Titulo del trabajo : Algoritmos de busqueda y ordenamiento en Python

Alumnos :

Lautaro Burgos – comisión 10 lautaroburgos244@gmail.com

Mauricio Campioni – comisión 11 mauriciocampioni8@gmail.com

Materia : Programacion 1

Profesor/a : Cinthia Rigoni

Fecha de entrega : 09/06/2025

Indice

1.Introduccion

El presente trabajo tiene como objetivo explorar el funcionamiento de los algoritmos de ordenamiento y búsqueda aplicados sobre una estructura más compleja: una lista de objetos. En este caso, se desarrolló una clase `Cliente` que representa una entidad con dos atributos: número de cliente y nombre.

El ordenamiento y la búsqueda son operaciones fundamentales en la programación, y su aplicación a objetos permite resolver problemas reales como la gestión de usuarios en sistemas o bases de datos. Se utilizaron los algoritmos de Bubble Sort para ordenar y Búsqueda Binaria para localizar un cliente por su número, adaptando ambos a la estructura de objetos.

Este trabajo busca reforzar los conceptos de ordenamiento y búsqueda, a la vez que introduce la necesidad de adaptar algoritmos a estructuras no primitivas.

2. Marco Teórico

Los algoritmos de ordenamiento permiten reorganizar una colección de datos en función de un criterio. El Bubble Sort compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto, repitiendo este proceso hasta que toda la lista está ordenada. Aunque no es el más eficiente, su implementación es sencilla y adecuada para fines educativos.

La Búsqueda Binaria es un método eficiente para localizar un valor dentro de una lista previamente ordenada. Consiste en comparar el valor deseado con el elemento central, descartando la mitad del arreglo en cada iteración. Su complejidad es O(log n), lo cual la vuelve muy eficiente para listas grandes.

En Python, al trabajar con objetos, es necesario especificar por qué atributo se realiza la comparación. En este caso, tanto el ordenamiento como la búsqueda se realizaron según el atributo `numero` de cada objeto `Cliente`.

3. Caso Práctico

Se creó una clase `Cliente` que contiene dos atributos: `numero` y `nombre`. A partir de una lista de objetos Cliente, se implementaron las funciones `bubble_sort_clientes` y `busqueda_binaria_clientes`, que ordenan y buscan elementos según el número de cliente.

El siguiente es el codigo utilizado con clientes ficticios:

```
# Definicion de clase Cliente
class Cliente:
    def __init__(self, numero, nombre):
        self.numero = numero
        self.nombre = nombre

    def __repr__(self):
        return f"Cliente({self.numero}, '{self.nombre}')"

# Algoritmo de ordenamiento Bubble Sort adaptado para objetos de
tipo Cliente
```

```
def bubble_sort_clientes(clientes):
    n = len(clientes)
    for i in range(n):
        for j in range(0, n - i - 1):
            # Comparamos por el número de cliente
            if clientes[j].numero > clientes[j + 1].numero:
                clientes[j], clientes[j + 1] = clientes[j + 1],
clientes[j]

# Algoritmo de busqueda binaria adaptado para objetos de tipo
Cliente
def busqueda_binaria_clientes(clientes, numero_buscado):
    inicio = 0
    fin = len(clientes) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if clientes[medio].numero == numero_buscado:
            return medio
        elif clientes[medio].numero < numero_buscado:
            inicio = medio + 1
        else:
            fin = medio - 1
    return -1

# Lista de clientes (desordenada)
clientes = [
    Cliente(102, "Julian Alvarez"),
    Cliente(56, "Alexis Macallister"),
    Cliente(78, "Emiliano Martinez"),
    Cliente(33, "Lisandro Martinez"),
    Cliente(150, "Cristian Romero")
]

# Mostrar lista original
print("Lista original de clientes:")
for cliente in clientes:
    print(cliente)

# Ordenar la lista
bubble_sort_clientes(clientes)

# Mostrar lista ordenada
```

```
print("\\nLista de clientes ordenada por número:")
for cliente in clientes:
    print(cliente)

# Buscar un cliente por numero de cliente.
numero_a_buscar = 78 # Deberia encontrar al cliente Emiliano
Martinez
indice_encontrado = busqueda_binaria_clientes(clientes,
numero_a_buscar)

# Mostrar resultado de la búsqueda
if indice_encontrado != -1:
    print(f"\\nCliente con número {numero_a_buscar} encontrado en la
posición {indice_encontrado}:")
    print(clientes[indice_encontrado])
else:
    print(f"\\nCliente con número {numero_a_buscar} no encontrado.")
```

4.**Metodología Utilizada**

El desarrollo del trabajo siguió las siguientes etapas:

- Investigación de algoritmos básicos en fuentes oficiales de Python.
- Diseño de una clase personalizada con atributos simples.
- Implementación de los algoritmos adaptados a objetos.
- Prueba del código con distintos conjuntos de datos.
- Validación de resultados mediante impresión en consola.

Se utilizó Visual Studio Code como entorno de desarrollo, Python 3.12 como lenguaje de programación y pruebas manuales en la terminal.

5.Resultados Obtenidos

- Se obtuvo un ordenamiento correcto de objetos `Cliente` según su atributo `numero`.
- La búsqueda binaria localizó correctamente un cliente dado.
- Se comprobó la importancia de tener los datos ordenados para aplicar búsqueda binaria.

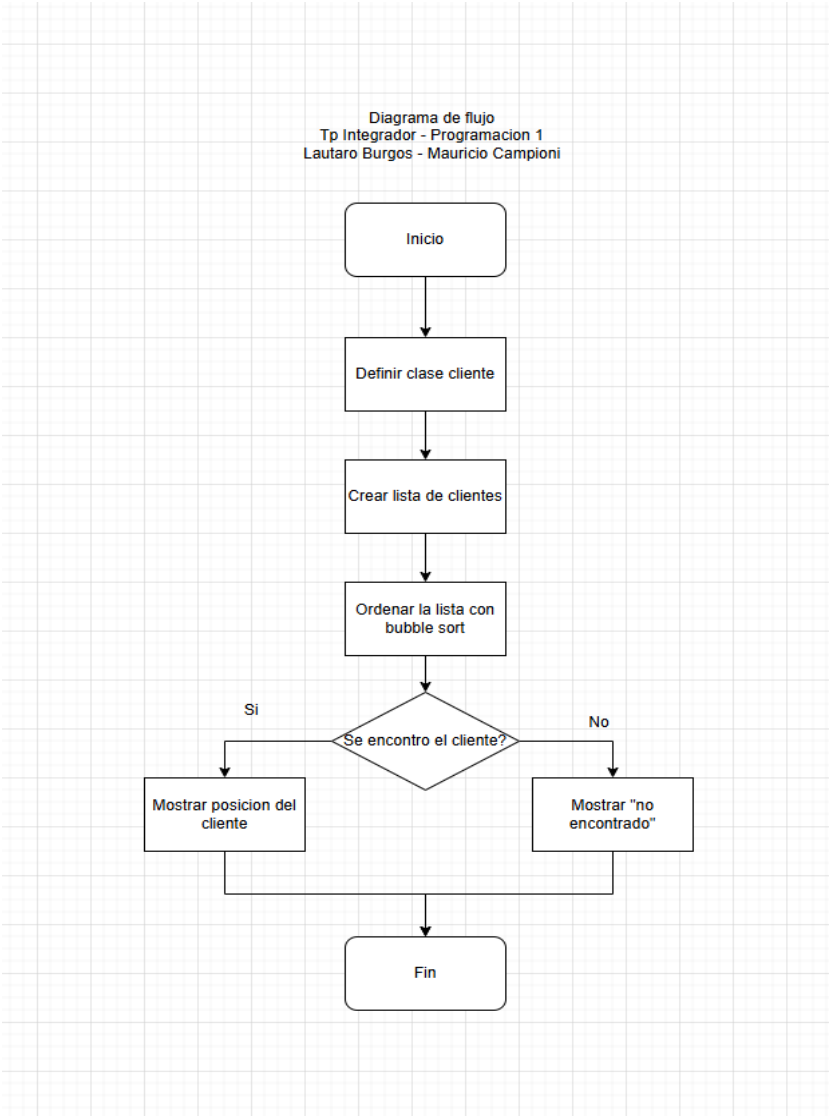
6. Conclusiones

Este trabajo permitió aplicar los algoritmos de ordenamiento y búsqueda en un contexto más cercano al uso real, trabajando con objetos. Se consolidó la comprensión de estructuras de datos, la adaptación de algoritmos clásicos y la importancia de comparar correctamente atributos.

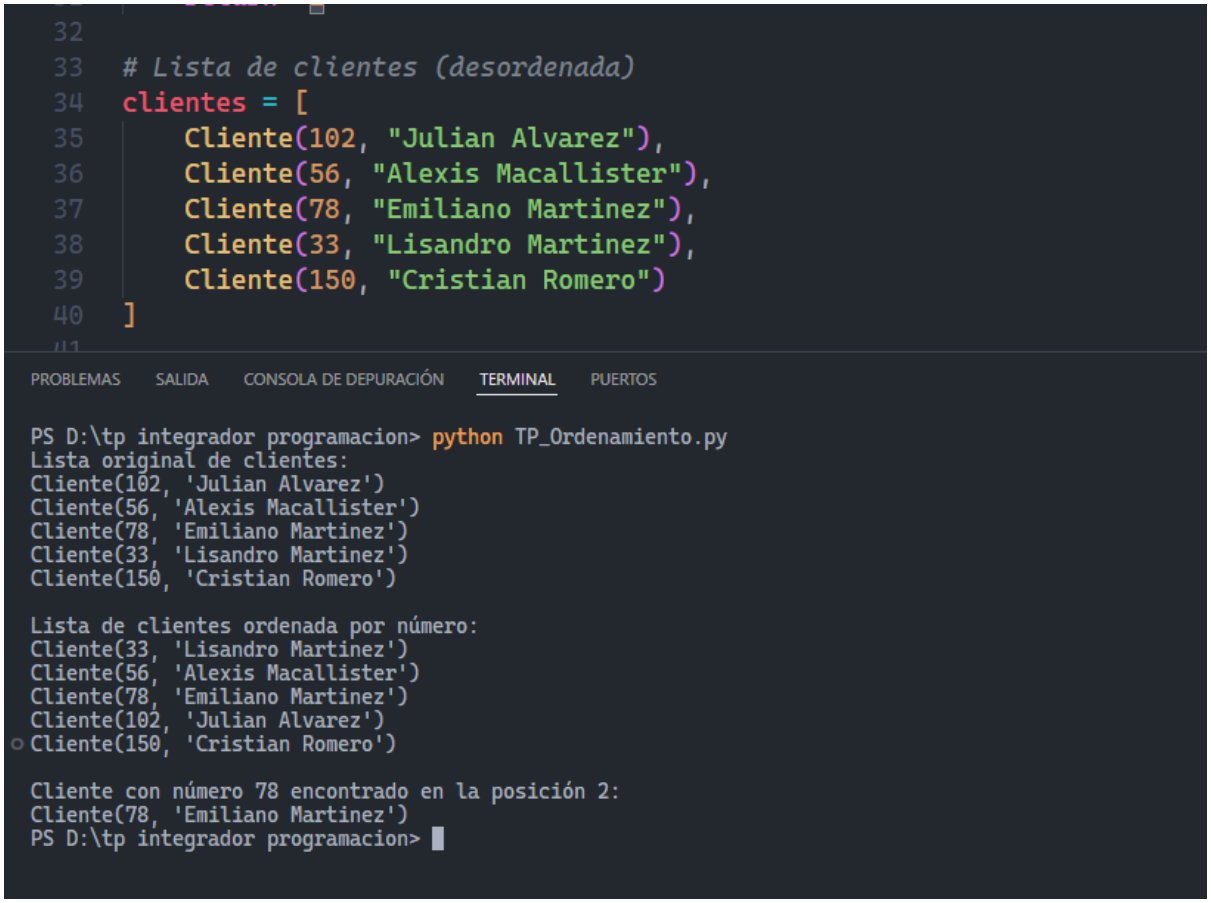
Como mejora futura, se podría implementar ordenamiento por otros criterios (como nombre) o usar algoritmos más eficientes como Merge Sort. También podría agregarse una interfaz gráfica o leer los datos desde un archivo externo.

7.Anexos

Diagrama de flujo



Captura de pantalla del programa funcionando



```
32
33  # Lista de clientes (desordenada)
34  clientes = [
35      Cliente(102, "Julian Alvarez"),
36      Cliente(56, "Alexis Macallister"),
37      Cliente(78, "Emiliano Martinez"),
38      Cliente(33, "Lisandro Martinez"),
39      Cliente(150, "Cristian Romero")
40  ]
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS D:\tp integrador programacion> python TP_Ordenamiento.py
Lista original de clientes:
Cliente(102, 'Julian Alvarez')
Cliente(56, 'Alexis Macallister')
Cliente(78, 'Emiliano Martinez')
Cliente(33, 'Lisandro Martinez')
Cliente(150, 'Cristian Romero')

Lista de clientes ordenada por número:
Cliente(33, 'Lisandro Martinez')
Cliente(56, 'Alexis Macallister')
Cliente(78, 'Emiliano Martinez')
Cliente(102, 'Julian Alvarez')
Cliente(150, 'Cristian Romero')

Cliente con número 78 encontrado en la posición 2:
Cliente(78, 'Emiliano Martinez')
PS D:\tp integrador programacion>
```