Trabajo Integrador Final - Algoritmos de Búsqueda y Ordenamiento en Python

Universidad Tecnológica Nacional - Facultad Regional Avellaneda

Tecnicatura Universitaria en Programación - Modalidad a Distancia

Ciclo Lectivo 2025

Fecha de entrega: 9/6/2025

Alumnos: Matías D'Agostino, Agustina Milagros Cruz

Correos: matias.dagostino@tupad.utn.edu.ar, agustina.cruz@tupad.utn.edu.ar

Materia: Programación I

Tutor: Luciano Chiroli

Profesor: Ariel Enferrel

GitHub: https://github.com/codewtato/UTN-TUPaD-P1/tree/main/10%20B%C3%BAsqueda%20y%20ordenamiento

Índice

1. Introducción

Este trabajo se centra en el estudio de algoritmos de búsqueda y ordenamiento, fundamentales en programación para organizar y acceder eficientemente a datos. Se eligió este tema debido a su aplicabilidad práctica en sistemas reales y su importancia en estructuras de datos y rendimiento computacional. El objetivo principal es comprender, implementar y probar algoritmos de búsqueda y ordenamiento en Python, evaluando su comportamiento y eficiencia.

2. Marco Teórico

Los algoritmos de ordenamiento y búsqueda son esenciales para organizar y acceder a datos de manera eficiente. Su correcta elección puede impactar directamente en el rendimiento de una aplicación, especialmente cuando se trabaja con grandes volúmenes de datos.

El algoritmo de ordenamiento implementado fue Bubble Sort, conocido por su simplicidad pero también por su baja eficiencia en listas grandes (complejidad $O(n^2)$). Para la búsqueda, se utilizó búsqueda por interpolación, un algoritmo que, en listas ordenadas y con datos uniformemente distribuidos, puede ser más eficiente que la búsqueda binaria (mejor caso: $O(\log \log n)$).

3. Caso Práctico

En este trabajo se desarrollaron e implementaron dos algoritmos principales: uno de ordenamiento (Bubble Sort) y uno de búsqueda (búsqueda por interpolación). Ambos fueron programados en Python y probados sobre listas numéricas de diferentes tamaños.

Ordenamiento con Bubble Sort

Antes de aplicar el algoritmo de búsqueda, fue necesario ordenar la lista. Para esto se utilizó el algoritmo Bubble Sort, que recorre la lista varias veces, comparando elementos adyacentes e intercambiándolos si están desordenados. Aunque no es el método más eficiente en términos de tiempo, su implementación es sencilla y adecuada para listas pequeñas.

Código en Python:

```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
```

```
            lista[j], lista[j + 1] = lista[j + 1], lista[j]
    return lista
```

Búsqueda por Interpolación

Una vez ordenada la lista, se aplicó la búsqueda por interpolación en su forma iterativa. Este algoritmo es especialmente eficiente cuando los datos están distribuidos de manera uniforme, ya que estima la posición probable del elemento buscado, en lugar de simplemente dividir el espacio en mitades como hace la búsqueda binaria. La versión iterativa ofrece un mejor control del proceso, mayor eficiencia en memoria y facilita su depuración.

Código en Python:

```
def busqueda_interpolacion(lista, objetivo):
    izquierda = 0
    derecha = len(lista) - 1
    while izquierda <= derecha and lista[izquierda] <= objetivo <= lista[derecha]:
        if izquierda == derecha:
            if lista[izquierda] == objetivo:
                return izquierda
            return -1
        pos = izquierda + ((objetivo - lista[izquierda]) * (derecha - izquierda) //
                    (lista[derecha] - lista[izquierda]))
        if lista[pos] == objetivo:
            return pos
        elif lista[pos] < objetivo:
            izquierda = pos + 1
        else:
            derecha = pos - 1
    return -1
```

Pruebas de funcionamiento

Se realizaron dos pruebas:

- Una con una lista del 1 al 100 buscando el valor 90.
- Otra con una lista del 1 al 10.000 buscando el valor 10.000.

En ambas se desordenó primero la lista, luego se ordenó con Bubble Sort y finalmente se buscó el valor con interpolación. Los resultados demostraron que ambos algoritmos funcionaron correctamente, aunque el ordenamiento fue significativamente más lento en la lista grande, como se esperaba.

4. Metodología Utilizada

El trabajo se desarrolló en lenguaje Python 3.11. Se implementaron los algoritmos paso a paso en Visual Studio Code y se utilizaron funciones del módulo `time` para medir los tiempos de ejecución de cada operación. Se utilizaron listas ordenadas y desordenadas para probar el funcionamiento correcto de los algoritmos.

5. Resultados Obtenidos

Las pruebas realizadas demostraron que el algoritmo de búsqueda por interpolación funcionó correctamente en ambos casos, siendo muy eficiente en listas grandes. En cambio, el algoritmo Bubble Sort, aunque preciso, presentó un rendimiento pobre cuando se trabajó con listas extensas. Esto refleja la necesidad de utilizar algoritmos de ordenamiento más avanzados en contextos reales.

```
PS C:\Users\code\Desktop\Pruebas> & C:/Users/code/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/code/Downloads/busquedayordenamiento.py
=== ORDENAMIENTO (Bubble Sort) ===
Lista ordenada: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88
, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
Tiempo de ordenamiento: 0.0004104000 segundos

=== BÚSQUEDA (Interpolación) ===
Elemento buscado: 90
Encontrado en posición: 89
Tiempo de búsqueda: 0.0000031000 segundos
PS C:\Users\code\Desktop\Pruebas>
```

```
580, 9581, 9582, 9583, 9584, 9585, 9586, 9587, 9588, 9589, 9590, 9591, 9592, 9593, 9594, 9595, 9596, 9597, 9598, 9599, 9600, 9601, 9602, 9603, 9604, 9605, 9606, 9607, 9608, 9609,
9610, 9611, 9612, 9613, 9614, 9615, 9616, 9617, 9618, 9619, 9620, 9621, 9622, 9623, 9624, 9625, 9626, 9627, 9628, 9629, 9630, 9631, 9632, 9633, 9634, 9635, 9636, 9637, 9638, 9639,
 9640, 9641, 9642, 9643, 9644, 9645, 9646, 9647, 9648, 9649, 9650, 9651, 9652, 9653, 9654, 9655, 9656, 9657, 9658, 9659, 9660, 9661, 9662, 9663, 9664, 9665, 9666, 9667, 9668, 9669
, 9670, 9671, 9672, 9673, 9674, 9675, 9676, 9677, 9678, 9679, 9680, 9681, 9682, 9683, 9684, 9685, 9686, 9687, 9688, 9689, 9690, 9691, 9692, 9693, 9694, 9695, 9696, 9697, 9698, 969
9, 9700, 9701, 9702, 9703, 9704, 9705, 9706, 9707, 9708, 9709, 9710, 9711, 9712, 9713, 9714, 9715, 9716, 9717, 9718, 9719, 9720, 9721, 9722, 9723, 9724, 9725, 9726, 9727, 9728, 97
29, 9730, 9731, 9732, 9733, 9734, 9735, 9736, 9737, 9738, 9739, 9740, 9741, 9742, 9743, 9744, 9745, 9746, 9747, 9748, 9749, 9750, 9751, 9752, 9753, 9754, 9755, 9756, 9757, 9758, 9
759, 9760, 9761, 9762, 9763, 9764, 9765, 9766, 9767, 9768, 9769, 9770, 9771, 9772, 9773, 9774, 9775, 9776, 9777, 9778, 9779, 9780, 9781, 9782, 9783, 9784, 9785, 9786, 9787, 9788,
9789, 9790, 9791, 9792, 9793, 9794, 9795, 9796, 9797, 9798, 9799, 9800, 9801, 9802, 9803, 9804, 9805, 9806, 9807, 9808, 9809, 9810, 9811, 9812, 9813, 9814, 9815, 9816, 9817, 9818,
 9819, 9820, 9821, 9822, 9823, 9824, 9825, 9826, 9827, 9828, 9829, 9830, 9831, 9832, 9833, 9834, 9835, 9836, 9837, 9838, 9839, 9840, 9841, 9842, 9843, 9844, 9845, 9846, 9847, 9848
, 9849, 9850, 9851, 9852, 9853, 9854, 9855, 9856, 9857, 9858, 9859, 9860, 9861, 9862, 9863, 9864, 9865, 9866, 9867, 9868, 9869, 9870, 9871, 9872, 9873, 9874, 9875, 9876, 9877, 987
8, 9879, 9880, 9881, 9882, 9883, 9884, 9885, 9886, 9887, 9888, 9889, 9890, 9891, 9892, 9893, 9894, 9895, 9896, 9897, 9898, 9899, 9900, 9901, 9902, 9903, 9904, 9905, 9906, 9907, 99
08, 9909, 9910, 9911, 9912, 9913, 9914, 9915, 9916, 9917, 9918, 9919, 9920, 9921, 9922, 9923, 9924, 9925, 9926, 9927, 9928, 9929, 9930, 9931, 9932, 9933, 9934, 9935, 9936, 9937, 9
938, 9939, 9940, 9941, 9942, 9943, 9944, 9945, 9946, 9947, 9948, 9949, 9950, 9951, 9952, 9953, 9954, 9955, 9956, 9957, 9958, 9959, 9960, 9961, 9962, 9963, 9964, 9965, 9966, 9967,
9968, 9969, 9970, 9971, 9972, 9973, 9974, 9975, 9976, 9977, 9978, 9979, 9980, 9981, 9982, 9983, 9984, 9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997,
 9998, 9999, 10000]
Tiempo de ordenamiento: 4.0566549000 segundos

=== BÚSQUEDA (Interpolación) ===
Elemento buscado: 10000
Encontrado en posición: 9999
Tiempo de búsqueda: 0.0000054000 segundos
```

6. Conclusiones

El análisis realizado permitió comprender el funcionamiento, ventajas y limitaciones de los algoritmos seleccionados. Se verificó la importancia de aplicar ordenamiento previo cuando se utiliza un algoritmo de búsqueda como la interpolación. Además, se evidenció cómo el tamaño de los datos influye directamente en el rendimiento, especialmente en algoritmos como Bubble Sort.

7. Bibliografía

- Video sobre algoritmos de búsqueda y ordenamiento:
https://www.youtube.com/watch?v=gJlQTq80lIg
- Explicación de algoritmos:

https://www.youtube.com/watch?v=xntUhrhtLaw
- Casos prácticos de búsqueda y ordenamiento:
https://www.youtube.com/watch?v=u1QuRbx-_x4
- Google Colab:
https://colab.research.google.com/drive/1KVqiJSzYLTPDFRwTYjN8CP7G4LPreD9J?usp=sharing

8. Anexos

- Capturas del programa funcionando.

- Código fuente en GitHub: https://github.com/codewtato/UTN-TUPaD-P1/tree/main/10%20B%C3%BAsqueda%20y%20ordenamiento

- Link del video explicativo: https://youtu.be/BQmNQFKR354?si=iYNRxf9vIKmo5-Aq