Trabajo Práctico - Algoritmo de Búsqueda y Ordenamiento

Alumnos:

Verdun Walter - walterame123@gmail

Zarate Lucas - zaratelucasmartin1@gmail.com

Materia: Programación 1

Profesor: AUS Bruselario, Sebastian

Docente Tutor: Gubiotti, Flor

Fecha de entrega: 9 junio 2025

Índice

Introducción

En el ámbito de la programación, los algoritmos de búsqueda y ordenamiento son piezas clave para manejar la información de forma eficiente. Gracias a ellos, es posible organizar y encontrar datos de manera más rápida y precisa, lo cual resulta fundamental en el desarrollo de software. Estos algoritmos no solo ayudan a que los programas funcionen más rápido, sino que también facilitan resolver problemas difíciles de forma más sencilla y práctica.

Este trabajo busca explicar los algoritmos más comunes en estas áreas, tanto los que sirven para buscar datos como los que se usan para ordenarlos. La idea es entender cómo funcionan, cuáles son sus ventajas y cuándo es mejor usarlos. Se hará especial énfasis en la búsqueda binaria, porque es una técnica muy eficiente cuando los datos ya están ordenados. Conocer estos algoritmos es muy útil para cualquier programador que quiera escribir código más limpio y efectivo.

Marco Teórico

Un algoritmo de ordenamiento es un procedimiento que permite organizar los elementos de una lista o vector según un criterio específico de orden. Es decir, toma una colección de datos y los reordena de manera que el resultado cumpla con una relación de orden establecida, asegurando que los elementos están dispuestos en la secuencia correcta.

Algunos de los algoritmos de ordenamiento más comunes se encuentran:

Bubble Sort: compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite varias veces, haciendo que los elementos más grandes “burbujeen” hacia el final hasta que toda la lista queda ordenada.

Quick Sort: Elige un elemento pivote y divide la lista en dos partes, una con los elementos menores que el pivote y otra con los mayores. Luego ordena recursivamente esas partes y las combina para obtener la lista completamente ordenada.

Insertion Sort: Construye una lista ordenada tomando cada elemento de la lista original uno a uno e insertándose en la posición correcta dentro de la parte ya ordenada, desplazando elementos si es necesario.

Selection Sort: Busca el elemento más pequeño del arreglo y lo intercambia con el primer elemento sin ordenar. Repite este proceso para la sublista restante hasta que toda la lista esté ordenada.

Por otro lado, los algoritmos de búsqueda tienen como objetivo localizar un valor específico dentro de una colección de datos. Dependiendo de cómo estén organizados los datos, se puede utilizar un tipo de búsqueda más eficiente que otro.

Destacamos:

Búsqueda lineal: Recorre la colección elemento por elemento, de forma secuencial, comparando cada uno con el valor buscado. Si encuentra una coincidencia, devuelve su posición. Es sencilla y útil cuando los datos no están ordenados, aunque puede ser lenta en listas largas.

Búsqueda binaria: Es un método mucho más eficiente, pero solo se puede aplicar en listas ordenadas. Consiste en comparar el valor buscado con el elemento que está en el centro de la lista.
La búsqueda binaria funciona de la siguiente manera:

1. Si coinciden, se ha encontrado la posición del elemento.
2. Si el valor buscado es menor, se repite la búsqueda en la mitad izquierda de la lista.
3. Si es mayor, se repite en la mitad derecha. Este proceso se repite dividiendo el espacio de búsqueda en mitades sucesivas hasta encontrar el valor o hasta que ya no haya más elementos por revisar.

Caso Práctico

El objetivo de observar la práctica el comportamiento de los algoritmos de ordenamiento y la búsqueda binaria. Se implementó un programa en Python el tiempo de ordenamiento de Quicksort y Bubble Sort sobre una matriz de lista de productos aleatorios, simulando una base de datos de una Frutería. La lógica consiste en generar listas de tamaño grandes y aplicar ambos algoritmos para medir cuánto tiempo tarda en ordenar dicha lista. Además, implementó un algoritmo de búsqueda binaria las cual nos trae la lista de producto según el índice indicado, para quicksort fue utilizado por búsqueda de precio y por bubble sort búsqueda por vitaminas.

Función del algoritmo Quicksort:

```python
#quick Sort
def quick_sort(matriz):
    if len(matriz) <= 1:
        return matriz
    pivote = matriz[0]
    menores = []
    mayores = []

    for fila in matriz[1:]:
        if fila[2] <= pivote[2]:
            menores.append(fila)
        else:
            mayores.append(fila)

    return quick_sort(menores) + [pivote] + quick_sort(mayores)
```

Función de algoritmo Bubble Sort:

```python
# Bubble Sort
def bubble_sort(matriz):
    n = len(matriz)
    for i in range(n):
        for j in range(0, n - 1 - i):
            if matriz[j][1] > matriz[j + 1][1]:
                matriz[j], matriz[j + 1] = matriz[j + 1], matriz[j]
```

Función de algoritmo de Búsqueda Binaria:

```python
# Búsqueda binaria
def busqueda_binaria(matriz, valor, columna):
    inicio = 0
    fin = len(matriz) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        actual = matriz[medio][columna]
        if str(actual).lower() == str(valor).lower():
            return medio
        elif str(actual).lower() < str(valor).lower():
            inicio = medio + 1
        else:
            fin = medio - 1
    return -1
```

Metodología Utilizada

La elaboración del trabajo se realizó en las siguientes etapas:

- Recolección de información teórica en documentación confiable.
- Implementación en Python de los algoritmos estudiados.
- Pruebas con diferentes conjuntos de datos.
- Registro de resultados y validación de funcionalidad.
- Elaboración de este informe y preparación de anexos.

Resultados Obtenidos

- El programa ordenó correctamente la lista de números ingresada.
- La búsqueda binaria localizó de forma eficiente el número especificado.
- Se comprendieron las diferencias en complejidad entre los algoritmos estudiados.
- Se valoró la importancia de tener los datos ordenados para aplicar búsqueda binaria.

Conclusiones

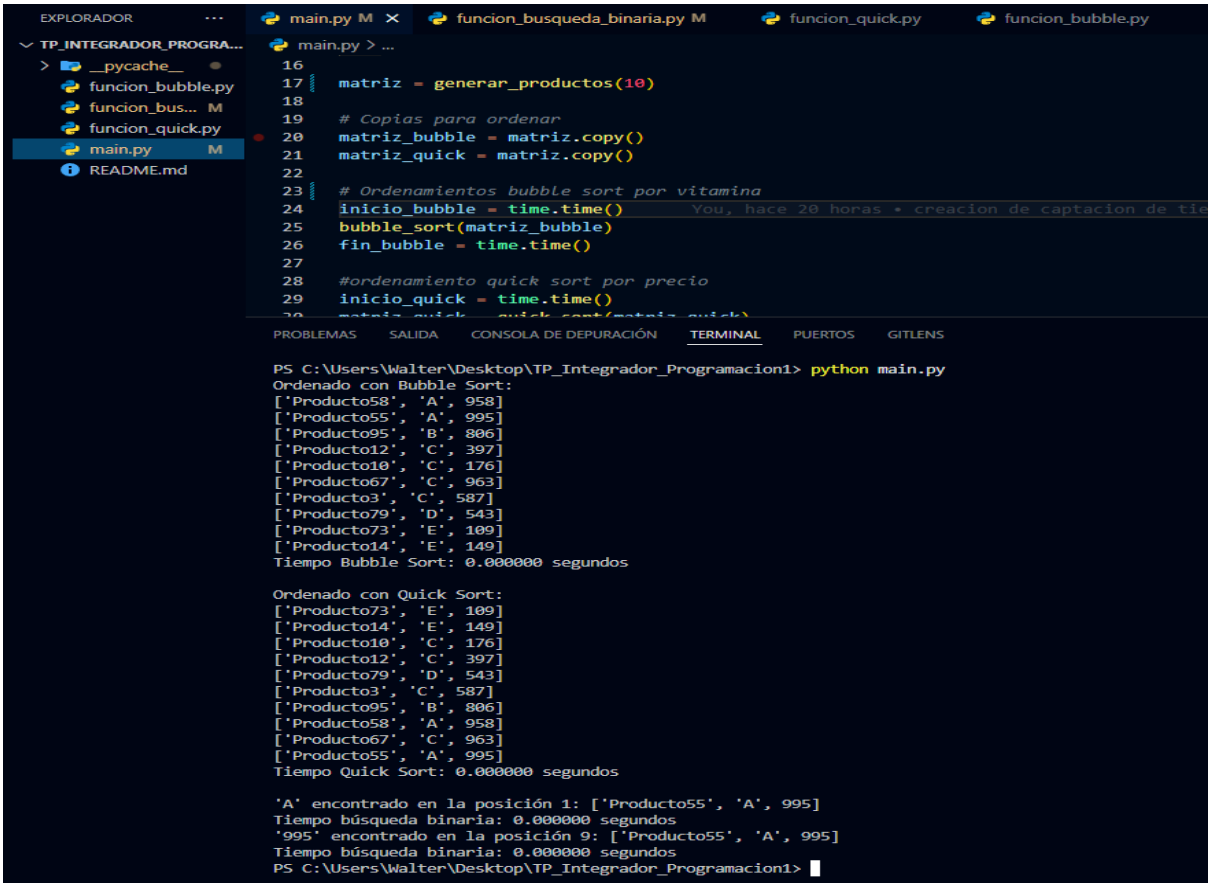
Los algoritmos de búsqueda y ordenamiento son herramientas esenciales en la programación, ya que permiten optimizar el acceso y la organización de los datos. A lo largo del trabajo, se ha demostrado como su correcta aplicación puede mejorar significativamente el rendimiento de los programas. En particular, la búsqueda binaria destaca por su eficiencia cuando se trabaja con estructuras ordenadas, lo que refuerza la importancia de elegir el algoritmo adecuado según el contexto. Comprender estos conceptos no solo enriquece el conocimiento técnico, sino que también contribuye al desarrollo de soluciones informáticas más sólidas y eficaces.

Bibliografía

1. Algoritmo de ordenamiento - Wikipedia, la enciclopedia libre
2. Bubble Sort - Python - GeeksforGeeks
3. QuickSort - Python - GeeksforGeeks
4. Árbol de búsqueda binaria - GeeksforGeeks
5. Pdf utilizados a lo largo de la materia.

Anexos

1. Link de repositorio con codigo fuente del proyecto: walter404/TP_Integrador_Programacion1
2. Video explicativo del proyecto: TRABAJO INTEGRADOR TUP PROGRAMACION 1


EXPLORADOR
TP_INTEGRADOR_PROGRA...
__pycache__
funcion_bubble.py
funcion_bus... M
funcion_quick.py
main.py M
README.md
main.py M
funcion_busqueda_binaria.py M
funcion_quick.py
funcion_bubble.py
main.py > ...
16
17 matriz = generar_productos(10)
18
19 # Copias para ordenar
20 matriz_bubble = matriz.copy()
21 matriz_quick = matriz.copy()
22
23 # Ordenamientos bubble sort por vitamina
24 inicio_bubble = time.time()
You, hace 20 horas • creacion de captacion de tie
25 bubble_sort(matriz_bubble)
26 fin_bubble = time.time()
27
28 #ordenamiento quick sort por precio
29 inicio_quick = time.time()
PROBLEMAS
SALIDA
CONSOLA DE DEPURACIÓN
TERMINAL
PUERTOS
GITLENS
PS C:\Users\Walter\Desktop\TP_Integrador_Programacion1> python main.py
Ordenado con Bubble Sort:
['Producto58', 'A', 958]
['Producto55', 'A', 995]
['Producto95', 'B', 806]
['Producto12', 'C', 397]
['Producto10', 'C', 176]
['Producto67', 'C', 963]
['Producto3', 'C', 587]
['Producto79', 'D', 543]
['Producto73', 'E', 109]
['Producto14', 'E', 149]
Tiempo Bubble Sort: 0.000000 segundos
Ordenado con Quick Sort:
['Producto73', 'E', 109]
['Producto14', 'E', 149]
['Producto10', 'C', 176]
['Producto12', 'C', 397]
['Producto79', 'D', 543]
['Producto3', 'C', 587]
['Producto95', 'B', 806]
['Producto58', 'A', 958]
['Producto67', 'C', 963]
['Producto55', 'A', 995]
Tiempo Quick Sort: 0.000000 segundos
'A' encontrado en la posición 1: ['Producto55', 'A', 995]
Tiempo búsqueda binaria: 0.000000 segundos
'995' encontrado en la posición 9: ['Producto55', 'A', 995]
Tiempo búsqueda binaria: 0.000000 segundos
PS C:\Users\Walter\Desktop\TP_Integrador_Programacion1>