Trabajo Integrador: Algoritmos de Búsqueda y Ordenamiento en Python

---

- **Título del trabajo:** Algoritmos de Búsqueda y Ordenamiento en Python
- **Alumnos:** Lerda Fernando – fglerda@gmail.com

  Lopez Joana – Jl105658@gmail.com

- **Materia:** Programación I
- **Profesor/a:** Cinthia Rigoni
- **Tutor/a:** Ana Mutti
- **Fecha de entrega:** 09/06/2025

---

1. Introducción

Este trabajo se centra en el estudio y aplicación de algoritmos de búsqueda y ordenamiento, herramientas fundamentales dentro del campo de la programación. Se eligió este tema por su relevancia en el manejo eficiente de datos, especialmente en sistemas que requieren acceder rápidamente a información organizada.

El objetivo principal del trabajo es analizar y aplicar algoritmos de búsqueda (particularmente la búsqueda binaria) y ordenamiento en Python, utilizando listas de diccionarios que representan estudiantes. Se pretende demostrar cómo estos algoritmos pueden ser implementados para resolver problemas concretos con eficiencia.

---

2. Marco Teórico

Algoritmos de Ordenamiento: Un algoritmo de ordenamiento organiza elementos de una lista de acuerdo a un criterio, por ejemplo, de menor a mayor. En Python, el método `sorted()` permite aplicar este tipo de algoritmos mediante funciones `key` personalizadas.

Algoritmos de Búsqueda: La búsqueda binaria es un método eficiente para encontrar elementos en listas ordenadas, reduciendo el espacio de búsqueda a la mitad en cada iteración.

Implementación en Python:

- Ordenamiento con `sorted()` y funciones `lambda`
- Búsqueda binaria utilizando índices izquierdo/derecho/medio
- Normalización de texto con `unicodedata` para tratar caracteres especiales

---

3. Caso Práctico

Descripción: Se desarrolló un programa en Python que gestiona una lista de estudiantes cargados desde un archivo CSV. El sistema permite ordenar según apellido, promedio, carrera o legajo, y realizar una búsqueda binaria según legajo, apellido o promedio.

Archivos:

- `ordenar_estudiantes.py`: funciones de ordenamiento por criterios definidos.
- `busqueda_estudiante.py`: implementación de búsqueda binaria con normalización de datos.
- `programa_unificado.py`: interfaz principal, integración de carga de datos, ordenamiento y búsqueda.

Diseño: Se eligió la búsqueda binaria por su eficiencia (órden O(log n)) frente a la búsqueda lineal. El ordenamiento previo es fundamental para su funcionamiento.

Validación: Se realizaron pruebas con criterios existentes y valores no encontrados para verificar el comportamiento del programa.

---

4. Metodología Utilizada

Etapas:

1. Investigación sobre algoritmos de ordenamiento y búsqueda.
2. Desarrollo modular del código (ordenamiento, búsqueda, integración).
3. Pruebas con datos simulados.
4. Medición de tiempos con `timeit` para evaluar rendimiento.

Herramientas:

- Python 3.12
- IDE: Visual Studio Code
- Librerías: `csv`, `unicodedata`, `math`, `timeit`
- Archivo CSV de estudiantes “Se genera un archivo con datos ficticios”

---

5. Resultados Obtenidos

- El sistema permite ordenar listas de más de 500 estudiantes de forma rápida y precisa.
- La búsqueda binaria devuelve resultados en milisegundos.
- Se detectaron errores como: conversión de tipos incorrectos, falta de normalización en apellidos con acento.
- Se corrigieron mediante el uso de `unicodedata` y conversión de datos.

- Tiempo de búsqueda promedio: < 0.001 segundos para listas ordenadas.

---

6. Conclusiones

Este trabajo permitió comprender en profundidad el funcionamiento de los algoritmos de ordenamiento y búsqueda, su importancia para mejorar la eficiencia de los programas y su aplicación real mediante Python.

Se aprendió a modularizar código, validar entradas, normalizar datos y medir rendimiento. El conocimiento adquirido es directamente aplicable a proyectos futuros que requieran procesamiento de datos.

Posibles mejoras incluyen implementar una interfaz gráfica, añadir otros métodos de ordenamiento como quicksort y comparar su rendimiento.

---

7. Bibliografía

- Python Software Foundation. (2024). *Python 3 Documentation*. https://docs.python.org/3/
- `Material de clase y apuntes del docente.`

---

8. Anexos

- Códigos fuente: `.py`
- Archivo CSV de prueba: `estudiantes.csv`