Trabajo Integrador Programacion 1:

Búsqueda y Ordenamiento en Python

Materia: Programacion 1

Profesora: Cinthia Rigoni

Tutora: Ana Mutti

----

Integrantes:

Thiago Ezequiel Lagranda - [ezequielthiago885@gmail.com](mailto:ezequielthiago885@gmail.com)

Lautaro Laner - [lautilaner@gmail.com](mailto:lautilaner@gmail.com)

Comisión: 16

----

Video del proyecto: [https://www.youtube.com/watch?v=OfPOdR4ZZcE](https://www.youtube.com/watch?v=OfPOdR4ZZcE%20)

GitHub del proyecto: <https://github.com/Lautalocos/TP-Integrador-Utn-LautaroLaner-ThiagoEzequielLagranda>

**1. Introducción**

En el campo de la informática y las ciencias de la computación, los algoritmos cumplen un papel fundamental como instrucciones finitas y precisas que permiten resolver problemas de manera eficiente. Entre los más estudiados y utilizados se encuentran los algoritmos de búsqueda y ordenamiento, esenciales para la manipulación, análisis y recuperación de datos en cualquier sistema computacional.

Los algoritmos de búsqueda permiten localizar elementos dentro de estructuras de datos, como listas o arreglos, ya sea mediante técnicas simples como la búsqueda lineal o estrategias más eficientes como la búsqueda binaria. Por su parte, los algoritmos de ordenamiento tienen como objetivo organizar los datos en un orden específico (ascendente o descendente), optimizando así procesos posteriores de búsqueda, comparación o visualización. Entre los más conocidos se encuentran *Bubble Sort*, *Insertion Sort*, *Merge Sort* y *Quick Sort*, cada uno con características, ventajas y niveles de eficiencia distintos.

Comprender el funcionamiento, la lógica y la complejidad de estos algoritmos no solo es clave para el diseño de software eficiente, sino también para el análisis de rendimiento y la toma de decisiones en el desarrollo de aplicaciones. Este trabajo tiene como propósito presentar una investigación detallada sobre los principales algoritmos de búsqueda y ordenamiento, sus principios, diferencias, ventajas, desventajas y aplicaciones prácticas.

Siendo una herramienta tan común y esencial, tanto en la vida cotidiana (como lo es a la hora de hacer compras online, buscar mensajes en chats o emails, navegar por internet) como en el ámbito profesional, nos parecio un tema ideal para abordarlo en el trabajo integrador, debido a ser una herramienta que usamos todos los días, pero normalmente lo tomamos por hecho.

**2. Marco teórico**

**Algoritmo de búsqueda**

En cuanto a los algoritmos de búsqueda hay varios tipos de estos como pueden ser el algoritmo de búsqueda lineal, el de búsqueda binaria, búsqueda por interpolación y búsqueda por saltos, etcétera.

A continuación, vamos a dar una pequeña explicación de cada uno.

**Búsqueda Lineal:**

Este es el método más simple y directo. Consiste en recorrer cada elemento de la lista uno por uno, desde el principio hasta el final, comparando cada elemento con el valor que se desea encontrar.

* **Ventajas**: No requiere que los datos estén ordenados.
* **Desventajas**: Es ineficiente para listas grandes, ya que su tiempo de ejecución es proporcional al número de elementos (*O(n)*).
* **Cuando se usa**: Ideal para listas pequeñas o cuando no se conoce el orden de los datos.

**Búsqueda Binaria:**

Este algoritmo es mucho más eficiente que la búsqueda lineal, pero solo funciona si los datos están **ordenados**. Comienza comparando el valor buscado con el elemento del medio; si no coincide, decide si buscar en la mitad inferior o superior, dividiendo el problema a la mitad en cada paso.

* **Ventajas**: Alta eficiencia en listas grandes ordenadas (*O(log n)*).
* **Desventajas**: Solo funciona con listas ordenadas.
* **Cuando se usa**: Muy útil en bases de datos, diccionarios o cualquier estructura de datos previamente ordenada.

**Búsqueda por Interpolación:**

Este método es similar a la búsqueda binaria, pero en lugar de dividir la lista exactamente por la mitad, intenta predecir la posición del valor buscado utilizando una fórmula matemática, basada en el valor de los extremos.

* **Ventajas**: Más rápida que la búsqueda binaria cuando los datos están uniformemente distribuidos.
* **Desventajas**: Ineficiente si los datos no están uniformemente distribuidos o si no se conoce su rango.
* **Cuando se usa**: En contextos donde los datos son numéricos y están distribuidos de forma uniforme.

**Búsqueda por Saltos (Jump Search):**

Divide la lista en bloques del mismo tamaño y revisa los elementos de cada bloque hasta encontrar el intervalo correcto. Luego hace una búsqueda lineal dentro de ese bloque.

* **Ventajas**: Mejor rendimiento que la búsqueda lineal en listas ordenadas.
* **Desventajas**: No es tan eficiente como la búsqueda binaria.
* **Cuando se usa**: Alternativa viable cuando no se puede aplicar la interpolación y se desea mejorar la búsqueda lineal en listas ordenadas.

Cada uno de estos algoritmos tiene su aplicación específica, dependiendo de la estructura de los datos y los requisitos de eficiencia. Conocerlos permite seleccionar la mejor herramienta para resolver problemas de búsqueda en programación y desarrollo de software.

A continuación, vamos a mencionar algunas aplicaciones que usan los algoritmos de búsqueda:

**1. Motores de búsqueda (Google, Bing)**

**Algoritmos usados**: Búsqueda binaria, búsqueda por árboles, técnicas de búsqueda en grafos.

**Aplicación**: Google organiza miles de millones de páginas. Usa algoritmos avanzados de búsqueda (como PageRank combinado con búsquedas eficientes) para encontrar resultados relevantes al instante.

**2. Apps de libros y documentos (Kindle, Adobe Reader, Word)**

**Algoritmos usados**: Búsqueda lineal y búsqueda por saltos.

**Aplicación**: Permiten buscar palabras o frases dentro de documentos extensos.

**3. Tiendas online (Amazon, Mercado Libre)**

**Algoritmos usados**: Búsqueda binaria, búsqueda por relevancia.

**Aplicación**: Buscan productos dentro de catálogos enormes, y filtran resultados según el texto ingresado por el usuario.

**Plataformas de streaming (Spotify, YouTube, Netflix):**

**Algoritmos usados**: Búsqueda lineal optimizada, sistemas de indexación, árboles B.

**Aplicación**: Cuando escribes el nombre de una canción, artista o serie, se hace una búsqueda en una base de datos gigantesca.

**Algoritmo de ordenamiento**

Los algoritmos de ordenamiento son procedimientos que permiten organizar un conjunto de datos en un orden específico, ya sea ascendente o descendente. Su objetivo es facilitar tareas como la búsqueda, comparación o visualización de datos de manera más eficiente.

Existen diferentes tipos de algoritmos de ordenamiento, cada uno con su propio funcionamiento, ventajas y desventajas. Algunos son más simples y fáciles de implementar, mientras que otros son más rápidos y eficientes para grandes volúmenes de datos, pero más difíciles de implementar.

A continuación, daremos una breve explicación de cada uno.

**Bubble Sort (ordenamiento burbuja):**

Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Es sencillo, pero poco eficiente para listas grandes.

**Selection Sort:**

Busca el elemento más pequeño y lo coloca al inicio. Repite el proceso con el resto de la lista.

**Insertion Sort:**

Inserta cada elemento en su posición correcta dentro de una lista parcialmente ordenada.

**Merge Sort:**

Divide la lista en partes más pequeñas, las ordena y luego las combina. Es muy eficiente y tiene complejidad *O(n log n)*.

**Quick Sort:**

Elige un elemento “pivote” y ordena los elementos menores y mayores en dos partes. Es rápido y ampliamente usado.

Aunque muchas veces pasan desapercibidos, los algoritmos de ordenamiento están profundamente integrados en nuestra vida diaria. Cada vez que realizamos una búsqueda en una tienda online, revisamos nuestras canciones por nombre o vemos resultados organizados por relevancia, hay un algoritmo trabajando en segundo plano.

A continuación, daremos algunos ejemplos del uso de estos algoritmos en aplicaciones y servicios web que usamos día a día.

**Apps que usan algoritmos de ordenamiento**

Como hicimos con los algoritmos de búsqueda también lo vamos hacer con estos algoritmos, algunas apps que usan estos algoritmos y serian:

**1. Amazon, Mercado Libre, eBay (tiendas online)**

* **Aplicación del ordenamiento**: Ordenan los productos según:
  + Precio (de menor a mayor o viceversa).
  + Relevancia o popularidad.
  + Valoraciones de los usuarios.
* **Algoritmos usados**: Quick Sort, Merge Sort y combinaciones personalizadas según filtros.

**2. Google Drive, Windows Explorer, macOS Finder**

* Aplicación del ordenamiento:
  + Archivos por nombre, fecha de modificación, tamaño, tipo.
* Algoritmos usados: Generalmente Merge Sort o Insertion Sort optimizados internamente.

3. WhatsApp, Telegram, Gmail (mensajería y correo)

* Aplicación del ordenamiento:
  + Chats y correos ordenados por fecha o prioridad.
* Algoritmos usados: Algoritmos eficientes como Merge Sort o Quick Sort para ordenar listas largas de mensajes.

**5. Excel, Google Sheets**

* **Aplicación del ordenamiento**:
  + Ordenar columnas numéricamente, alfabéticamente, por fecha, etc.
* **Algoritmos usados**: Insertion Sort para listas pequeñas y Merge/Quick Sort en hojas grandes.

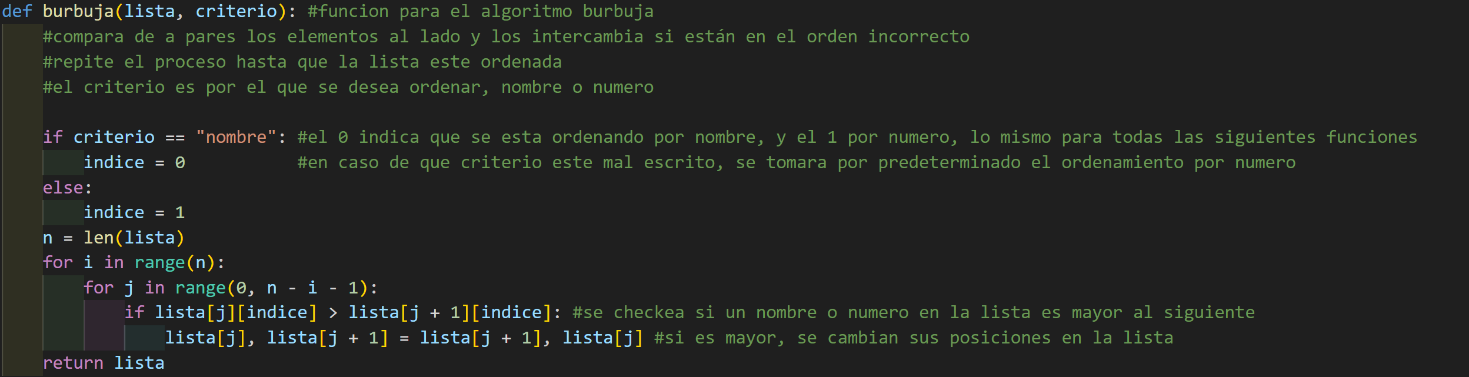
**3. Caso Practico**

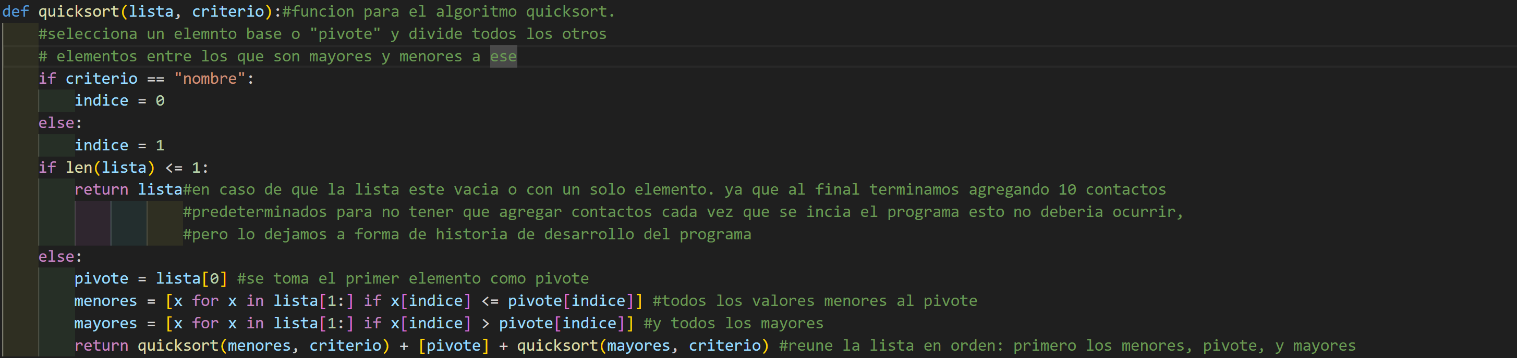
Para nuestro proyecto, decidimos crear por Python la simulación de una agenda telefónica, en la que se pueden guardar contactos con su nombre y número. Su funcionalidad consiste en:

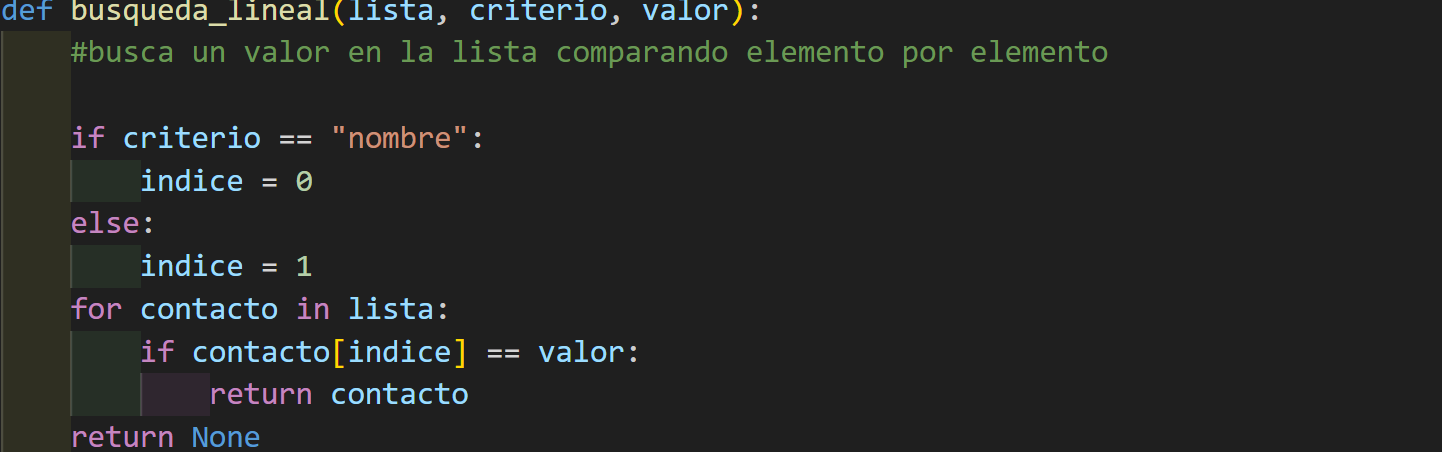
* Mostar la agenda por consola
* Crear nuevos contactos
* Ordeñar esos contactos por nombre o número, con algoritmo de burbuja o Quick Sort
* Buscar contactos por nombre o número, con algoritmo linear o binario, y si son múltiples con el mismo nombre o número mostrar todos
* Medir el tiempo que se tarda en ordenar y buscar contactos

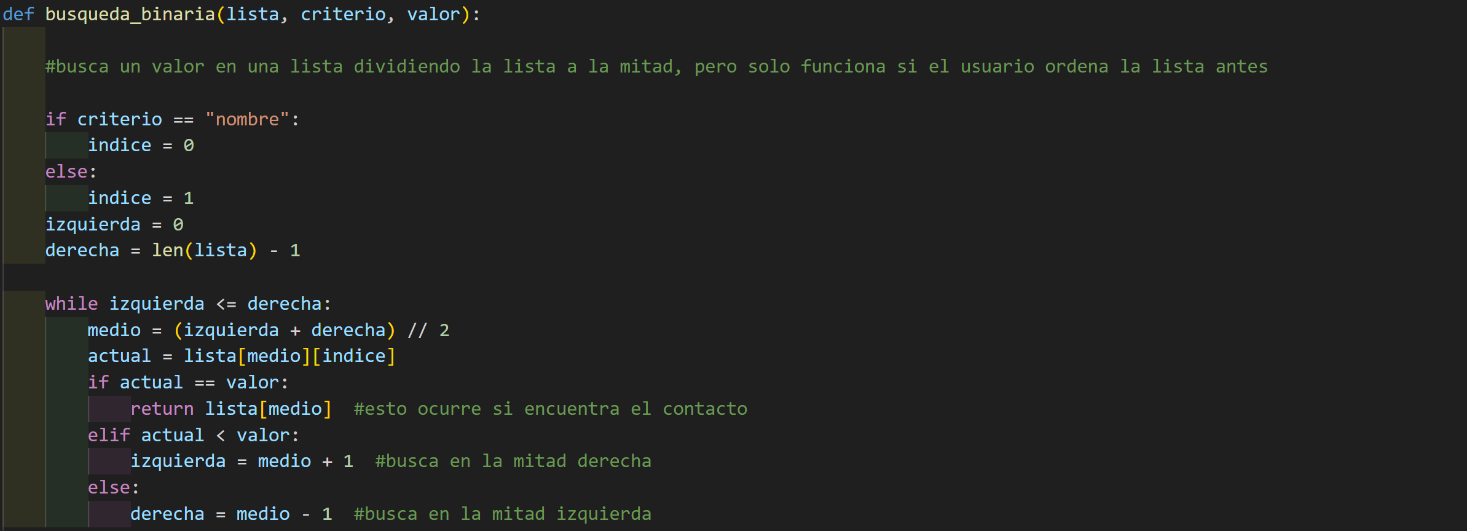
La agenda ya incluye varios contactos predeterminados para probar su funcionalidad sin necesidad de agregar contactos manualmente.

El programa esta dividido en varias funciones para cada algoritmo de ordenamiento y búsqueda, y una función principal que las llama cuando es necesario.

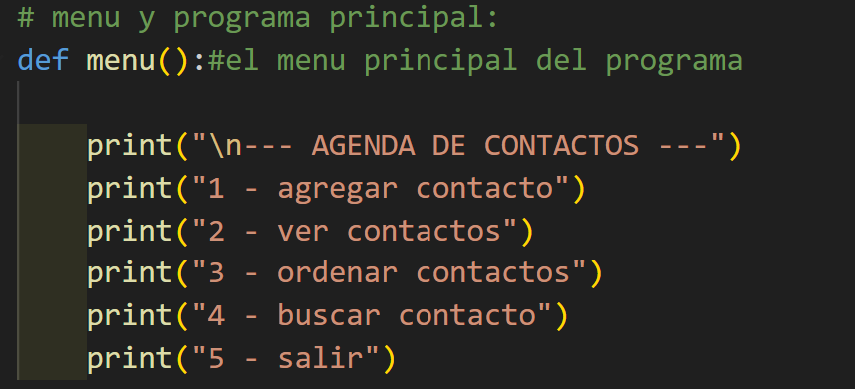


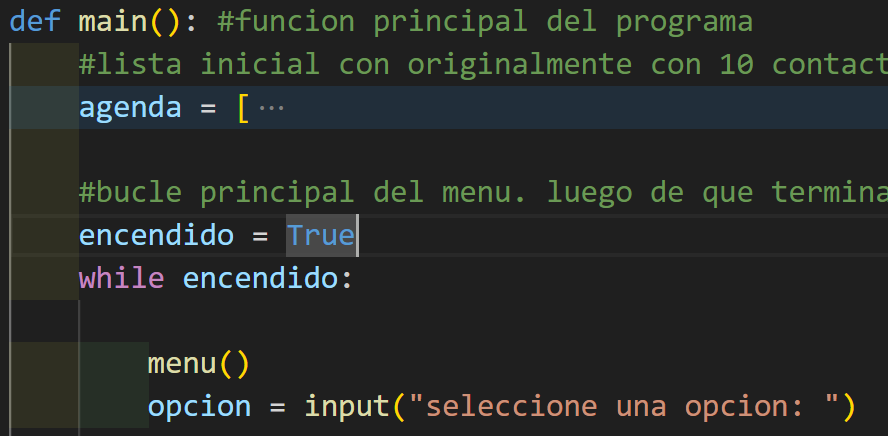


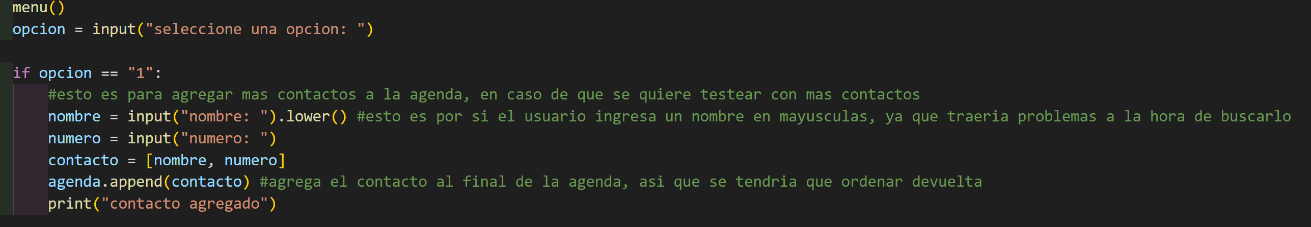


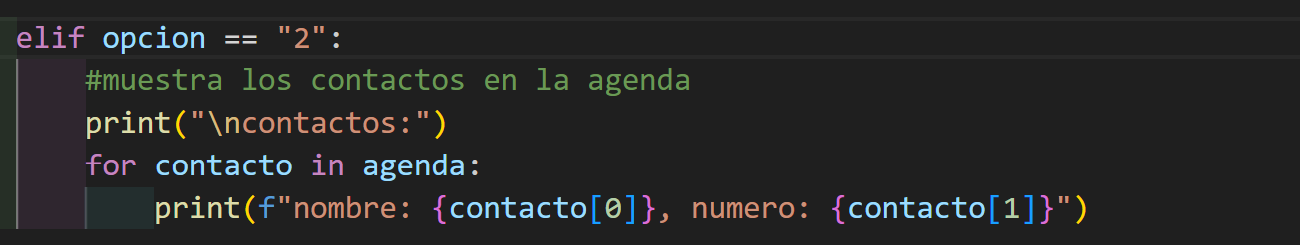


La función principal tiene una interfaz que le pide al usuario que ingrese números para acceder a las otras funciones anteriormente definidas, y luego le pide los valores que usara esa función.

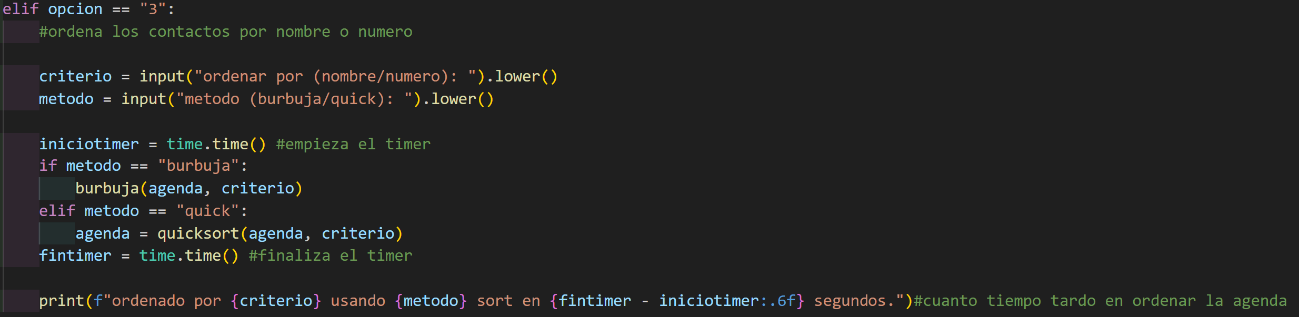


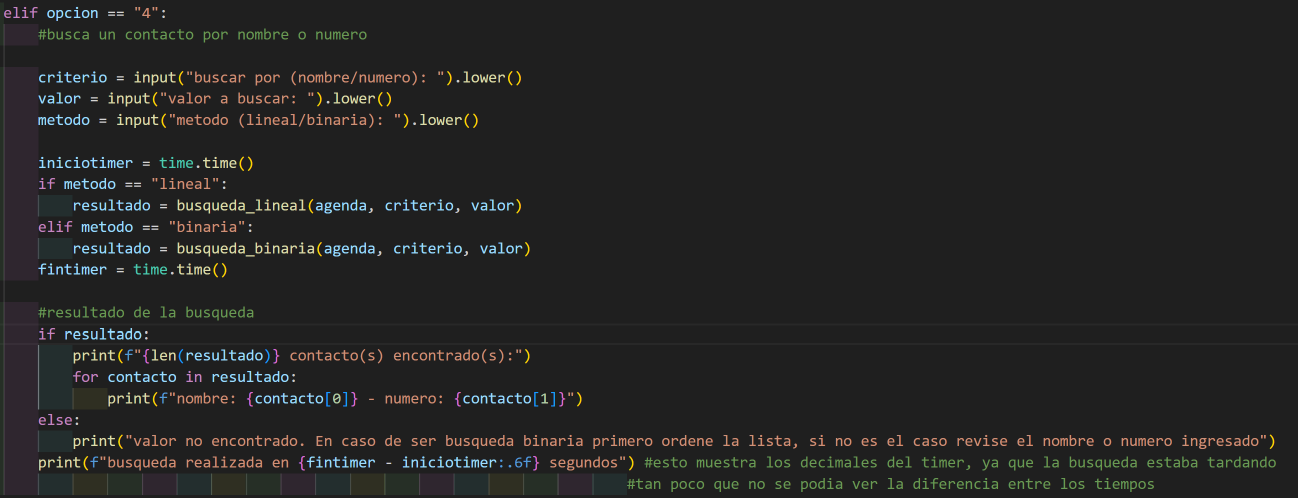


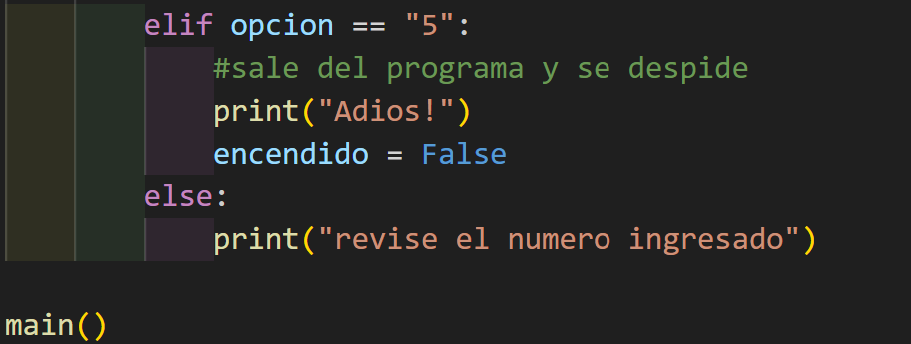




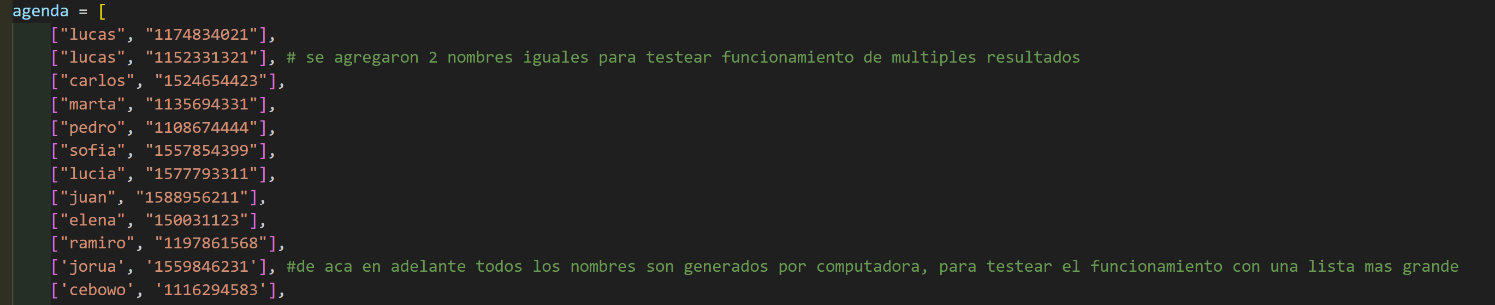
El programa importa el modulo math, ya que es necesario para calcular el tiempo de búsqueda y ordenamiento.



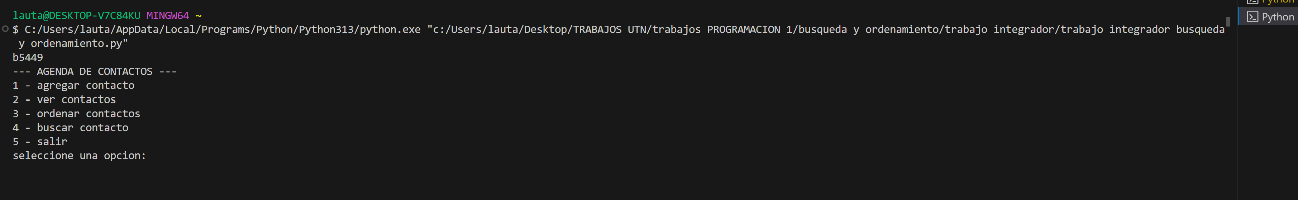


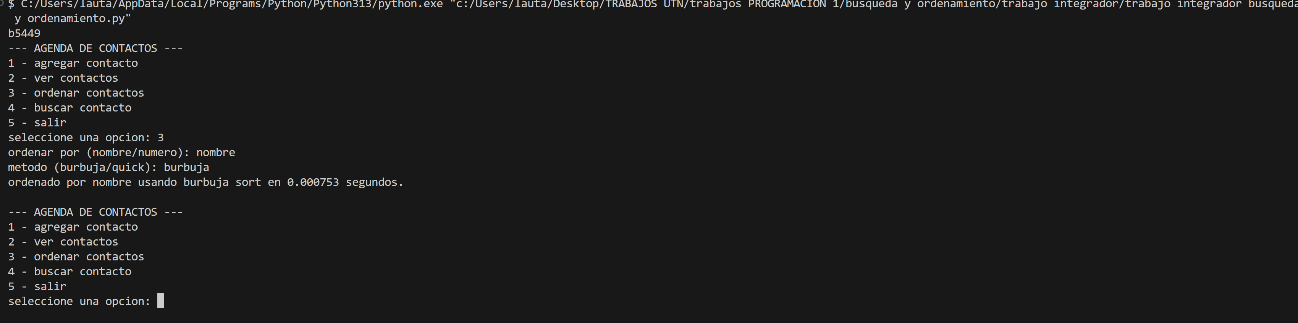


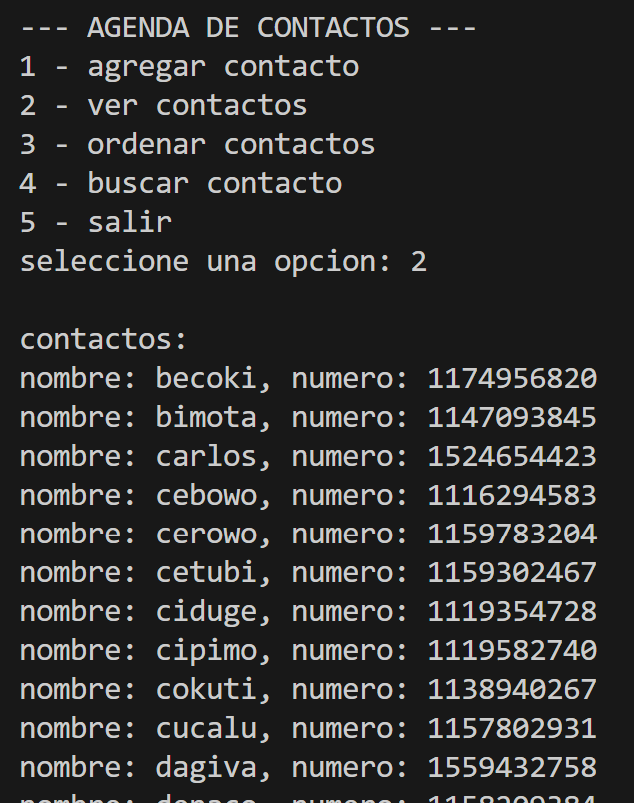
Dentro de la función principal tambien esta la agenda, que funciona como una lista anidada. Las funciones de ordenamiento y búsqueda usan la variable “índice” para acceder al campo que buscan (0 para nombre, 1 para número)

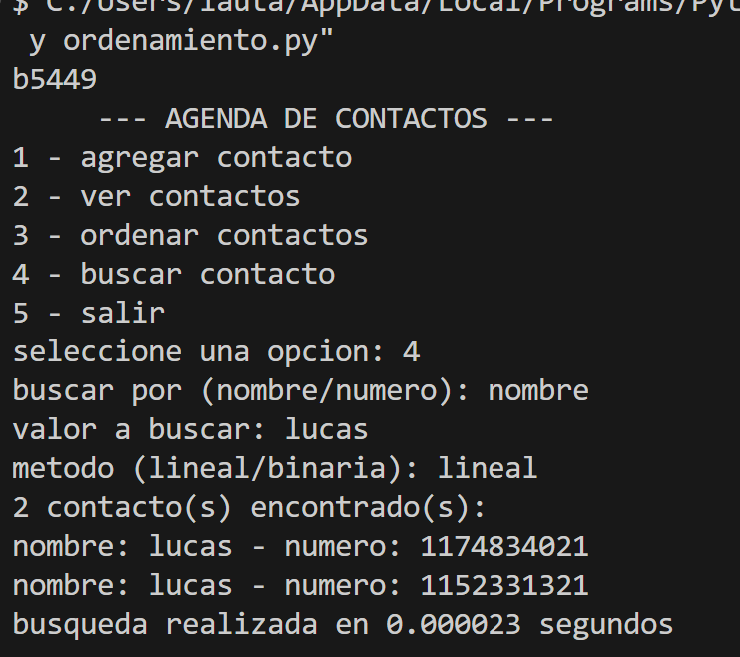


A continuación, mostramos un ejemplo del funcionamiento del programa.









**4. Metodología**

Para desarrollar el trabajo, nos basamos en etapas que nos permitieron trabajar de forma ordenada, con una división del trabajo igualada:

* Selección del tema: se eligió el tema central de búsqueda y ordenamiento por ser una herramienta del día a día que se da por hecho, y nos parecio interesante abordarla.
* Investigación: se uso principalmente el material del aula de la unidad de búsqueda y ordenamiento, pero tambien recursos online para dudas respecto al código y para investigar sobre que algoritmos usan las aplicaciones web que usamos comúnmente.
* Diseño: decidimos desde el principio armar la agenda como una lista anidada, ya que nos parecía mas facil armar el programa alrededor de ella, y la otra parte que decidimos desde el principio fue dividir todos los algoritmos en funciones separadas que son llamadas si lo pide el usuario.
* División de trabajo: luego de decidirnos en la creación de la agenda anidada, nos dividimos las funciones de ordenamiento para uno, y el otro las funciones de búsqueda. Luego de que fueran terminadas uno empezó con el texto, mientras que el otro armaba la función principal que llamaba a todas las funciones de algoritmos. Luego entre los dos miembros del grupo nos pusimos a corregir y limpiar el código para que sea más entendible.
* Desarrollo: a medida que se desarrollo el programa y se probo su funcionalidad, encontramos varios problemas y situaciones especificas que nos hicieron volver a escribir el código tomándolas en cuenta, como por ejemplo los casos donde había contactos con el mismo nombre o número. Tambien se decidió agregarle a la agenda contactos generados por computadora, ya que necesitábamos testear su funcionamiento con un número más grande de contactos que tardarían mucho siendo agregados de forma manual. Dentro del programa quedaron varios comentarios de versiones anteriores y situaciones que ya no aplican, que se dejaron a forma de recordatorio de problemas.
* Pasos finales: luego de terminar el programa y testear su funcionalidad, se procedió a subir el programa a GitHub, a completar este texto con imágenes del código funcionando, y por último a grabar el video grupal.

**5. Resultados obtenidos**

El programa pudo completar su función, siendo capaz de mostrar la agenda por consola, agregar mas contactos a ella, ser ordenada por numero y nombre por algoritmos de burbuja y Quick Sort, buscar contactos por nombre y número, por algoritmos lineares y binarios, y calcular el tiempo de ordenamiento y búsqueda.

En nuestro caso, al ser una agenda con una cantidad limitada de contactos, no se pudo apreciar mucho la diferencia de tiempo de búsqueda y ordenamiento entre diferentes algoritmos, siendo todos tan rápidos que tuvimos que usar decimales para diferenciarlos, pero si pudimos entender como funcionarían si tuviéramos que escalar el volumen de datos que se ordenan.

**6. Conclusión**

A lo largo de este trabajo, pudimos profundizar en el funcionamiento y la utilidad de los algoritmos de búsqueda y ordenamiento, herramientas fundamentales en la programación y el tratamiento de datos. Mediante el desarrollo de una agenda telefónica en Python, implementamos distintas técnicas como búsqueda lineal, binaria, y ordenamientos como Bubble Sort y Quick Sort, comprobando en la práctica cómo cada algoritmo se adapta mejor a diferentes situaciones según la estructura y cantidad de datos.

Además de reforzar los conceptos teóricos, el trabajo nos permitió mejorar nuestras habilidades de programación, organización del código, y trabajo colaborativo, especialmente a la hora de usar funciones, que nos permitió dividir el trabajo en pequeños bloques de código que nos pudimos dividir para hacer entre los dos. Para nuestros próximos proyectos, nos gustaría familiarizarnos más con el uso de GitHub, ya que no lo usamos en este proyecto para la parte colaborativa, y nos hubiera servido en este caso. Si bien, debido al tamaño reducido de la agenda, las diferencias de rendimiento entre algoritmos no fueron tan evidentes, logramos visualizar su comportamiento y escalabilidad, lo cual resulta clave para futuros proyectos más complejos.

Esta experiencia no solo consolidó nuestro conocimiento sobre algoritmos, sino que también nos demostró la importancia de elegir la herramienta adecuada según el problema a resolver, algo esencial para desarrollar soluciones eficientes.

**7. Bibliografía**

<https://www.luigisbox.es/blog/tipos-de-algoritmos-de-busqueda/>

<https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento>

<https://www.freecodecamp.org/espanol/news/guia-de-funciones-de-python-con-ejemplos/>