Trabajo Integrador – Algoritmos de Búsqueda y Ordenamiento.

• Alumnos:

* Gabriel Torres – jose.torres@tupad.utn.edu.ar
* Daiana Velasquez – daiana.velasquez@tupad.utn.edu.ar

• Materia: Programación I

• Profesor/a: Nicolas Quirós.

* Tutor: Matias Santiago Torres.

• Fecha de Entrega: 09/06/2025

Índice

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía

8. Anexos

**Introducción**

En el ámbito de la programación y la informática, los algoritmos de búsqueda y ordenamiento representan fundamentos esenciales para el manejo eficiente de datos. Estas técnicas permiten localizar elementos dentro de estructuras de datos y organizarlos de manera que se facilite su acceso y procesamiento. Python, como uno de los lenguajes de programación más populares y versátiles en la actualidad, ofrece múltiples herramientas y bibliotecas que simplifican la implementación de estos algoritmos.

El estudio de los algoritmos de búsqueda y ordenamiento en Python no solo es relevante desde un punto de vista académico, sino también práctico, ya que estos procesos son comunes en una gran variedad de aplicaciones, desde bases de datos hasta inteligencia artificial. Entender cómo funcionan estos algoritmos y cómo se implementan en Python permite desarrollar programas más rápidos, eficientes y escalables, logrando así una optimización de los tiempos de respuesta y de recursos cuando sea posible.

En este trabajo abordaremos los conceptos fundamentales de búsqueda y ordenamiento, explorando tanto algoritmos simples como búsqueda lineal y ordenamiento por burbuja, como también técnicas más complejas como búsqueda binaria y algoritmos de ordenamiento eficientes como Quicksort, con ejemplos aplicados en Python.

**Marco teórico**

**Búsqueda**

La búsqueda es una herramienta que se utiliza en una amplia variedad de aplicaciones de programación. Es también una operación fundamental en programación que se utiliza para encontrar un elemento específico dentro de un conjunto de datos.

Es importante porque se utiliza en una amplia variedad de aplicaciones, es una tarea común en muchas de ellas, como bases de datos, sistemas de archivos y algoritmos de inteligencia artificial. Al comprender los diferentes algoritmos de búsqueda y cómo utilizarlos, se puede mejorar el rendimiento y la eficiencia de los programas.

En Python, los algoritmos de búsqueda y ordenamiento son herramientas fundamentales para manipular y gestionar datos. Los algoritmos de búsqueda permiten encontrar elementos específicos dentro de una estructura de datos, mientras que los algoritmos de ordenamiento organizan los elementos en un orden específico.

**Algoritmos de Búsqueda:**

Los algoritmos de búsqueda son fundamentales en ciencias de la computación, ya que permiten localizar uno o más elementos dentro de una estructura de datos, como listas, tuplas o diccionarios. En Python, existen diversos métodos de búsqueda, desde técnicas simples hasta algoritmos más eficientes, dependiendo del tipo y la organización de los datos.

A continuación, se describen los principales algoritmos de búsqueda:

* **Búsqueda Lineal (Sequential Search):**

Itera sobre cada elemento de la lista hasta encontrar el objetivo o llegar al final.

* **Búsqueda Binaria (Binary Search):**

Requiere que la lista esté ordenada, esto es fundamental, ya que si no está ordenada el resultado podría ser erróneo. Funciona dividiendo repetidamente la lista ordenada por la mitad, para determinar en qué segmento podría encontrarse el valor buscado.

**Algoritmos de Ordenamiento:**

Los algoritmos de ordenamiento (o "sorting algorithms") son procedimientos diseñados para reorganizar una colección de elementos (como números, cadenas o estructuras de datos) en un orden determinado, generalmente ascendente o descendente. Su aplicación es fundamental en la programación, ya que muchas operaciones posteriores —como búsquedas eficientes, estadísticas o visualización de datos— dependen de datos ordenados.

Python proporciona funciones incorporadas para ordenar listas (por ejemplo, list.sort() y sorted()) y también permite implementar algoritmos de búsqueda y ordenamiento personalizados. La elección del algoritmo depende de factores como el tamaño de la lista, si está ordenada y los requisitos de rendimiento.

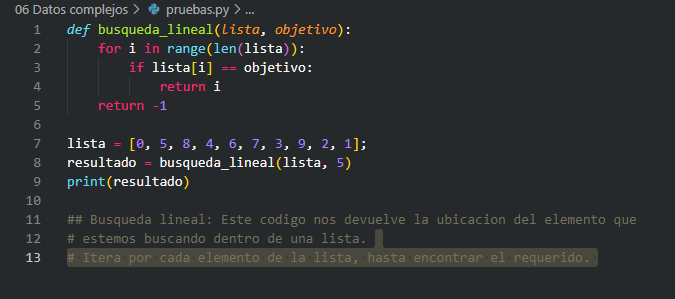
En Python, existen múltiples formas de implementar algoritmos de ordenamiento. A continuación, se describen algunos de los más representativos:

* **Ordenamiento por Inserción (Insertion Sort):** Inserta cada elemento en su posición correcta dentro de la lista ordenada hasta el momento.
* **Ordenamiento por Selección (Selection Sort):** Encuentra el elemento más pequeño (o más grande) en la lista y lo mueve a la posición correcta.
* **Ordenamiento por Burbuja (Bubble Sort):** Compara elementos adyacentes y los intercambia si están en el orden incorrecto.
* **Ordenamiento por Burbuja Mejorado (Bubble Sort Mejorado):** Consiste en el mismo algoritmo bubble sort pero con unas ligeras modificaciones para evitar que sus dos ciclos sigan recorriendo la lista cuando los elementos ya están ordenados. Esto nos permite una optimización del tiempo en algunos casos.
* **Ordenamiento Rápido (Quick Sort):** Divide la lista en sublistas y recursivamente las ordena.
* **##Ordenamiento por Montón (Heap Sort):** Construye un monton binario y extrae los elementos de forma ordenada.
* **##Ordenamiento por ShellSort:** Una mejora del ordenamiento por inserción que usa incrementos para comparar elementos no adyacentes.

**Caso Practico**

Implementación de Algoritmos de búsquedas de datos en Python:

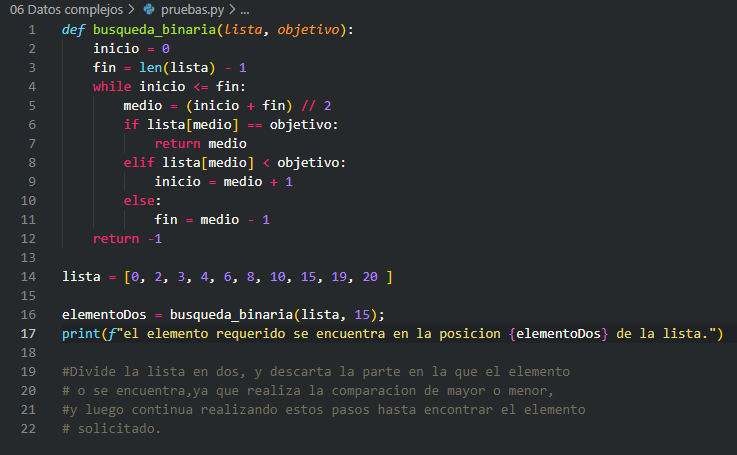
**Algoritmos de Búsqueda lineal en Python**.



Acá podemos ver como al solicitar el elemento “5”, que se encuentra en la posición “1”, la consola nos devuelve la ubicación del elemento.

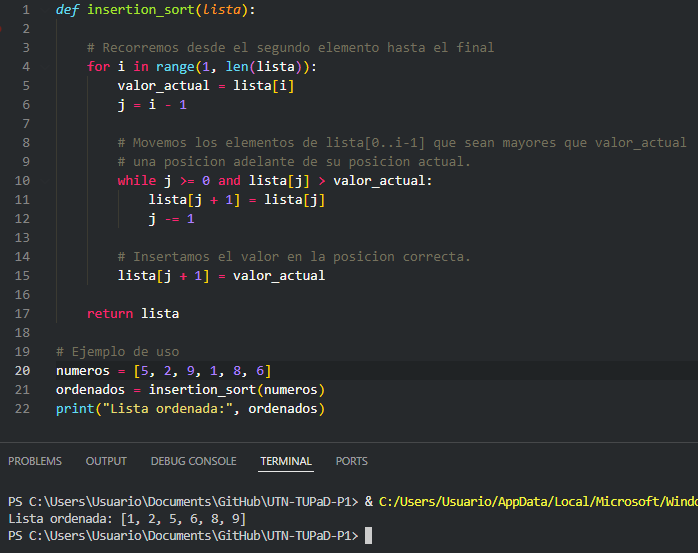
**Algoritmo de búsqueda Binaria en Python.**

Acá podemos ver como al solicitar el elemento “15”, que se encuentra en la posición “7”, la consola nos devolvería la ubicación del elemento.



**Algoritmos de Ordenamiento en Python.**

**Ordenamiento por Inserción (Insertion Sort):**



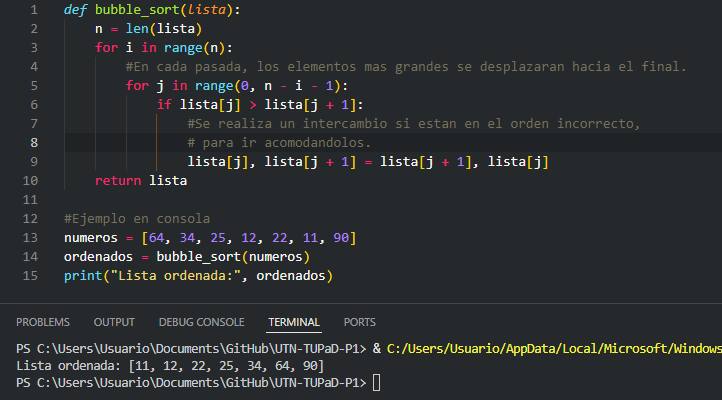
En este caso, para ordenar la lista por inserción, recorrimos los elementos de la lista, comienza desde el segundo elemento y compara hacia atrás. Si encuentra un valor mayor, lo desplaza hacia adelante. Luego inserta el valor actual en el lugar correcto. En la consola podemos ver la lista ordenada.

**Ordenamiento por Selección (Selection Sort):**



En este caso, para ordenar la lista por selección, el algoritmo primero busca y encuentra el elemento más pequeño del arreglo y lo coloca en la primera posición. En la consola podemos ver la lista ordenada.

**Ordenamiento por Burbuja (Bubble Sort):**



En este caso, para ordenar la lista por el método burbuja, compara pares de elementos adyacentes, es decir, que se encuentran de manera continua.