Datos Generales

- **Título del trabajo**: Algoritmos de Búsqueda y Ordenamiento en Python
- **Alumnos**: Tobias Alamedas | tobias_alamedas@hotmail.com
- **Materia**: Programación I
- **Profesor/a**: Flor Camila Gubiotti
- **Fecha de Entrega**: 04/07/2025

Índice

---

1. Introducción

El presente trabajo integrador aborda el estudio y la aplicación de algoritmos de búsqueda y ordenamiento, fundamentales en la programación. Se eligió este tema por su relevancia en la manipulación eficiente de datos

Dentro del amplio conjunto de algoritmos existentes, se trabajó con la **búsqueda lineal** y el **ordenamiento por inserción (insertion sort)**, dos algoritmos básicos pero muy útiles para el aprendizaje. El objetivo principal fue diseñar, implementar y analizar una solución que permita al usuario buscar un alumno por su nombre y mostrar sus notas ordenadas.

---

2. Marco Teórico

Búsqueda Lineal

La búsqueda lineal es un algoritmo que recorre secuencialmente una lista para encontrar un valor. Su principal ventaja es que no requiere que los datos estén ordenados.

Complejidad: O(n) en el peor de los casos.
Ventaja: Simplicidad de implementación.
Desventaja: Baja eficiencia en listas grandes.

Ordenamiento por Inserción (Insertion Sort)
El ordenamiento por inserción construye una lista ordenada insertando cada elemento en su posición correcta.
Complejidad: Mejor caso O(n), peor caso O(n^2)
Ventaja: Muy eficiente en listas pequeñas o casi ordenadas.
Desventaja: No recomendable para listas grandes no ordenadas.

3. Caso Práctico

El programa permite ingresar el nombre de un alumno y visualizar sus notas ordenadas de menor a mayor.
Se eligió búsqueda lineal porque es fácil de implementar y funciona bien con un conjunto pequeño de nombres.

Se optó por insertion sort por su buen rendimiento en listas cortas como las notas de un alumno.

---

4. Metodología Utilizada

- Investigación previa: se consultaron fuentes como la documentación oficial de Python y libros de algoritmos.
- Diseño: se definieron los algoritmos a utilizar y la estructura del programa.
- Implementación: se programó en Visual Studio Code en tres archivos separados.
- Pruebas: se realizaron pruebas con múltiples nombres y conjuntos de notas.
- Control de versiones: se utilizó GitHub para documentar y alojar el proyecto.
-

---

5. Resultados Obtenidos

- El programa funciona correctamente y cumple con los objetivos planteados.
- Se validó que la búsqueda encuentra nombres con distintas mayúsculas/minúsculas.
- Se comprobó que el algoritmo insertion sort ordena correctamente las listas de notas.
- Se subió el proyecto a GitHub:

Errores corregidos:

- La búsqueda no funcionaba con mayúsculas distintas → se resolvió agregando un .lower().
- insertion_sort no retornaba la lista → se resolvio agregando un return

6. Conclusiones

- Este trabajo permitió aplicar de forma práctica dos algoritmos fundamentales y afianzar conocimientos en programación estructurada.
- Se mejoró la capacidad de analizar problemas y dividir soluciones en módulos reutilizables.
- Además, se aprendió el uso básico de Git y la importancia de documentar y probar el código.
- Como futuras mejoras, se podrían incorporar otros algoritmos más eficientes como búsqueda binaria y quicksort, o extender el programa con un menú interactivo.

7. 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.
- W3Schools. Python Search and Sort Algorithms. https://www.w3schools.com/python
- Cormen, T. et al. (2009). Introduction to Algorithms. MIT Pres