Trabajo Práctico - Programación I

Título: Algoritmos de Búsqueda y Ordenamiento en Python aplicados a una lista de empleados
Alumno: Juan Manuel Gomez
Materia: Programación I
Profesor/a: Ariel Enferrel

Índice

---

1. Introducción

En el camino de aprender a programar, uno de los pilares fundamentales es entender cómo buscar y ordenar información de forma eficiente. Por eso, este trabajo práctico se centra en dos herramientas clave: los algoritmos de búsqueda y de ordenamiento. Lejos de ser solo teoría, estos algoritmos están presentes en casi todos los programas que usamos a diario, desde aplicaciones que muestran listas ordenadas hasta sistemas que buscan datos en grandes volúmenes de información.

La idea principal fue aplicar estos conceptos a un caso sencillo y cercano: una lista de empleados. Se eligió este ejemplo porque resulta fácil de imaginar y permite trabajar con distintos tipos de datos (nombre, legajo,

antigüedad, etc.), como si estuviéramos diseñando un sistema básico de gestión de recursos humanos.

El objetivo fue doble: por un lado, entender cómo funcionan internamente algoritmos como Bubble Sort o la búsqueda lineal; y por otro, programar una solución que no solo sea funcional, sino también fácil de usar y expandir. A partir de eso, el proyecto creció: se incorporaron funciones para agregar, modificar y eliminar empleados, y se desarrolló un menú interactivo que guía al usuario. Todo el código fue escrito en Python, un lenguaje ideal para este tipo de prácticas por su simplicidad y legibilidad.

Este trabajo también tuvo un componente personal: me ayudó a conectar los temas vistos en clase con algo concreto, a resolver problemas reales paso a paso, y a pensar en cómo mejorar lo hecho, como si fuera parte de un proyecto más grande.

---

2. Marco Teórico

Algoritmos de Ordenamiento

Un algoritmo de ordenamiento tiene como objetivo organizar elementos de una lista según un criterio específico. Existen varios algoritmos con distintos niveles de eficiencia:

- **Bubble Sort:** Recorre la lista varias veces, comparando elementos adyacentes y los intercambia si están en el orden incorrecto. Es simple pero poco eficiente: tiene complejidad $O(n^2)$.
- **QuickSort:** Más eficiente en promedio. Utiliza el enfoque "divide y vencerás" para separar los elementos menores y mayores a un pivote, y ordena recursivamente. Su complejidad promedio es O(n log n).

Algoritmos de Búsqueda

Sirven para localizar un dato dentro de una estructura. Los más comunes son:

- **Búsqueda lineal:** Recorre toda la lista desde el principio. Es fácil de implementar, pero lenta para listas grandes. Su complejidad es O(n).

- **Búsqueda binaria:** Mucho más rápida (O(log n)), pero requiere que la lista esté previamente ordenada. Divide el rango de búsqueda a la mitad en cada paso.

---

3. Comparación de Algoritmos

A la hora de ordenar o buscar información en una lista, no todos los algoritmos rinden igual. Algunos son muy fáciles de implementar pero lentos; otros son más eficientes, pero más complejos. Por eso, antes de programar, se hizo una comparación entre distintas opciones para elegir cuáles usar en este caso.

Bubble Sort es un algoritmo de ordenamiento muy conocido por su simplicidad. Recorre la lista varias veces y va intercambiando los elementos que están fuera de lugar. Aunque es fácil de entender y perfecto para aprender, tiene una eficiencia baja ($O(n^2)$), lo que significa que se vuelve lento cuando hay muchos elementos. Aun así, fue elegido para este proyecto porque la lista es corta y el foco está en aprender.

Por otro lado, **QuickSort** es más avanzado. Usa un enfoque recursivo donde se elige un pivote y se divide la lista en partes menores y mayores, ordenando cada una por separado. Tiene mejor rendimiento (O(n log n)) y es ideal para listas grandes, pero no fue necesario usarlo en este caso.

Para buscar datos, se compararon dos algoritmos:

- **Búsqueda lineal:** revisa uno por uno hasta encontrar el valor buscado. Es muy sencilla y no requiere que la lista esté ordenada, pero puede tardar si hay muchos elementos.
- **Búsqueda binaria:** mucho más rápida (O(log n)), pero solo funciona si la lista ya está ordenada.

En este proyecto, se optó por la **búsqueda lineal** justamente por su facilidad y porque no necesitábamos que la lista esté ordenada previamente. Como el objetivo era tener algo que funcione y se entienda, fue la mejor elección.

Esta comparación permitió ver que la elección de un algoritmo no depende solo de cuál es “mejor” en papel, sino del contexto: la cantidad de datos, la facilidad para implementarlo y el objetivo del programa.

---

4. Caso Práctico

Se diseñó un programa en Python que simula la gestión de una lista de empleados. Cada empleado tiene: nombre, sector, legajo y antigüedad.

El menú del programa permite:

- Ordenar empleados por nombre, antigüedad o legajo.
- Buscar un empleado por nombre.
- Agregar un nuevo empleado.
- Modificar datos de un empleado existente.
- Eliminar un empleado.

Estas funcionalidades permiten interactuar con la lista como si fuera un sistema básico de recursos humanos, y ofrecen una práctica concreta del uso de algoritmos, estructuras de datos y condicionales.

---

5. Metodología Utilizada

- Revisión de bibliografía y sitios de confianza sobre algoritmos.
- Análisis de distintos métodos para elegir los más adecuados.
- Diseño del código en papel para definir la lógica.
- Programación en Python con pruebas sobre cada función.
- Grabación de un video explicativo con presentación visual del proyecto.

---

6. Resultados Obtenidos

- El programa funciona correctamente con todas las opciones del menú.
- Se logró ordenar empleados por distintos criterios.
- La búsqueda devuelve todos los datos del empleado consultado.
- Se pueden agregar, modificar o eliminar datos en tiempo real.
- El sistema es fácil de usar y refleja un caso real de manejo de datos.

---

7. Conclusiones

Este trabajo me permitió poner en práctica muchos de los conceptos que venimos viendo en la materia, pero de una manera más concreta y cercana. No fue solo escribir un código por cumplir, sino tratar de pensar cómo se puede aplicar la programación en una situación real, como la gestión de una lista de empleados. Empezar desde cero, con una idea simple, e ir armando un programa que funcione y sea útil, fue una experiencia que me ayudó mucho a entender mejor la lógica detrás del desarrollo de software.

A nivel técnico, reforcé el uso de estructuras básicas como listas y diccionarios, y también la importancia de las funciones para dividir el problema en partes más manejables. Usar condicionales, bucles y entradas por teclado ya no fue algo aislado, sino una parte integrada del sistema. Implementar los algoritmos de ordenamiento (como Bubble Sort) y de búsqueda (como la búsqueda lineal) me obligó a prestar atención al detalle, a pensar cómo comparar datos, cómo recorrer listas y cómo optimizar, aunque sea de forma simple, el rendimiento del código.

Una parte que me gustó fue poder comparar los distintos algoritmos y no solo usarlos porque sí. Esta reflexión sobre cuándo conviene usar un algoritmo u otro, según el contexto, es algo que no se aprende solo leyendo la teoría.

Además, a medida que iba armando el programa fueron surgiendo ideas nuevas: ¿qué pasa si quiero cambiar un dato de un empleado? ¿Y si quiero eliminarlo o agregar uno nuevo? Eso me llevó a sumar más funciones y a

pensar en cómo hacer que el programa no sea estático, sino que pueda adaptarse a lo que el usuario quiera hacer. En ese proceso, sentí que dejé de hacer un ejercicio de clase y empecé a construir algo que podría ser la base de un sistema real.

Desde lo personal, fue un trabajo que me motivó mucho. Me permitió ver que, con lo que ya aprendí hasta ahora, se pueden hacer cosas útiles y con sentido. También me di cuenta de lo importante que es la organización, no solo en el código sino también en cómo se encara un proyecto: planificar, escribir, probar, corregir y mejorar.

Me quedó la sensación de que este tipo de prácticas son las que más ayudan a afianzar lo aprendido. No solo porque uno repasa conceptos, sino porque al aplicarlos a un caso concreto se entienden mejor. Y más allá del resultado, me quedé con ganas de seguir agregando cosas al programa, como guardar los datos en un archivo, trabajar con bases de datos o incluso crear una interfaz más visual. En resumen, fue una muy buena experiencia, tanto desde lo académico como desde lo personal.

---

8. Bibliografía

- Python Software Foundation. (2024). Python 3 Documentation. https://docs.python.org/3/
- Sweigart, A. (2019). *Automate the Boring Stuff with Python*. No Starch Press.
- Khan Academy. Algoritmos de búsqueda y ordenamiento. https://es.khanacademy.org/
- W3Schools. Python Lists and Dictionaries. https://www.w3schools.com/python/