Algoritmos de Búsqueda y Ordenamiento

en Python.

* Alumnos:

Damian Eduardo Tristant ([st\_demian@hotmail.com](mailto:st_demian@hotmail.com))

Enderson Eduardo Suarez Porras ( tu mail)

* Materia: Programación 1
* Profesor/a:
* Fecha de entrega: 20/6/2025

Indice

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía

8. Anexos

1. Introducción.

Este trabajo se centra en la aplicación de algoritmos de búsqueda y ordenamiento utilizando el lenguaje de programación Python. Se eligió el tema debido a su relevancia dentro del desarrollo de software, ya que estos algoritmos son fundamentales para la manipulación eficiente de datos. A través del desarrollo de una agenda de contactos, se busca ilustrar de forma práctica cómo se pueden implementar distintas técnicas para buscar y organizar información.

El objetivo principal es comparar el funcionamiento de distintos algoritmos, comprender sus ventajas y limitaciones, y aplicarlos en un problema cotidiano y sencillo como la gestión de contactos en dicha agenda.

2. Marco teórico

**Algoritmos de búsqueda:**

- **Búsqueda lineal**: Recorre los elementos uno a uno hasta encontrar el buscado. Facil de programar, pero más lenta. No requiere que la lista este ordenada.

Ideal para fuente de datos pequeñas o en casos donde no se pueda acceder aleatoriamente a ellos. Solo recorriéndolos uno por uno.

- **Búsqueda Binaria**: Divide la lista sucesivamente hasta encontrar el valor lo cual la hace mucho mas rápida. Una desventaja podría ser que es mas compleja de programar y siempre requiere que la lista ya este ordenada.

Ideal para grandes fuente de datos, que se pueda acceder aleatoriamente como listas o arreglos.

**Tipo de ordenamientos:**

**Bubble Sort**: Compara pares de elementos adyacentes y los intercambia si están desordenados. Sencilla pero muy lenta.

**Merge Sort:** “Divide y vencerás” Toma la lista original y la divide en dos para después ordenarlos en forma recursiva, una vez ordenados los combina para tener la lista original pero ordenada. Ideal para grandes listas de datos.

**Insertion Sort:** Construye una lista ordenada insertando cada elemento en su posición correcta. Ideal para listas pequeñas y casi ordenadas.

3. Caso Practico.

Se desarrolló una agenda de contactos simulada utilizando listas de diccionarios. Cada contacto contiene nombre, teléfono y email. Se implementaron algoritmos de ordenamiento (burbuja e inserción) y de búsqueda (lineal y binaria).