Trabajo Práctico Integrador - Programación

Algoritmos de Búsqueda y Ordenamiento

Optimización de Procesos Logísticos

Alumnos

Alex Pedro Dauria - a_p_dauria@hotmail.com

Laureano Escoca - escoca.laureano09@gmail.com

Profesor

Sebastián AUS Bruselario

Fecha de Entrega

7 de Junio de 2025

Índice

Introducción

En el campo de la programación, los algoritmos de búsqueda y ordenamiento constituyen herramientas fundamentales para la gestión eficiente de datos. Su correcta implementación permite optimizar tareas clave como la clasificación de registros, el acceso rápido a información y la mejora del rendimiento de sistemas en tiempo real. Estas capacidades resultan especialmente relevantes en áreas como la logística, donde el manejo de grandes volúmenes de datos es crítico para el almacenamiento, la distribución y el seguimiento de productos.

El presente trabajo se enfoca en el estudio y análisis comparativo de diferentes algoritmos de búsqueda y ordenamiento implementados en el lenguaje de programación Python. A través de un caso práctico que simula operaciones típicas en logística, se evaluará el desempeño de estos algoritmos sobre listas de distintos tamaños y características.

Aunque en entornos reales se suelen utilizar algoritmos optimizados como **Timsort**, en este proyecto se decidió trabajar con algoritmos clásicos (Bubble Sort, Insertion Sort, QuickSort, entre otros). Esto permite comprender sus fundamentos, identificar sus fortalezas y limitaciones, y reflexionar sobre su aplicabilidad según el contexto de uso.

Objetivo general

Analizar el comportamiento de distintos algoritmos de búsqueda y ordenamiento mediante una simulación práctica en Python, enfocada en el contexto logístico, y reflexionar sobre su impacto en la eficiencia operativa.

Marco Teórico

¿Qué son los algoritmos de búsqueda y ordenamiento?

En programación, los algoritmos de búsqueda y ordenamiento son herramientas esenciales que permiten la organización, recuperación y análisis de datos. Su uso resulta fundamental en aplicaciones que van desde sistemas de archivos y bases de datos hasta inteligencia artificial y logística avanzada.

- **Algoritmos de ordenamiento**: Reorganizan un conjunto de datos siguiendo un criterio definido (por ejemplo, de menor a mayor), mejorando la legibilidad, la eficiencia de acceso y el análisis posterior.

- **Algoritmos de búsqueda**: Permiten localizar un elemento específico dentro de una colección, con distintos niveles de eficiencia según el algoritmo aplicado y la estructura de datos involucrada.

Principales algoritmos de ordenamiento

- **Bubble Sort**: Compara elementos adyacentes e intercambia sus posiciones si están en el orden incorrecto. Es fácil de implementar pero ineficiente en listas grandes, con complejidad temporal $O(n^2)$.
- **Insertion Sort**: Inserta cada nuevo elemento en su posición correspondiente dentro de una lista ordenada parcial. Es útil en listas pequeñas o casi ordenadas.
- **Selection Sort**: Selecciona el elemento mínimo de la lista no ordenada y lo intercambia con el primero. Repite el proceso hasta ordenar completamente. No es eficiente para grandes volúmenes de datos.
- **QuickSort**: Divide la lista seleccionando un "pivote", y luego organiza recursivamente los elementos menores y mayores. Tiene buen rendimiento promedio ($O(n \log n)$), aunque en el peor caso puede degradarse a $O(n^2)$ si se eligen pivotes inadecuados.

Principales algoritmos de búsqueda

- **Búsqueda lineal**: Recorre la lista secuencialmente hasta encontrar el valor buscado. Es sencilla y no requiere orden, pero tiene rendimiento O(n).

- **Búsqueda binaria**: Requiere que la lista esté ordenada. Divide el conjunto por la mitad en cada iteración, descartando la mitad donde no puede estar el valor. Su eficiencia es O(log n).

- **Búsqueda por interpolación**: Estima la posición del valor según su magnitud. Mejora la búsqueda binaria en listas con distribución uniforme.

- **Búsqueda con hashing**: Utiliza una función hash para ubicar elementos en tiempo constante promedio (O(1)), como en diccionarios de Python.

¿Por qué son importantes?

Los **algoritmos de búsqueda y ordenamiento** son procedimientos esenciales en la programación que permiten organizar grandes volúmenes de datos y acceder a información específica de manera rápida.

En logística, esto se traduce en la capacidad de procesar listas de productos, pedidos o paquetes con rapidez, un factor crítico para la eficiencia operacional.

Clasificación de algoritmos de ordenamiento

Interno vs. externo:

- **Interno:** todo el proceso ocurre en memoria principal (ej. QuickSort).
- **Externo:** se usa cuando los datos no caben en memoria y se accede a disco (ej. MergeSort adaptado).

Estabilidad:

- Un algoritmo es **estable** si mantiene el orden relativo de los elementos con claves iguales.
- Por ejemplo, Insertion Sort y MergeSort son estables; QuickSort y Selection Sort no.

Estructuras de datos involucradas

Los algoritmos de búsqueda y ordenamiento trabajan sobre estructuras de datos lineales, principalmente listas (arrays). Sin embargo, en aplicaciones más avanzadas se utilizan:

- **Árboles binarios de búsqueda (BST):** permiten búsquedas O(log n) sin requerir orden previo.
- **Tablas hash:** permiten acceso promedio en O(1), pero no conservan orden.

Notación Big O y Análisis de Eficiencia

La notación Big O describe el crecimiento del tiempo de ejecución o uso de memoria en función del tamaño de la entrada **(n)**. Permite comparar algoritmos de manera abstracta y prever su comportamiento frente a grandes volúmenes de datos.

🔹 **¿Por qué es importante?**

- **Compara algoritmos entre sí** sin necesidad de ejecutarlos.
- Permite **anticipar problemas de escalabilidad**.
- Es clave en áreas como logística, donde se manejan grandes volúmenes de datos.

🔸 **Tipos de complejidad más comunes**

Complejidad	Nombre Común	Comportamiento a medida que crece
O(1)	Constante	Siempre toma el mismo tiempo
O(log n)	Logarítmica	Crece lentamente
O(n)	Lineal	Escala proporcional al tamaño de los datos
O(n log n)	Quasilineal	Escala bien, ideal para ordenamientos rápidos
$O(n^2)$	Cuadrática	Muy lento con entradas grandes

🔸 **Tiempo y Espacio**

La eficiencia algorítmica no se limita al tiempo de ejecución: también es fundamental considerar el uso de memoria.
En sistemas como los logísticos, donde se procesan grandes volúmenes de datos en tiempo real, **escoger el algoritmo correcto puede marcar la diferencia entre fluidez y cuello de botella**.

A continuación, se presenta una tabla comparativa con los algoritmos implementados en este proyecto, analizados tanto en su **peor caso en tiempo de ejecución** como en **uso de memoria**.

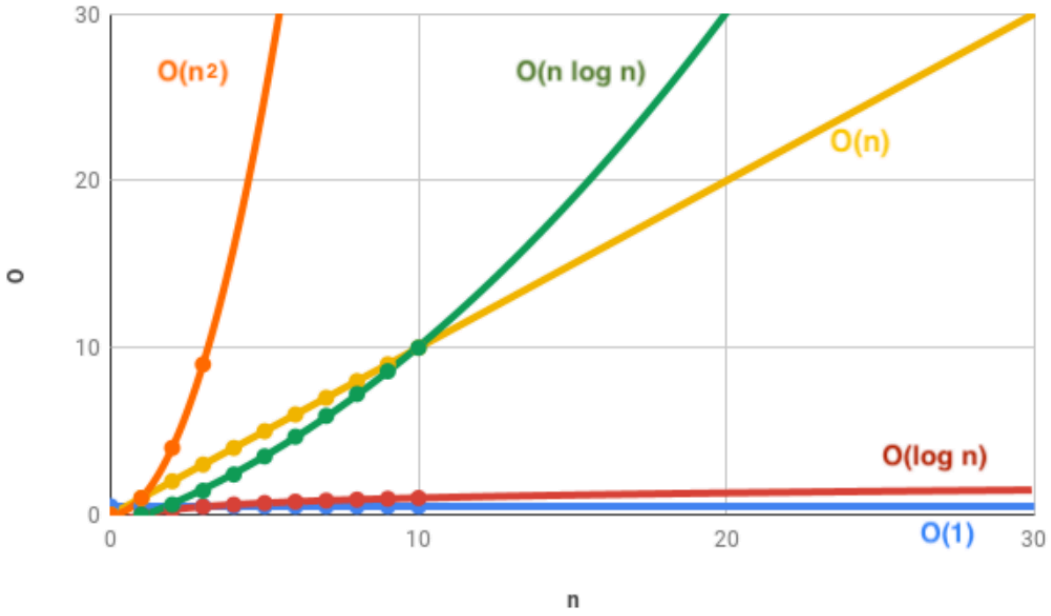
Este análisis permite anticipar qué tan escalable y eficiente será cada algoritmo según el escenario.

Algoritmo	Tiempo peor caso	Uso de memoria	Comentarios
Bubble Sort	$O(n^2)$	O(1)	Muy ineficiente en datos grandes
Insertion Sort	$O(n^2)$	O(1)	Bueno para listas chicas o casi ordenadas
Selection Sort	$O(n^2)$	O(1)	Siempre compara todo, poco eficiente
QuickSort	$O(n^2)$*	O(log n)	Rápido en promedio, cuidado con pivotes
Búsqueda Lineal	O(n)	O(1)	Sirve si la lista no está ordenada
Búsqueda Binaria	O(log n)	O(1)	Requiere lista previamente ordenada

* QuickSort puede degradarse a $O(n^2)$ si elige malos pivotes (ej: lista ya ordenada y elige extremos).

- **Visualización comparativa**

Para entender mejor cómo se comportan las distintas complejidades a medida que crece el tamaño de entrada, el siguiente gráfico ilustra su crecimiento relativo:



Como se observa, mientras que algoritmos de complejidad O(1) o O(log n) apenas crecen, aquellos con O(n²) escalan muy rápidamente, lo que los vuelve ineficientes para grandes volúmenes de datos.

Caso Práctico

Para aplicar los conceptos de búsqueda y ordenamiento en un entorno simulado, desarrollamos un programa en Python que representa un escenario logístico común: la localización de un producto dentro de un inventario. Este tipo de problema es habitual en almacenes donde es necesario identificar la posición de un artículo en un depósito, ya sea ordenado o desordenado.

El sistema parte de una lista generada aleatoriamente con 1.000 elementos que representan productos logísticos. Se selecciona uno de ellos al azar (en este caso seleccionamos el que se encuentra en la mitad) como el objetivo a buscar, simulando un pedido puntual.

El programa evalúa la eficiencia de diferentes algoritmos de búsqueda (lineal y binaria) en listas desordenadas y ordenadas, así como la performance de diversos algoritmos de ordenamiento (Bubble Sort, Insertion Sort, Selection Sort y QuickSort), midiendo el tiempo que tardan en organizar el inventario para que luego las búsquedas sean más efectivas.

El objetivo es demostrar cómo la correcta elección de algoritmos puede mejorar drásticamente los tiempos de respuesta y eficiencia operativa en contextos reales donde el volumen de datos impacta directamente sobre los resultados logísticos.

Metodología Utilizada

La metodología aplicada consistió en una simulación computacional práctica, basada en los siguientes pasos:

- **Generación de datos simulados:**
  Se creó una lista de 1.000 números aleatorios que simulan códigos de productos en un inventario.

- **Selección de objetivo:**
  Se elige como objetivo un elemento de la lista, simulando la necesidad de encontrar un producto específico en el depósito.

- **Ejecución de algoritmos de búsqueda:**
  Se aplicaron dos técnicas sobre la lista desordenada: búsqueda lineal (válida) y búsqueda binaria (inválida en este contexto), para demostrar que la búsqueda binaria requiere un inventario ordenado para funcionar correctamente.

- **Ordenamiento del inventario:**
  Se implementaron cuatro algoritmos de ordenamiento (Bubble Sort, Insertion Sort, Selection Sort y QuickSort) para organizar la lista y luego permitir búsquedas eficientes. Se midió el tiempo de ejecución de cada uno.

- **Nueva búsqueda sobre lista ordenada:**
  Se repitió la búsqueda lineal y binaria sobre la lista ordenada con QuickSort, mostrando cómo cambia el rendimiento y la validez de los resultados.

- **Medición de tiempos:**

  Para todas las operaciones se utilizó `time.perf_counter()` para obtener la duración exacta en segundos, permitiendo comparar de forma cuantitativa la eficiencia de cada enfoque.

- **Presentación dinámica:**

  El programa muestra todos los pasos y resultados en consola, incluyendo tiempos, posiciones encontradas y comparaciones entre los métodos, brindando una experiencia clara y didáctica.

Resultados Obtenidos

Se realizó la simulación de búsqueda y ordenamiento varias veces para obtener tiempos promedios representativos.

Búsqueda

- La **búsqueda lineal** encontró siempre el producto, aunque con tiempos de respuesta más altos (promedio ~17-37 microsegundos en listas de 1000 productos).

- La **búsqueda binaria** sólo funciona si la lista está ordenada. Al aplicarla sobre la lista desordenada, devolvió resultados erróneos, confirmando que no es válida en ese contexto.

- En listas ordenadas, la **búsqueda binaria** fue consistentemente la más rápida (promedio ~5 microsegundos), superando ampliamente a la **búsqueda lineal**.

Ordenamiento

- **QuickSort** fue el algoritmo más eficiente, ordenando 1000 productos en promedio en ~2.6 milisegundos.

- **Insertion Sort** funcionó bien para listas pequeñas o parcialmente ordenadas, pero con tiempos mucho más altos (~48 milisegundos).

- **Bubble Sort** y **Selection Sort** fueron los menos eficientes, con tiempos que superaron los 29 y 115 milisegundos respectivamente, confirmando su poca practicidad para volúmenes grandes.

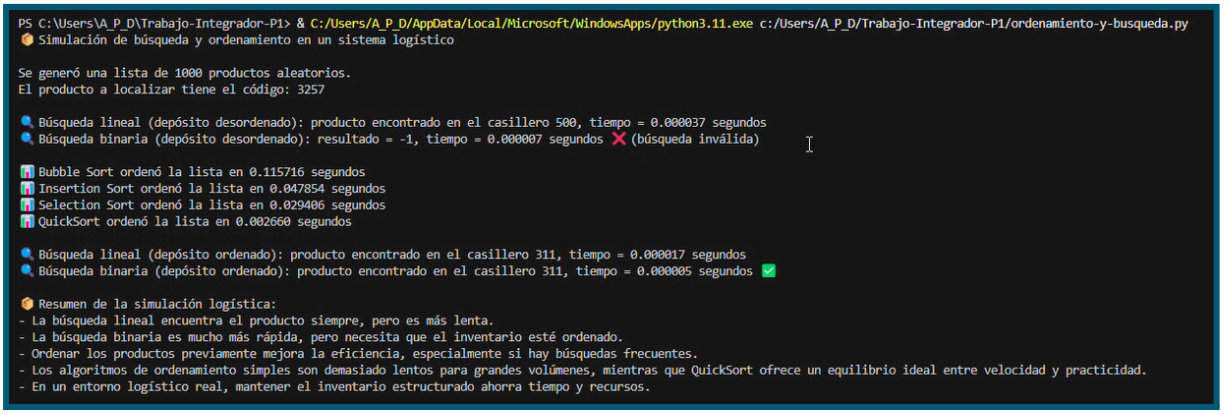
Conclusiones Prácticas

- Ordenar previamente el inventario permite aprovechar búsquedas binarias rápidas, lo que es clave en sistemas logísticos con consultas frecuentes.

- Aunque ordenar inicialmente implica un costo computacional, este se amortiza con creces cuando las búsquedas se realizan múltiples veces.

- En la simulación, **QuickSort** mostró el mejor balance entre velocidad y practicidad, siendo el más recomendado para grandes volúmenes.

Ejemplo representativo de salida que ilustra el funcionamiento de los algoritmos en un escenario logístico simulado y confirma las conclusiones anteriores:



Conclusiones

En este trabajo quedó claro que, aunque los algoritmos de búsqueda y ordenamiento son clásicos, siguen siendo fundamentales para optimizar la logística real. Implementarlos manualmente nos permitió entender su funcionamiento y, sobre todo, cuándo y por qué elegir uno u otro según el contexto.

La clave está en saber que no basta con usar funciones pre-hechas; comprender cómo y cuándo aplicar cada algoritmo marca la diferencia en eficiencia y resultados.

Además, el ejercicio reforzó nuestras habilidades técnicas y el trabajo en equipo, preparando el terreno para afrontar desafíos más complejos con bases sólidas y decisiones informadas.

Bibliografía

Universidad Tecnológica Nacional

Algoritmos de búsqueda
https://youtu.be/gJlQTq80lIg

Algoritmos de ordenamiento
https://youtu.be/xntUhrhtLaw

Búsqueda y ordenamiento en Programación

https://tup.sied.utn.edu.ar/mod/resource/view.php?id=3434

Fuentes Externas

Documentación oficial de Python

https://docs.python.org/3/library/

https://docs.python.org/3/howto/sorting.html

GeeksforGeeks: Algoritmos de Búsqueda y Ordenamiento

https://www.geeksforgeeks.org/searching-algorithms/

https://www.geeksforgeeks.org/sorting-algorithms/

FreeCodeCamp: Complejidad de Algoritmos simples y Estructuras de Datos

https://www.freecodecamp.org/espanol/news/la-complejidad-de-los-algoritmos-simples-y-las-estructuras-de-datos-en-js/

Uso de Inteligencia Artificial

Durante la elaboración de este trabajo académico, se utilizó inteligencia artificial, en particular ChatGPT de OpenAI y Grok, para asistir en la organización del contenido, redacción de explicaciones técnicas y apoyo en la estructuración del informe. Estas herramientas facilitaron la generación de textos claros, la explicación detallada de comandos y la propuesta de metodología acorde al tema tratado.

Se reconoce que el uso de IA fue un complemento y apoyo en la construcción del trabajo, pero las decisiones sobre los comandos utilizados, interpretaciones del escaneo y análisis fueron realizadas por los autores.