Título del proyecto:

Búsqueda binaria y ordenamiento por selección en Python

Alumnos:

Senger Edgardo Nicolás (nicosenger98@gmail.com)

Materia:

Programación I

Profesor:

Cinthia Rigoni

Fecha de Entrega:

9 de Junio de 2025

1. Introducción

Los algoritmos de búsqueda y ordenamiento son fundamentales para la programación eficiente. En este trabajo se investigan dos algoritmos clásicos: la búsqueda binaria y el ordenamiento por selección. Se analiza cómo funcionan y se implementan en Python.

2. Marco Teórico

Búsqueda binaria

La búsqueda binaria es un algoritmo eficiente para encontrar un elemento en una lista ordenada.

- Divide el rango de búsqueda a la mitad en cada paso.
- Requiere que los datos estén ordenados.
- Complejidad: O(log n).

Ordenamiento por selección

Este algoritmo ordena una lista encontrando el valor mínimo y colocándolo al principio, repitiendo el proceso para el resto.

- Sencillo de implementar.
- No es muy eficiente para grandes volúmenes de datos.
- Complejidad: $O(n^2)$.

Aplicaciones

- Búsqueda binaria: diccionarios digitales, bases de datos indexadas, juegos.
- Ordenamiento: preparación de datos, organización previa a búsquedas.

3. Caso Práctico

Se desarrolló una aplicación en Python que:

1. Ordena una lista de números con ordenamiento por selección.
2. Permite buscar un valor en esa lista usando búsqueda binaria.

Código principal (busqueda_ordenamiento.py):

```
def ordenamiento_seleccion(lista):
    n = len(lista)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if lista[j] < lista[min_idx]:
                min_idx = j
        lista[i], lista[min_idx] = lista[min_idx], lista[i]
    return lista

def busqueda_binaria(lista, objetivo):
    izquierda = 0
    derecha = len(lista) - 1

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista[medio] == objetivo:
            return medio
        elif lista[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    return -1

numeros = [22, 11, 99, 88, 9, 7, 42]
print("Lista original:", numeros)

ordenada = ordenamiento_seleccion(numeros.copy())
print("Lista ordenada:", ordenada)

valor = int(input("Ingrese un número a buscar: "))
resultado = busqueda_binaria(ordenada, valor)
```

```
if resultado != -1:
    print(f"El número {valor} fue encontrado en la posición {resultado}.")
else:
    print(f"El número {valor} no está en la lista.")
```

4. Metodología Utilizada

- Estudio de algoritmos clásicos de búsqueda y ordenamiento.
- Implementación paso a paso en Python.
- Ejecución con diferentes listas para observar el comportamiento.
- Pruebas interactivas para verificar funcionamiento real.
- Optimización del código utilizando chat gpt.

5. Resultados Obtenidos

- Se logró ordenar correctamente la lista con el algoritmo de selección.
- La búsqueda binaria permitió encontrar o descartar valores rápidamente.
- Se verificó que la búsqueda falla si la lista no está ordenada previamente.

6. Conclusiones

El ordenamiento previo de datos es fundamental para aplicar algoritmos eficientes como la búsqueda binaria. Aunque el ordenamiento por selección no es óptimo para grandes listas, resulta útil para enseñar la lógica de comparación. Esta práctica permitió reforzar los conceptos de algoritmos, estructuras secuenciales y análisis de eficiencia.

Reflexión: Aplicar estos algoritmos en Python ayuda a comprender la lógica detrás de muchas aplicaciones del mundo real, como motores de búsqueda o sistemas de recomendación.

7. Bibliografía

- Documentación oficial de Python: https://docs.python.org/3/
- Documentación de la materia.
- Chat gpt.