

Trabajo Integrador

Tema : Algoritmos de Búsqueda y Ordenamiento en Python

Alumnos:

- Luna Gonzalo – gonzaloluna561@gmail.com
- González Federico – fedeglz.ok@gmail.com

Materia:

- Programación I

Profesora:

- Julieta Trapé

Tutor:

- Tomás Ferro

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

1. Introducción

Este trabajo aborda el estudio y aplicación práctica de algoritmos de búsqueda y ordenamiento utilizando el lenguaje de programación Python. La elección de este tema se debe a su relevancia en múltiples áreas de la programación, desde bases de datos hasta inteligencia artificial, donde es necesario organizar o acceder a grandes volúmenes de datos de manera eficiente.

El objetivo principal es comprender el funcionamiento de algoritmos clásicos como la búsqueda lineal y binaria, y los métodos de ordenamiento burbuja, selección e inserción, evaluando su rendimiento y utilidad en distintos contextos. A través de una investigación teórica y un desarrollo práctico, se busca afianzar los conocimientos adquiridos durante la cursada.

2. Marco Teórico

En programación, los algoritmos de búsqueda y ordenamiento permiten manipular y organizar datos de forma eficiente. Son fundamentales para tareas como encontrar información, clasificar registros, optimizar procesos y mejorar el rendimiento de un sistema.

- Búsqueda

La búsqueda consiste en localizar un elemento específico dentro de un conjunto de datos. Es una operación común en bases de datos, archivos, IA, etc. (Material de cátedra, 2025).

- **Búsqueda Lineal:**

Recorre cada elemento de la lista hasta encontrar el objetivo. Complejidad: $O(n)$.

```
def busqueda_lineal(lista, objetivo):  
    for i in range(len(lista)):  
        if lista[i] == objetivo:  
            return i  
    return -1
```

- **Búsqueda Binaria:**

Funciona en listas ordenadas, dividiendo el conjunto en mitades. Complejidad: $O(\log n)$.

```
def busqueda_binaria(lista, objetivo):  
    inicio = 0  
    fin = len(lista) - 1  
    while inicio <= fin:  
        medio = (inicio + fin) // 2  
        if lista[medio] == objetivo:  
            return medio  
        elif lista[medio] < objetivo:  
            inicio = medio + 1  
        else:  
            fin = medio - 1  
    return -1
```

- Ordenamiento

Ordenar datos permite trabajar con ellos de manera más eficiente y legible. Existen varios algoritmos clásicos, cada uno con características distintas.

Bubble Sort (Burbuja)

Compara pares de elementos adyacentes e intercambia si están desordenados. Es fácil de entender pero poco eficiente para listas grandes ($O(n^2)$).

Selection Sort (Selección)

Busca el elemento más pequeño y lo coloca en su posición final, repitiendo el proceso con el resto de la lista.

Complejidad (Big O)

Algoritmo	Complejidad Temporal
Búsqueda Lineal	$O(n)$
Búsqueda Binaria	$O(\log n)$
Bubble / Selección	$O(n^2)$

Estos algoritmos permite optimizar programas, mejorar tiempos de ejecución y trabajar con datos de manera más organizada y escalable.

3. Caso Práctico

Ordenamiento:

Para este caso práctico se desarrolló un programa en Python que permite ingresar una lista de autos (marca, modelo y año), y luego los ordena utilizando dos algoritmos clásicos de ordenamiento: **Bubble Sort** (por marca) y **Selection Sort** (por año). El objetivo es mostrar cómo se aplican estos algoritmos a datos reales y observar cómo organizan la información de forma más útil para el usuario.

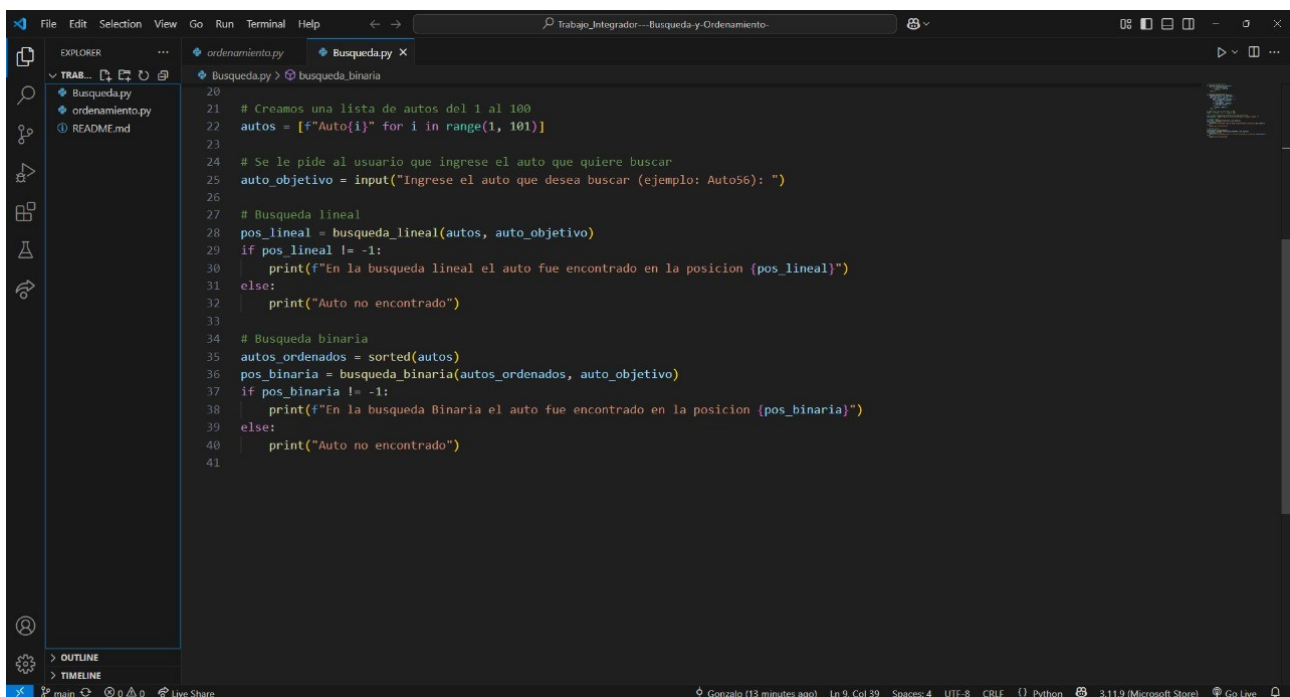
```
ordenamiento.py X
ordenamiento.py > ...
1  autos = []
2
3  cantidad = int(input("Ingrese la cantidad de autos a registrar: "))
4
5  for i in range(cantidad):
6      print(f"\nRegistro del auto {i+1}:")
7      marca = input("Ingrese la marca del auto: ")
8      modelo = input("Ingrese el modelo del auto: ")
9      año = int(input("Ingrese el año del auto: "))
10     autos.append({"marca": marca, "modelo": modelo, "año": año})
11 print()
12
13 #Ordenar autos por marca(Bubble Sort)
14 def ordenar_autos_por_marca(autos):
15     n = len(autos)
16     for i in range(n):
17         for j in range(0, n-i-1):
18             if autos[j]["marca"] > autos[j+1]["marca"]:
19                 autos[j], autos[j+1] = autos[j+1], autos[j]
20
21
22 #Ordenar autos por año(Selection Sort)
23 def ordenar_autos_por_año(autos):
24     n = len(autos)
25     for i in range(n):
26         min_index = i
27         for j in range(i+1, n):
28             if autos[j]["año"] < autos[min_index]["año"]:
29                 min_index = j
30     autos[i], autos[min_index] = autos[min_index], autos[i]
31
32
33 #Probar las funciones de ordenamiento
34 print("Lista de Autos Original:")
35 for auto in autos:
36     print(auto)
37
38 #Ordenar por marca
39 autos_por_marca = autos.copy()
40 ordenar_autos_por_marca(autos_por_marca)
41 print("\nAutos Ordenados por Marca:")
42 for auto in autos_por_marca:
43     print(auto)
44
45 #Ordenar por año
46 autos_por_año = autos.copy()
47 ordenar_autos_por_año(autos_por_año)
48 print("\nAutos Ordenados por Año:")
49 for auto in autos_por_año:
50     print(auto)
51
52
```

- **Ingreso de datos:** el usuario registra la cantidad deseada de autos, ingresando marca, modelo y año.
- **Ordenamiento por marca:** se aplica Bubble Sort, que compara pares de elementos y ordena alfabéticamente.
- **Ordenamiento por año:** se aplica Selection Sort, que selecciona el menor año y lo posiciona correctamente.
- **Salida final:** el programa imprime la lista de autos ordenada por marca y luego por año.

Búsqueda:

Luego del ordenamiento, el programa permite buscar un auto específico utilizando dos métodos:

- **Búsqueda lineal:** recorre toda la lista original de autos ingresados por el usuario hasta encontrar una coincidencia exacta con el formato "marca modelo" (por ejemplo, ford falcon).
- **Búsqueda binaria:** se construye una nueva lista con los autos en formato "marca modelo", se ordena alfabéticamente, y se aplica el algoritmo binario para buscar más rápidamente.



```
20
21 # Creamos una lista de autos del 1 al 100
22 autos = [f"Auto{i}" for i in range(1, 101)]
23
24 # Se le pide al usuario que ingrese el auto que quiere buscar
25 auto_objetivo = input("Ingrese el auto que desea buscar (ejemplo: Auto56): ")
26
27 # Búsqueda lineal
28 pos_lineal = busqueda_lineal(autos, auto_objetivo)
29 if pos_lineal != -1:
30     print(f"En la búsqueda lineal el auto fue encontrado en la posición {pos_lineal}")
31 else:
32     print("Auto no encontrado")
33
34 # Búsqueda binaria
35 autos_ordenados = sorted(autos)
36 pos_binaria = busqueda_binaria(autos_ordenados, auto_objetivo)
37 if pos_binaria != -1:
38     print(f"En la búsqueda Binaria el auto fue encontrado en la posición {pos_binaria}")
39 else:
40     print("Auto no encontrado")
41
```

El usuario ingresa el nombre del auto que desea buscar, y el programa informa si fue encontrado y en qué posición se encuentra, mostrando también la lista usada para la búsqueda binaria.

4. Metodología Utilizada

- Se investigó a partir del material de la cátedra y documentación oficial.
- Se desarrolló el código en Visual Studio Code.
- El trabajo fue realizado de forma colaborativa entre los dos integrantes.
- El proyecto se alojará en GitHub como repositorio público.

5. Resultados Obtenidos

Ordenamiento:

```

fed@fedeg:~/Escritorio/UTN/1er CUATRIMESTRE/PROGRAMACION I/trabajo integrador BUSQUEDA Y ORDENAMIENTO/Codigo Python$ /bin/python3 "/home/fede/Escritorio/UTN/1er CUATRIMESTRE/PROGRAMACION I/trabajo integrador BUSQUEDA Y ORDENAMIENTO/Codigo Python/ordenamiento.py"
Ingrese la cantidad de autos a registrar: 3

Registro del auto 1:
Ingrese la marca del auto: Toyota
Ingrese el modelo del auto: corolla
Ingrese el año del auto: 2024

Registro del auto 2:
Ingrese la marca del auto: Renault
Ingrese el modelo del auto: duster
Ingrese el año del auto: 2017

Registro del auto 3:
Ingrese la marca del auto: Ford
Ingrese el modelo del auto: falcon
Ingrese el año del auto: 1972
-----
Lista de Autos Original:
{'marca': 'Toyota', 'modelo': 'corolla', 'año': 2024}
{'marca': 'Renault', 'modelo': 'duster', 'año': 2017}
{'marca': 'Ford', 'modelo': 'falcon', 'año': 1972}

Autos Ordenados por Marca:
{'marca': 'Ford', 'modelo': 'falcon', 'año': 1972}
{'marca': 'Renault', 'modelo': 'duster', 'año': 2017}
{'marca': 'Toyota', 'modelo': 'corolla', 'año': 2024}

Autos Ordenados por Año:
{'marca': 'Ford', 'modelo': 'falcon', 'año': 1972}
{'marca': 'Renault', 'modelo': 'duster', 'año': 2017}
{'marca': 'Toyota', 'modelo': 'corolla', 'año': 2024}
fed@fedeg:~/Escritorio/UTN/1er CUATRIMESTRE/PROGRAMACION I/trabajo integrador BUSQUEDA Y ORDENAMIENTO/Codigo Python$

```

Busqueda:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL

Microsoft Windows [Versión 10.0.26100.4061]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\gonza\OneDrive\Escritorio\Python\TP BUSQUEDAS BINARIAS Y LINEALES\Trabajo_Integrador---Busqueda-y-Ordenamiento->C:/Users/gonza/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/gonza/OneDrive/Escritorio/Python/TP BUSQUEDAS BINARIAS Y LINEALES/Trabajo_Integrador---Busqueda-y-Ordenamiento-/Busqueda.py"
Ingrese el auto que desea buscar (ejemplo: Auto56): Auto12
En la busqueda lineal el auto fue encontrado en la posicion 11
En la busqueda Binaria el auto fue encontrado en la posicion 4

C:\Users\gonza\OneDrive\Escritorio\Python\TP BUSQUEDAS BINARIAS Y LINEALES\Trabajo_Integrador---Busqueda-y-Ordenamiento->

```

6. Conclusiones

Logramos aplicar de manera práctica varios de los conceptos fundamentales de la programación, como el uso de funciones, estructuras de datos y algoritmos. Utilizamos dos métodos de ordenamiento y dos de búsqueda, entendiendo no solo cómo funcionan, sino también en qué casos conviene usar cada uno.

El trabajar con una temática cercana como los autos nos ayudó a visualizar mejor cómo estos algoritmos organizan y procesan información en situaciones reales. Además, separar el código en módulos y crear un archivo principal (`main.py`) nos permitió trabajar de forma más ordenada y profesional.

Durante el desarrollo del trabajo Integrador nos enfrentamos a algunos desafíos, sobre todo al momento de combinar búsqueda y ordenamiento, pero nos sirvió para repasar lógica y fortalecer el trabajo en equipo.

En resumen, el realizar el trabajo integrador nos ayudó a afianzar nuestros conocimientos, y prepararnos mejor para los futuros proyectos de programación.

7. Bibliografía

- Apuntes personales tomados durante los encuentros virtuales.
- Material teórico aportado por la cátedra de Programación I
- https://www.w3schools.com/python/python_dsa_bubblesort.asp
- https://www.w3schools.com/python/python_dsa_selectionsort.asp

8. Anexos

- Capturas de pantalla del código en ejecución
- Enlace al repositorio de GitHub : https://github.com/fedeglz/Trabajo_Integrador---Busqueda-y-Ordenamiento-
- Enlace al video explicativo : <https://www.youtube.com/watch?v=CB5xYSaYrDU>