muy útil comprender que no existe un único algoritmo “mejor”, sino que su efectividad depende del contexto de uso. Por ejemplo, cuando se trabaja con listas pequeñas o no ordenadas, los algoritmos simples como la búsqueda lineal o el Bubble Sort son suficientes y fáciles de aplicar. Sin embargo, a medida que crece el volumen de datos, se hace evidente la necesidad de utilizar algoritmos más eficientes, como la búsqueda binaria en combinación con un ordenamiento previo mediante Quick Sort.

También fue entender cómo las decisiones que tomamos como programadores pueden afectar directamente el rendimiento del software, y por ello la importancia de hacer “economía de cómputo”.

7. Bibliografía

Álvarez, D. (2014). La eficiencia de los algoritmos [Artículo]. Dialnet. (Referencia basada en el nombre del archivo). Si tienes los datos completos (autor, año, revista, etc.), puedo ayudarte a ajustarla correctamente.

Documentación de catedra. (s.f.). Búsqueda y ordenamiento en programación. [Material educativo].

Paredes, J., & Castillo, M. (s.f.). Algoritmos de búsqueda y ordenamiento [Presentación académica]. https://gc.scalahed.com/recursos/files/r161r/w24810w/Algoritmos_de_busqueda_y_ordenamiento.pdf

Universidad Don Bosco. (2019). Guía 3 – Programación IV [Guía de curso]. https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-3.pdf

Universidad San Marcos. (2020). LEC ING SIST 0020 2020 [Material académico]. https://repositorio.usam.ac.cr/xmlui/bitstream/handle/11506/2052/LEC%20ING%20SIST%200020%202020.pdf

8. Anexos

Capturas de pantalla del programa funcionando.

Repositorio en GitHub: [https://github.com/usuario/repositorio]

Video explicativo: [URL del video]

Instrucciones de uso y casos de prueba incluidos en el archivo README.