**Trabajo Integrador - Programación I**

Título del proyecto:

Algoritmos de búsqueda y ordenamiento: análisis teórico y práctico

Alumnos:

Darío Gimenez - josedariogimenez73@gmail.com

Mauro Gaspar – gaspar.mau27@gmail.com

Materia: Programación I

Profesor: Ramiro Hualpa

Fecha de Entrega: 09 de Junio de 2025

**1. Introducción**

**2. Marco Teórico**

**3. Caso Práctico**

**4. Metodología Utilizada**

**5. Resultados Obtenidos**

**6. Conclusiones**

**7. Bibliografía**

**8. Anexos**

**1. Introducción**

En el campo de la informática y el desarrollo de software, los algoritmos son herramientas fundamentales que permiten resolver problemas de forma eficiente. Dos tipos de algoritmos especialmente relevantes en la organización y manejo de datos son los algoritmos de búsqueda y los de ordenamiento. Ambos desempeñan un papel crucial en el procesamiento y recuperación de información en sistemas computacionales, bases de datos, estructuras de datos y aplicaciones del mundo real.

El objetivo de este trabajo es analizar, comparar y comprender el funcionamiento y la eficiencia de diferentes algoritmos de búsqueda y ordenamiento, considerando tanto su rendimiento en distintos casos como su aplicabilidad práctica. Este análisis se enfoca principalmente en la búsqueda lineal y binaria, así como en los algoritmos de ordenamiento clásicos como el de burbuja, selección, inserción y los más eficientes como merge-sort y quick-sort.

**2. Marco Teórico**

### **Algoritmos de búsqueda**

Los algoritmos de búsqueda tienen como objetivo localizar un elemento dentro de una estructura de datos, como una lista o un arreglo. Dos de los métodos más conocidos son:

* Búsqueda lineal (O(n)): Recorre la lista elemento por elemento hasta encontrar el valor buscado. Es simple pero poco eficiente en listas grandes.
* Búsqueda binaria (O(log n)): Solo funciona en listas ordenadas. Divide repetidamente el rango de búsqueda a la mitad hasta encontrar el valor. Es considerablemente más eficiente que la búsqueda lineal.

### **Algoritmos de ordenamiento**

Los algoritmos de ordenamiento permiten organizar datos en un orden específico, generalmente ascendente o descendente. Estos algoritmos pueden clasificarse según su eficiencia (complejidad temporal), uso de memoria y si son estables o no.

Entre los más importantes se encuentran:

* Bubble Sort (burbuja): Algoritmo simple que compara pares de elementos adyacentes e intercambia si están en el orden incorrecto.
* Selection Sort (selección): Encuentra el elemento mínimo y lo coloca en su posición final iterativamente.
* Insertion Sort (inserción): Construye la lista ordenada una a una, insertando cada elemento en su posición correcta.
* Merge Sort (combinación): Divide el arreglo en mitades y las ordena de forma recursiva. Es eficiente y estable.
* Quick Sort (rápido): Divide y conquista utilizando un elemento pivote para separar y ordenar las partes.
* Heap Sort (montón): Utiliza una estructura de datos tipo heap para seleccionar el siguiente elemento más grande (o más pequeño).
* Radix Sort y Counting Sort: Algoritmos no basados en comparaciones, eficientes en ciertos tipos de datos con rangos conocidos.

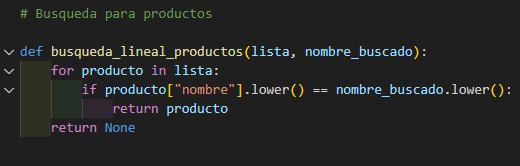
Cada algoritmo presenta ventajas y desventajas según el contexto, tamaño de los datos y recursos del sistema. Su estudio y análisis comparativo permite tomar decisiones informadas sobre qué técnica aplicar en un caso determinado.

**3. Código en Python**

Se implementó en Python una aplicación que permite ordenar una lista de productos por precio y luego buscar uno por nombre. El código se estructura en tres archivos principales:

* ordenamiento.py: contiene los algoritmos de ordenamiento.
* busqueda.py: incluye la búsqueda lineal personalizada para productos.
* main.py: ejecuta las pruebas sobre una lista predefinida.
* test\_algoritmos.py: realiza pruebas automatizadas con listas grandes.

Ejemplo de búsqueda lineal en productos:

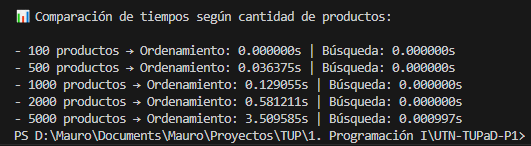


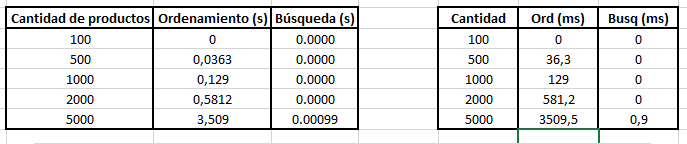
**4. Metodología Utilizada**

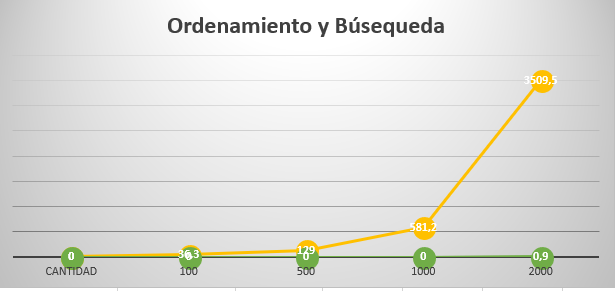
* Implementación de los algoritmos en Python.
* Ejecución del programa con una lista base de productos (main.py).
* Simulación con listas grandes para evaluación de rendimiento (test\_algoritmos.py).
* Medición del tiempo de ejecución con time.time().
* Comparación de resultados para cada combinación de ordenamiento + búsqueda.

**5. Resultados Obtenidos (ejemplo)**

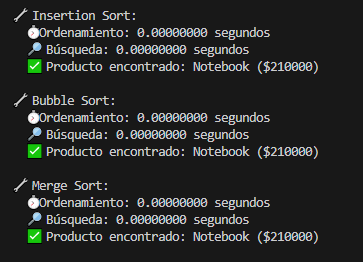
Pruebas automatizadas (test\_algoritmos.py):







Se observa crecimiento casi lineal del tiempo de búsqueda y cuadrático del tiempo de ordenamiento con *Insertion Sort*.



**6. Conclusiones**

El presente trabajo permitió profundizar en el análisis de algoritmos de ordenamiento y búsqueda aplicados a una estructura de datos común como lo es una lista de productos. A través de la implementación y comparación de los algoritmos *Bubble Sort*, *Insertion Sort* y *Merge Sort*, se evidenciaron diferencias significativas en los tiempos de ejecución, especialmente a medida que aumentaba la cantidad de elementos.

Se pudo comprobar que, si bien los algoritmos simples como *Bubble Sort* o *Insertion Sort* son adecuados para listas pequeñas por su facilidad de implementación, se vuelven ineficientes frente a volúmenes de datos mayores. Por el contrario, *Merge Sort*, aunque más complejo, ofrece un rendimiento mucho más estable y escalable gracias a su complejidad O(n log n).

Respecto a la búsqueda, se optó por una búsqueda lineal adaptada al contexto de productos representados mediante diccionarios. Aunque su complejidad es lineal, fue suficiente para el tamaño de listas manejado en este trabajo. En casos reales con grandes catálogos, sería recomendable complementar con estructuras de datos más eficientes o búsquedas más sofisticadas (como binarias o indexadas).

Este trabajo no solo reforzó conceptos fundamentales de programación y eficiencia algorítmica, sino que también promovió una mirada crítica sobre la elección del algoritmo más adecuado según el contexto, la estructura de los datos y los requerimientos de rendimiento.

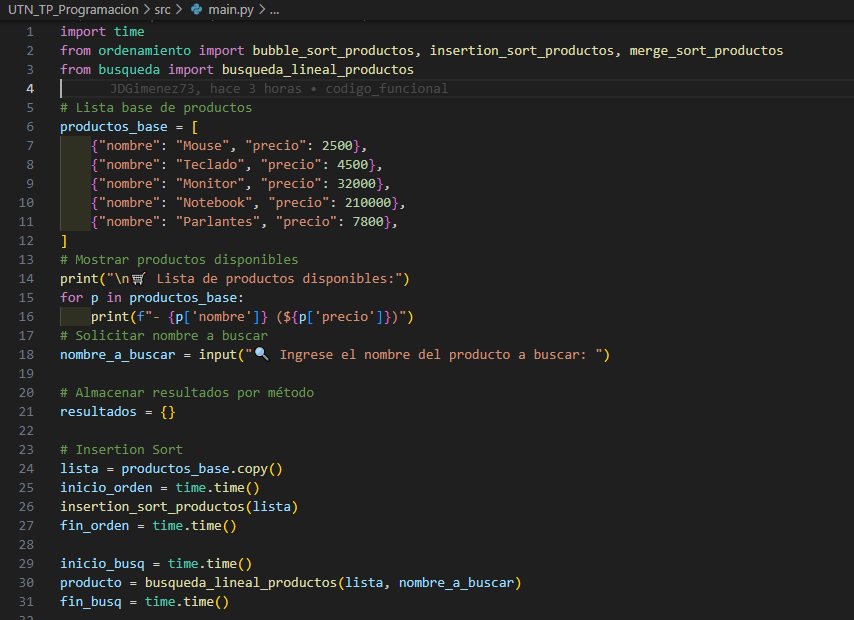
**7. Bibliografía**

[**http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap01.htm**](http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap01.htm)

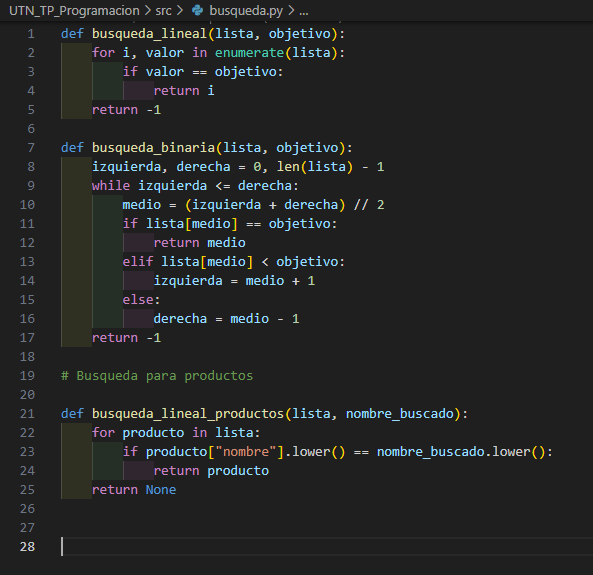
**https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/**

**8. Anexos**

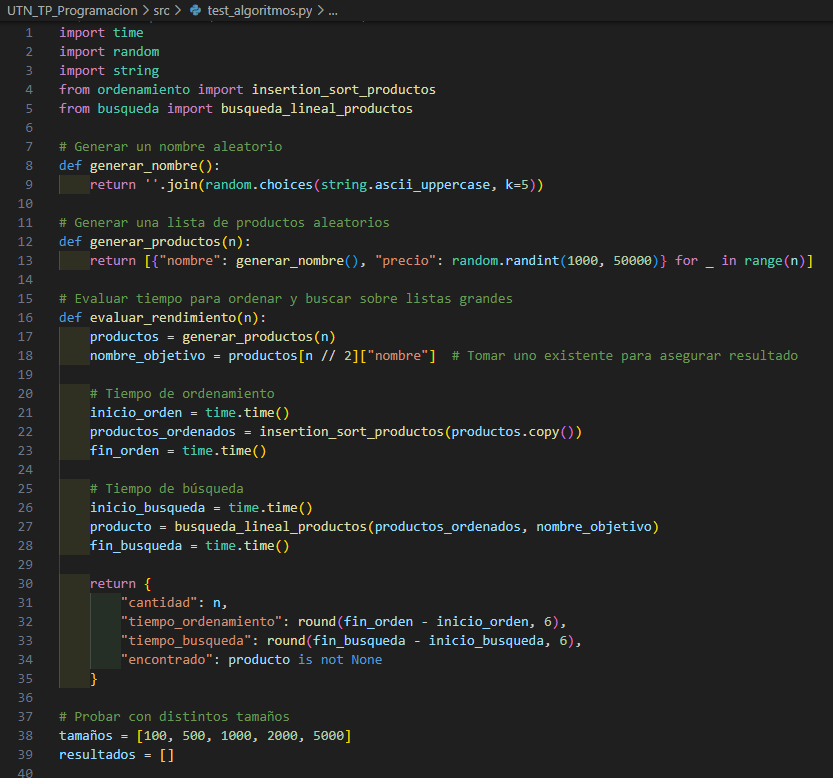
main.py:

****

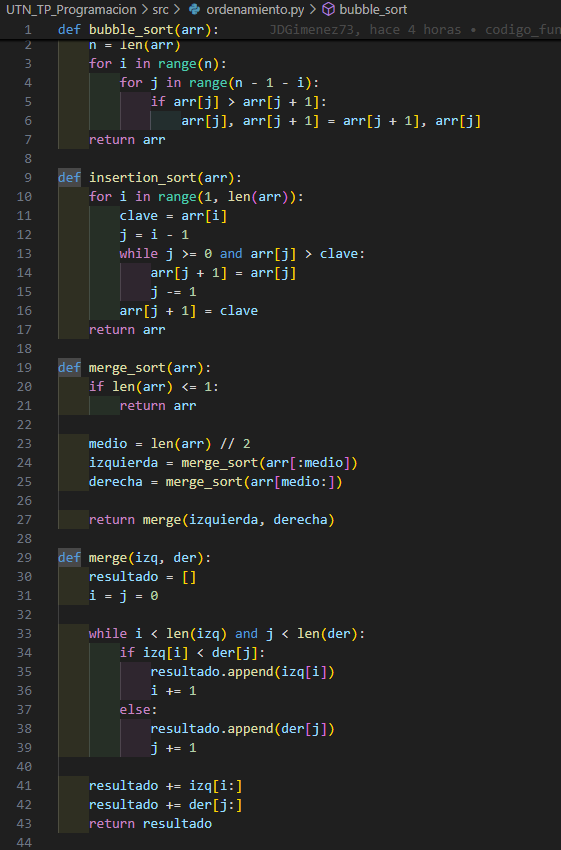
busqueda.py:

****

test\_algoritmos.py:

****

[ordenamiento.py](http://ordenamiento.py):

****

* **Video: (enlace drive)**
* **Repositorio GitHub:** https://github.com/JDGimenez73/UTN\_TP\_Programacion