Tecnicatura Universitaria en Programación a Distancia	

Tema: Algoritmos de Búsqueda y Ordenamiento

Alumnos:
Claudio Fiorito – claudio80.cf@gmail.com
Diana Falla – diana.falla.cba@gmail.com

Materia: Programación I

Profesor: Ariel Enferrel

2025

Índice

1. Introducción

En el desarrollo de la asignatura Programación I, se estudiaron distintos algoritmos que permiten organizar y buscar información de manera eficiente. Los algoritmos de búsqueda y ordenamiento son herramientas clave en informática y programación, ya que permiten trabajar con grandes volúmenes de datos de forma más estructurada. En este trabajo decidimos poner en práctica algunos de ellos utilizando el lenguaje Python, para comprender cómo funcionan en la práctica y cuáles son sus ventajas.

2. Marco Teórico

Los algoritmos de búsqueda y ordenamiento forman parte de los fundamentos esenciales en programación, ya que permiten gestionar colecciones de datos de manera eficiente. Estas técnicas no solo optimizan el rendimiento de las aplicaciones, sino que también mejoran la organización de la información para su posterior análisis o procesamiento.

Ordenamiento

Ordenar significa reacomodar los elementos de una lista según algún criterio (por ejemplo, de menor a mayor). Algunos métodos básicos son:

- **Bubble Sort:** compara elementos vecinos y los intercambia si están desordenados.
- **Insertion Sort:** va insertando cada elemento en su lugar correcto dentro de una lista ordenada.
- **Selection Sort:** busca el valor más pequeño y lo coloca en su posición final.

Estos algoritmos son fáciles de entender y programar, aunque no son los más rápidos para listas muy grandes.

Búsqueda

Buscar es encontrar un valor dentro de una lista. Existen dos métodos comunes:

- **Búsqueda lineal:** revisa todos los elementos uno por uno. Sirve para cualquier lista.
- **Búsqueda binaria:** solo funciona si la lista ya está ordenada. Es mucho más rápida porque va dividiendo la lista a la mitad.

Estos algoritmos permiten trabajar con los datos de forma más ordenada y son una base importante para seguir aprendiendo programación.

3. Caso Práctico

Se desarrolló un programa en Python que permite ordenar una lista de edades de un grupo de estudiantes utilizando el algoritmo **Selection Sort**. Luego, el programa solicita al usuario una edad específica y verifica si dicha edad se encuentra en la lista mediante el uso del algoritmo de **búsqueda lineal**.

Objetivo:
Practicar la implementación de algoritmos clásicos de ordenamiento y búsqueda, aplicándolos sobre una lista de números enteros. En este caso, las edades utilizadas permiten contextualizar el ejercicio en una situación cotidiana y sencilla para reforzar los conocimientos adquiridos en esta materia.

CÓDIGO EN PYTHON:

```
from colorama import init, Fore, Back, Style

init()

# Algoritmo de ordenamiento: Selection Sort

def ordenamiento_s_sort(lista):

    n = len(lista)

    for i in range(n):

        min_indice = i

        for j in range(i+1, n):

            if lista[j] < lista[min_indice]:

                min_indice = j

        lista[i], lista[min_indice] = lista[min_indice], lista[i]
```

```
# Algoritmo de búsqueda: Búsqueda lineal

def busqueda_lineal(lista, valor):

    for i in range(len(lista)):

        if lista[i] == valor:

            return i

    return -1



# Lista de edades

edades = [21, 18, 25, 19, 22, 20, 23]

print(Fore.GREEN + "Lista de EDADES de los estudiantes:", edades)



# Ordenar la lista

ordenamiento_s_sort(edades)

print(Fore.CYAN + "Lista ordenada:", edades)



# Buscar una edad específica

edad_buscada = int(input(Fore.LIGHTMAGENTA_EX + "Ingrese la edad a buscar: "))

posicion = busqueda_lineal(edades, edad_buscada)
```

```
if posicion != -1:
    print(f"La edad {edad_buscada} se encuentra en la posición {posicion} de la lista.")
else:
    print(f"La edad {edad_buscada} no se encuentra en la lista.")
```

4. Metodología Utilizada

Para la realización de este trabajo, se eligió una lista sencilla que representará edades de estudiantes, ya que resultaba una forma accesible de aplicar los algoritmos. A continuación, se buscó información para comprender el funcionamiento del algoritmo **Selection Sort**, utilizado para ordenar listas, y el de **búsqueda lineal**, que permite encontrar un dato dentro de una colección.

Una vez entendida la lógica de ambos algoritmos, se comenzó a escribir el código en Python paso a paso. Primero se programó la función encargada del ordenamiento y luego la función de búsqueda. El programa fue probado en distintas situaciones, con diferentes datos, para verificar su correcto funcionamiento.

Con el programa ya completo, se elaboró este informe con el objetivo de explicar el trabajo realizado y compartir lo aprendido durante el proceso.

5. Resultados Obtenidos

El programa desarrollado logró ordenar correctamente la lista de edades utilizando el algoritmo **Selection Sort**. Luego, mediante el uso de **búsqueda lineal**, se pudo localizar un valor ingresado por el usuario de forma efectiva. Las pruebas realizadas permitieron comprobar que ambas funciones realizaban lo que se esperaba. Además, se confirmó que el ordenamiento previo facilita la lectura y organización de los datos.

6. Conclusiones

El desarrollo de este trabajo nos permitió aprender cómo implementar y aplicar algoritmos de búsqueda y ordenamiento. Se comprendió la diferencia entre métodos secuenciales y métodos que requieren estructuras ordenadas. Además, se reforzaron conceptos clave como el uso de listas, estructuras condicionales y bucles en el lenguaje de programación Python.

7. Bibliografía

- Python Software Foundation. (s.f.). Python 3 documentation. https://docs.python.org/3/
- W3Schools Python Tutorial. https://www.w3schools.com/python/
- Material de clase de Programación I, UTN.

8. Anexos

- Video explicativo Youtube : https://youtu.be/5SyTiZOR6Yc
- Repositorio en GitHub: https://github.com/Diana-Falla/TP-Integrador-Programacion