**PROGRAMACIÓN I**

**Trabajo Práctico: Algoritmos de Búsqueda y Ordenamiento**

****

**Alumnos:**

Rodrigo Agüero - [aguerorodrigo91@gmail.com](mailto:aguerorodrigo91@gmail.com)

Federico Villalba Petrelli - [federico.villalba.petrelli@gmail.com](mailto:federico.villalba.petrelli@gmail.com)

**Materia:** Programación I

**Profesor:** Nicolás Quinteros

**Tutor:** Flor Camila Gubiotti

**Fecha de Entrega:**  20 de Junio de 2025

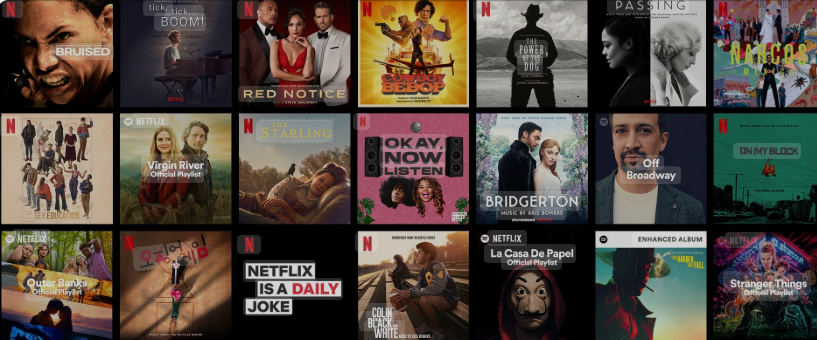
# Índice

| **Sección** |  | **Página** |
| --- | --- | --- |
| 1. Introducción |  | 3 |
| 2. Marco Teórico |  | 4 |
| 3. Caso Práctico |  | 7 |
| 3.1. Estructura del Programa |  | 8 |
| 3.2. Datos Utilizados |  | 9 |
| 3.3. Funcionalidades |  | 10 |
| 3.4. Código Fuente |  | 13 |
| 4. Metodología Utilizada |  | 22 |
| 5. Resultados Obtenidos |  | 23 |
| 6. Conclusiones |  | 27 |
| 7. Bibliografía |  | 28 |
| 8. Anexos |  | 29 |

**1. Introducción:**

En este trabajo práctico integrador de la materia Programación I, trata sobre algoritmos de búsqueda y ordenamiento usando el lenguaje Python. Estos algoritmos son muy importantes porque permiten organizar y encontrar datos de forma más rápida y eficiente.

Para hacerlo más práctico y fácil de entender, se utiliza un ejemplo basado en un pequeño catálogo de películas y series de la plataforma Netflix. De esta manera, se aplican los algoritmos en un caso realista e interesante, lo que permite visualizar mejor su utilidad.



El objetivo principal del trabajo es implementar y comparar distintos algoritmos, analizar cómo funcionan, qué tan rápidos son y en qué situaciones conviene usar cada uno. Además, se busca practicar el uso de estructuras de datos, dividir el código en partes más simples (programación modular), y escribir funciones que se puedan reutilizar.

Este trabajo permite conectar la teoría vista en las diferentes unidades con la práctica real, ayudando a entender mejor cómo se aplican estos conceptos al programar en Python.

**2. Marco Teórico:**

Los algoritmos de **Búsqueda** y **Ordenamiento** son fundamentales en programación. Se utilizan para encontrar datos específicos dentro de una colección o para organizarlos de manera que sea más fácil trabajar con ellos. Estos conceptos permiten mejorar la eficiencia y el rendimiento de los programas, sobre todo cuando se manejan grandes volúmenes de información. Se usan en muchos contextos: desde bases de datos y sistemas de archivos hasta aplicaciones web, videojuegos o plataformas de streaming, entre otros.



La **búsqueda** consiste en localizar un elemento dentro de un conjunto de datos. Hay distintos tipos de algoritmos de búsqueda, y se elige uno u otro según el tipo de datos y el contexto en el que se aplique.

Los **Tipos de Búsqueda** en los que nos centraremos en este trabajo práctico integrador son:

* **Búsqueda Lineal (Secuencial):** Recorre todos los elementos uno por uno hasta encontrar el que se busca. Es simple de entender y aplicar pero puede ser lento si la lista es muy grande.
* **Búsqueda Binaria:** Funciona en listas ordenadas, primero vamos a tener que ordenar la lista. Luego, divide la lista a la mitad y busca en la parte correspondiente. Es mucho más rápida que la búsqueda lineal, pero requiere que los datos estén previamente ordenados.



El **Ordenamiento** consiste en reorganizar los datos según algún criterio, por ejemplo, de menor a mayor, alfabéticamente, por duración o por puntaje. Esto ayuda a visualizar la información de forma más clara y a realizar búsquedas más rápidas.

Los **Tipos de Ordenamiento** en los que nos centraremos en este trabajo práctico integrador son:

* **Burbuja (Bubble Sort):** Compara elementos vecinos y los intercambia si están en el orden incorrecto. Es fácil de entender, pero muy lento para listas grandes.
* **Selección (Selection Sort):** Busca el valor más pequeño y lo coloca al principio. Es simple, pero poco eficiente.
* **Inserción (Insertion Sort):** Va colocando cada elemento en su lugar correcto dentro de una lista ordenada. Es más eficiente que los anteriores en listas pequeñas.
* **Rápido (Quick Sort):** Divide la lista y ordena las partes por separado. Es más complejo, pero muy eficiente, sobre todo para listas grandes.

| **Algoritmo** | **Ventajas** | **Desventajas** |
| --- | --- | --- |
| **Búsqueda Lineal** | Fácil de implementar, no necesita lista ordenada | Lenta en listas grandes |
| **Búsqueda Binaria** | Rápida y eficiente | Requiere lista ordenada |
| **Burbuja** | Simples de entender | Muy ineficientes en listas grandes |
| **Selection** | Fácil de implementar, no requiere memoria extra | Lenta en comparación con otros algoritmos |
| **Inserción** | Eficiente en listas pequeñas o casi ordenadas | No sirve bien para grandes cantidades de datos |
| **Rápida** | Muy rápida y eficiente | Más difícil de implementar |

**3. Caso Práctico:**

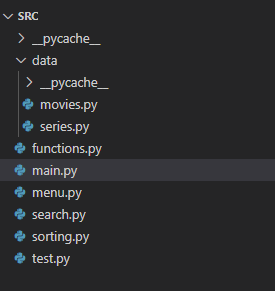
Para este trabajo práctico desarrollamos un programa en Python que permite aplicar y comparar distintos algoritmos de **búsqueda** y **ordenamiento** utilizando un conjunto de datos que simulan un catálogo de películas y series, similar al que podría tener una plataforma como Netflix pero más acotado por cuestiones prácticas. El objetivo fue crear una herramienta que permita explorar estos datos de manera interactiva, ordenarlos por título utilizando distintos métodos de ordenamiento, y buscar elementos específicos a través de diferentes técnicas de búsqueda. Esto permite observar cómo se comporta cada algoritmo en la práctica y comparar su rendimiento en situaciones concretas.



El enfoque práctico se centró en brindar una experiencia interactiva para que el usuario pueda aplicar los algoritmos vistos en clase, observar sus resultados y entender en qué casos conviene usar cada uno.

**3.1. Estructura del Programa:**

El programa está dividido en módulos para mantener el código organizado y facilitar su mantenimiento:



* **main.py:** archivo principal desde donde se ejecuta el programa.
* **menu.py:** contiene la lógica del menú interactivo.
* **search.py:** contiene las funciones de búsqueda (Secuencial y Binaria).
* **sorting.py:** contiene las funciones de ordenamiento.(burbuja, quick\_sort y otras).
* **functions.py:** contiene funciones auxiliares como mostrar el listado completo.
* **movies.py:** contiene los datos (lista de películas de Netflix).
* **series.py:** contiene los datos (lista de series de Netflix).
* **test.py:** es un archivo que utilizamos para ir realizando diferentes pruebas.

**3.2. Datos utilizados:**

Los datos están organizados como una **lista simple** de cadenas de texto (strings).

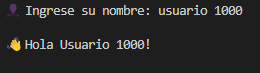
Cada elemento de la lista representa el nombre de una película. No se utiliza ninguna estructura de datos más compleja como diccionarios o listas anidadas. Este formato simplificado permite enfocarse directamente en el funcionamiento de los algoritmos de búsqueda y ordenamiento.

|  |  |
| --- | --- |

**3.3. Funcionalidades:**

El programa permite al usuario:

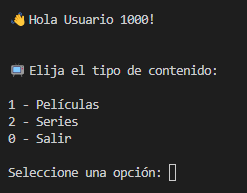
* Identificarse con su nombre.

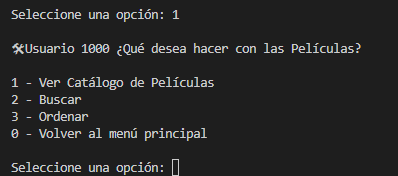


* Ver la Cantidad de Películas o Series cargadas en el Catálogo.

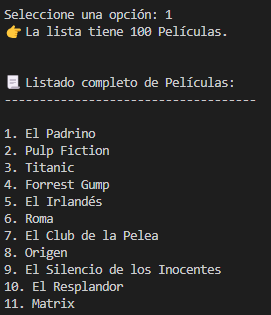


* Elegir entre diferente **Tipo de Contenido**:
* Película
* Serie

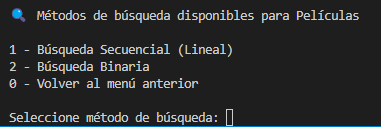


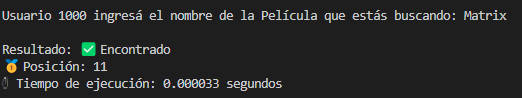


* Ver la Lista Completa de Películas o Series.

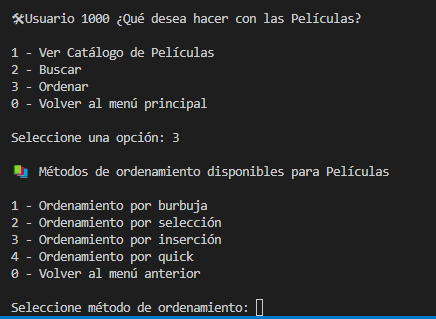


* **Buscar** una Película o Serie por su Título usando:
* Búsqueda Secuencial (Lineal)
* Búsqueda Binaria





* Ordenar la Lista por Título usando diferentes algoritmos de Ordenamiento:
* Burbuja
* Selección
* Inserción
* Rápida (quick sort)



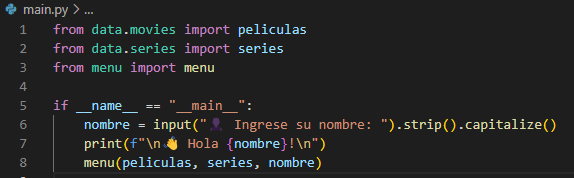
* Comparar los resultados y observar el comportamiento de los algoritmos.

| **Burbuja** | **Selección** | **Inserción** | **Quick Sort** |
| --- | --- | --- | --- |
|  |  |  |  |

**3.4. Código Fuente:**

A continuación, se muestran fragmentos representativos del código fuente con sus respectivos comentarios:

**Archivo Principal:**



El archivo **main.py** es el archivo principal desde donde se inicia la ejecución del programa.

En él se importan el menú principal, la lista de películas y la de series.

Luego, se solicita el nombre al usuario, se lo saluda y finalmente se llama al Menú pasándole como argumentos la lista de películas, la de series y el nombre del usuario.

**Menú Interactivo:**

El archivo **menu.py** contiene el menú principal del programa, que permite al usuario elegir si quiere trabajar con películas o series, y qué acción desea realizar (**Ver Catálogo**, **Buscar** u **Ordenar**).

Este archivo es clave porque organiza toda la interacción con el usuario y llama a las funciones necesarias para buscar u ordenar los datos según lo que se elija en el menú.

Incluye varias funciones importantes:

1. **menu()**
2. **solicitar\_opcion()**
3. **mostrar\_metodos\_busqueda()**
4. **mostrar\_metodos\_ordenamiento()**

Explicaremos brevemente cada función para indicar en que consiste cada una de ellas:

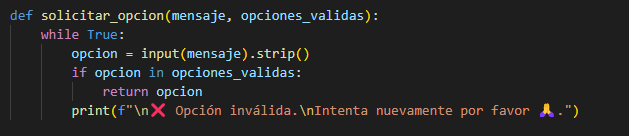
1. **menu()**

Es la función principal que muestra las opciones y controla el flujo del programa.

****

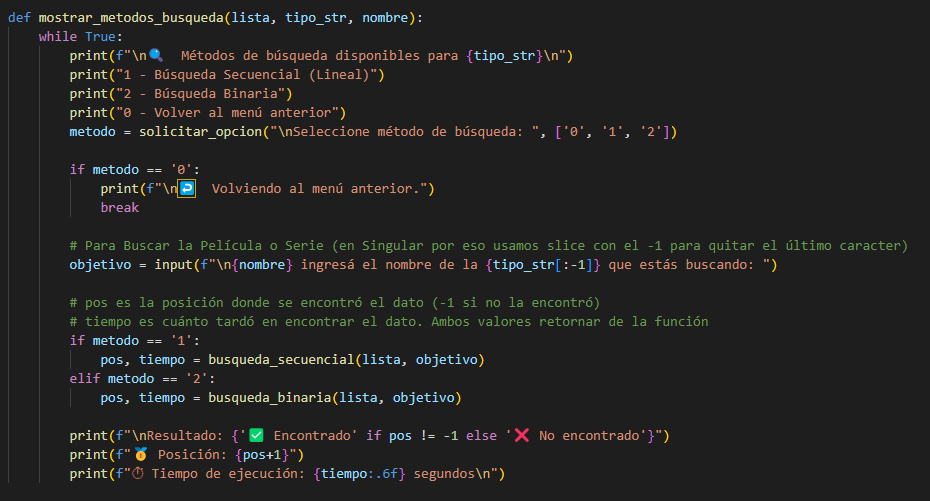
1. **solicitar\_opcion()**

Sirve para validar que las opciones ingresadas por el usuario sean correctas.

****

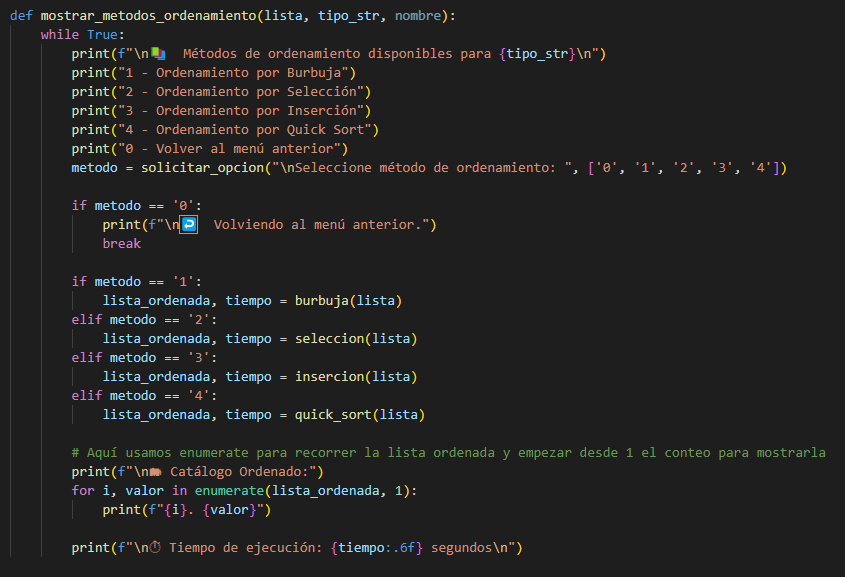
1. **mostrar\_metodos\_busqueda()**

Permite elegir entre búsqueda **secuencial** o **binaria**, pedir el nombre del contenido a buscar y mostrar el resultado junto con el tiempo que tardó.

****

1. **mostrar\_metodos\_ordenamiento()**

Permite elegir el tipo de ordenamiento (**Burbuja, Selección, Inserción o Quick Sort**), ordenar la lista y mostrar el resultado ordenado junto con el tiempo de ejecución.

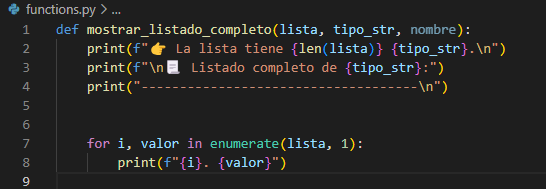
****

**Funciones Auxiliares:**

El archivo **functions.py** contiene funciones auxiliares que son utilizadas por el menú principal y otras partes del programa.

Por el momento sólo tenemos la función **mostrar\_listado\_completo()**, la cual se encarga de mostrar al usuario el Listado Completo de películas o series, mostrando cuántos elementos hay en la lista y luego cada título con su número de posición.

Esta función es útil cuando el usuario selecciona la opción de ver el catálogo completo, y se invoca desde el archivo [**menu.py**](http://menu.py).



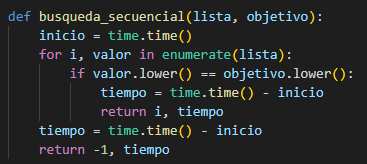
**Search:**

El archivo **search.py** contiene las funciones encargadas de realizar las búsquedas en las listas de películas o series.

* **busqueda\_secuencial() :** Implementa el algoritmo de **Búsqueda Secuencial**.

Recorre la lista buscando un elemento que coincida con el objetivo sin importar si está ordenada o no.

Luego, devuelve la posición del elemento encontrado y el tiempo que tardó en realizar la búsqueda.

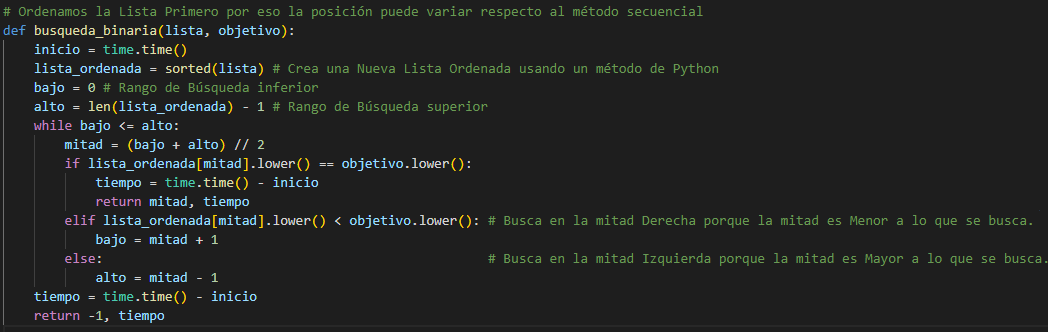


* **busqueda\_binaria():** Implementa el algoritmo de **Búsqueda Binaria**.

Primero ordena la lista y luego busca el elemento dividiendo repetidamente el rango de búsqueda por la mitad, lo que hace que sea mucho más rápido que la búsqueda secuencial.

Solo funciona si la lista está ordenada por eso primero la ordena.

También devuelve la posición del elemento encontrado y el tiempo que tardó en realizar la búsqueda.



Ambas funciones retornan el índice donde se encontró el elemento y el tiempo de ejecución de la búsqueda.

**TENER EN CUENTA:**

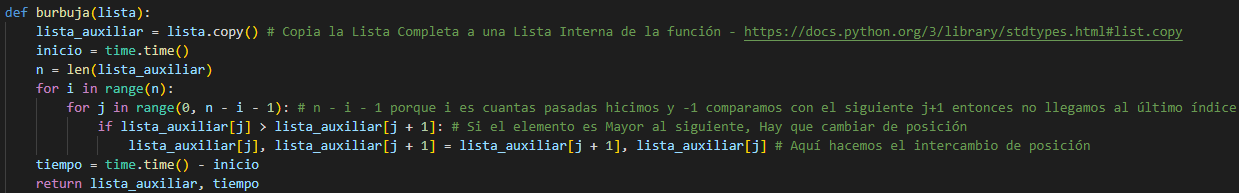
Muy probablemente el índice que devuelvan ambas funciones sea diferente, y esto se debe a que en la **Búsqueda Binaria** la lista se ordena primero. Como resultado, el título de la película o serie puede quedar en una posición diferente a la que tendría si se usara **Búsqueda Secuencial**, que no ordena la lista previamente.

**Sorting:**

El archivo **sorting.py** contiene las funciones que implementan diferentes algoritmos de ordenamiento de listas.

Los algoritmos que se implementan son los siguientes:

* **burbuja() :** Implementa el algoritmo de **Ordenamiento por Burbuja**.



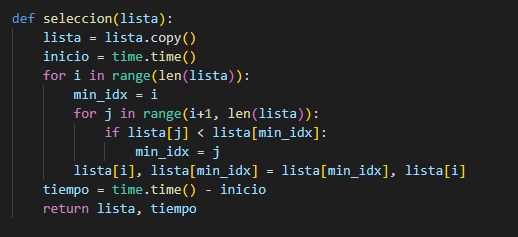
Este algoritmo compara cada elemento de la lista con el siguiente y los intercambia si están en el orden incorrecto.

Este proceso se repite varias veces hasta que la lista está ordenada.

Es un algoritmo simple, pero poco eficiente para listas grandes.

La función devuelve la lista ordenada y el tiempo que tardó en ordenarla.

* **seleccion() :** Implementa el algoritmo de **Ordenamiento por Selección**.



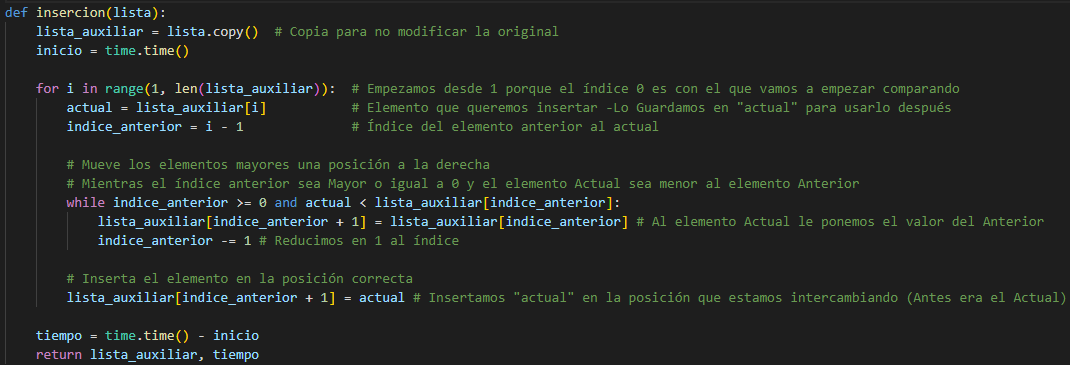
En cada paso, el algoritmo busca el valor más pequeño en la parte no ordenada de la lista y lo coloca en la posición correcta.

Este proceso se repite para cada elemento hasta que la lista queda ordenada.

Aunque es más eficiente que la burbuja, aún no es adecuado para listas grandes.

La función devuelve la lista ordenada y el tiempo de ejecución.

* **insercion() :** Implementa el algoritmo de **Ordenamiento por Inserción**.

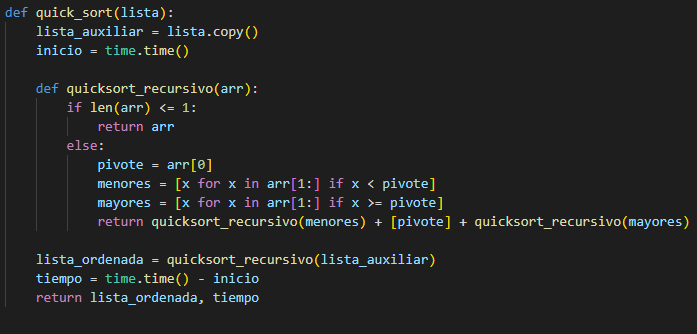


Este algoritmo toma un elemento de la lista y lo inserta en su lugar correcto dentro de la parte ordenada de la lista.

Es muy eficiente para listas pequeñas o casi ordenadas, pero no es el mejor para listas grandes.

La función también devuelve la lista ordenada y el tiempo de ejecución.

* **quick\_sort() :** Implementa el algoritmo de **Ordenamiento Rápido**.



Este es uno de los algoritmos más eficientes para listas grandes.

Se basa en el principio de dividir y conquistar, donde se elige un "pivote", y la lista se divide en dos sublistas, una con los elementos menores que el pivote y otra con los mayores.

Luego, las sublistas se ordenan recursivamente.

La función también devuelve la lista ordenada y el tiempo de ejecución.

**4. Metodología Utilizada:**

Para desarrollar este trabajo práctico se siguió una metodología basada en la combinación de investigación teórica, experimentando en un entorno controlado y desarrollando un programa funcional.



Primero, se investigaron los algoritmos de búsqueda y ordenamiento más conocidos, entendiendo cómo funcionan, cuáles son sus ventajas y desventajas, y en qué situaciones conviene usar cada uno. Esta parte teórica fue fundamental para aplicar correctamente los algoritmos más adelante y comparar sus comportamientos.

Luego, se creó un programa en Python que permite aplicar estos algoritmos sobre una lista de títulos de películas y series. El desarrollo se organizó de forma modular, separando el código en diferentes archivos según su función: uno para el menú principal, otros para las búsquedas, los métodos de ordenamiento y utilidades. Esta estructura ayudó a mantener el código más ordenado y fácil de entender.

Durante el proceso se realizaron pruebas desde la consola para verificar que los algoritmos funcionaran correctamente. Se probaron distintos casos, como ordenar la lista usando distintos métodos, o buscar películas específicas mediante búsqueda secuencial y binaria. También se agregaron validaciones y mensajes claros para guiar al usuario, por ejemplo indicando si la película fue o no encontrada.

Por último, se comentaron las partes importantes del código para explicar qué hace cada función y se capturaron imágenes del funcionamiento del programa, que se incluyen en este informe como parte de la documentación final.

**5. Resultados Obtenidos:**

Como resultado del trabajo práctico, se logró implementar un programa funcional en Python que permite aplicar y comparar diferentes algoritmos de búsqueda y ordenamiento sobre una lista de películas o series.

El programa cumplió con los objetivos planteados que eran poder mostrar cómo se comportan los algoritmos y permitir al usuario interactuar con ellos de forma sencilla.

**Funcionamiento General:**



* El menú interactivo funcionó correctamente, permitiendo al usuario elegir entre **ordenar** o **buscar** una película o una serie.
* Todos los algoritmos aplicados, tanto **Búsqueda Secuencial y Binaria** como **Ordenamientos por Burbuja, Selección, Inserción y Quick Sort,** devolvieron resultados correctos.
* La **búsqueda** mostró mensajes claros al usuario, indicando si la película o serie buscada estaba en la lista o no.
* El **ordenamiento** organizó correctamente los títulos alfabéticamente en todos los casos.

**Casos de Prueba realizados:**

****

Se probaron los siguientes escenarios:

* Búsqueda de películas o series que están al principio, en el medio y al final de la lista.
* Búsqueda de películas o series que no existían en la lista.
* Uso de Búsqueda Binaria con lista desordenada.
* Ordenamiento de la lista con cada uno de los algoritmos codeados.
* Diferentes Tamaños de listas.
* Multiplicar la lista por números grandes como 10.000
* Y varias pruebas más en el transcurso del trabajo para ir mejorando el código.

**Errores Corregidos:**

****

Durante el desarrollo se detectaron y corrigieron algunos errores, entre ellos:

* Inconsistencias en el orden del menú, que no coincidía con el orden teórico de los algoritmos. Esto fue corregido para mantener coherencia con el marco teórico.
* Se agregaron validaciones y opciones en el menú para permitir al usuario volver a pasos anteriores.
* Se incorporó una opción de personalización que solicita el nombre del usuario.
* Se añadió una opción para que el usuario pueda ver la lista completa de películas o series.
* Se añadieron mediciones de tiempo para comparar el rendimiento de los distintos algoritmos y evaluar cuál conviene en cada caso.
* Se realizaron ajustes estéticos y visuales para mejorar la experiencia de uso.
* Se modularizó el código para hacerlo más entendible y fácil de mantener.

Gracias a estas correcciones y mejoras, el programa se volvió más completo, intuitivo y coherente con los objetivos del trabajo práctico.

**Comparación de rendimiento:**

****

Aunque no se midió el tiempo de ejecución con funcionalidades específicas, se pudo observar de forma general que:

* La **Búsqueda Binaria** fue mucho más eficiente que la **Secuencial** cuando probamos con listas más grandes.
* Los algoritmos de **Ordenamientos** como **Burbuja**, **Selección** e **Inserción** tardaron más tiempo en ordenar listas más extensas.
* El algoritmo de ordenamiento **rápido (Quick Sort)** mostró resultados en tiempos considerablemente menores, especialmente con listas más largas.

Estas observaciones coinciden con lo visto en la teoría sobre la complejidad y eficiencia de cada algoritmo.

**6. Conclusiones:**

A lo largo de este trabajo práctico integrador pudimos aplicar de forma concreta los conceptos teóricos sobre algoritmos de búsqueda y ordenamiento, utilizando el lenguaje Python. Esto nos permitió no solo reforzar los contenidos vistos en las unidades de la materia Programación I, sino también experimentar con su implementación en un caso práctico concreto, como lo es un catálogo de películas o series de Netflix.

Uno de los principales aprendizajes fue entender cómo funciona cada algoritmo en la práctica, en qué situaciones conviene utilizar uno u otro, y cómo influye el tamaño de la lista en el rendimiento del programa. También pudimos ver de manera directa cómo afecta el orden de los datos a la eficiencia de ciertas búsquedas, como la binaria.

Además, trabajar de manera modular ayudó a organizar mejor el código y facilitó tanto las pruebas como la incorporación de nuevas funcionalidades.

Incorporar validaciones, mensajes claros y opciones adicionales mejoró notablemente la experiencia de usuario y permitió desarrollar una aplicación más completa.



Como conclusión general, este trabajo sirvió para conectar la teoría con la práctica, y para valorar la importancia de diseñar programas eficientes, bien estructurados y fáciles de usar. También nos demostró cómo se pueden aplicar conocimientos básicos de programación para resolver problemas reales de forma creativa y ordenada.

**7. Bibliografía:**



* Apuntes de la materia Programación I
* PDF y Videos sobre Búsqueda y Ordenamiento como Análisis de Algoritmos.
* Documentación oficial de Python. <https://docs.python.org>

**8. Anexos:**

A continuación, se incluyen materiales complementarios que ayudan a visualizar y comprender mejor el desarrollo y funcionamiento del programa:



* Capturas de Pantallas del programa funcionando, que muestran:
* Menú Principal
* Búsquedas exitosas y otras que no se encontró el resultado buscado.
* Resultados de ordenamiento utilizando diferentes algoritmos.
* Mensajes de validación y opciones adicionales del menú
* Código Fuente Completo, se encuentra adjunto en el repositorio:

<https://github.com/DesarrollosFVP/search-sort/settings/>

* Medición de Rendimiento incluída en el Punto 3. Caso Práctico.
* Cuadro Comparativo de Algoritmos presentado en el Punto 2. Marco Teórico que resume las Ventajas y Desventajas.
* Video Explicativo en repositorio.