Análisis de Algoritmos

IC-3002

Tarea 01

Implementación y Análisis de algoritmos

Profesor:

Ing. Joss Pecou Jhonson

Integrantes:

Yeremi Calvo Porras

Heldyis Agüero Espinoza

Derrick Gonzales Silva

Fecha de entrega:

27 / 08 / 2025

II Semestre 2025

Indice

Algoritmos de ordenamiento

Descripción del algoritmo

Para ordenar la matriz usamos QuickSort, porque es rápido y funciona bien con listas grandes. La idea es agarrar un número de la lista (nosotros usamos el del medio), y con eso separar la lista en tres:

- Los que son menores,
- Los iguales,
- Los mayores.

Después se vuelve a hacer lo mismo en las sublistas hasta que ya no se pueda dividir más. Al final se juntan todas las partes y queda la lista ordenada.
Como lo que tenemos es una lista de listas, aplicamos QuickSort a cada sublista por aparte, así cada una queda en orden.

Para comparar, usamos MergeSort, que también divide la lista en mitades hasta que queda una sola posición, y luego va mezclando las partes ordenadas de manera que queden todas en orden. Es un poco más predecible que QuickSort, y aunque también tiene que hacer varias operaciones, su peor caso sigue siendo O(n log n), así que es más estable en rendimiento.

Análisis de complejidad

QuickSort:

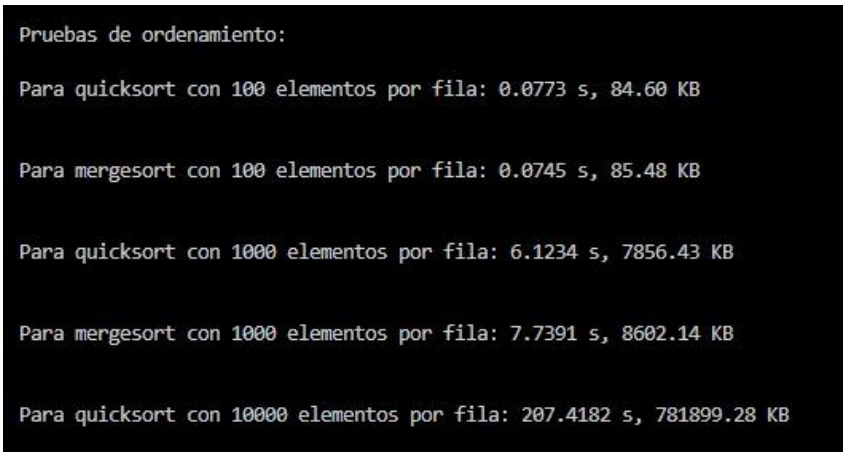
- Mejor caso: cuando siempre se parte la lista en mitades casi iguales → O(n log n)
- Caso promedio: también O(n log n)
- Peor caso: cuando el pivote no ayuda (por ejemplo, si la lista ya viene ordenada y siempre agarra un extremo) → $O(n^2)$
- Función de complejidad:

$T(n)=nlog⁡nT(n) = n \log n$T(n)=nlogn

MergeSort:

- Mejor caso, caso promedio y peor caso: siempre divide la lista a la mitad y mezcla → O(n log n)
- Función de complejidad:
- $T(n)=nlog⁡nT(n) = n \log n$T(n)=nlogn

Resultados



Comparación de eficiencia entre algoritmos

Para listas pequeñas (100 elementos), ambos algoritmos funcionan muy rápido y casi igual en memoria.

Para listas grandes (1000 elementos), QuickSort es un poco más rápido que MergeSort, pero MergeSort utiliza algo más de memoria debido a cómo guarda las sublistas mientras las mezcla.

En general:

QuickSort puede ser más rápido en promedio, pero su peor caso puede ser lento si el pivote no divide bien.

MergeSort es más estable y predecible, siempre O(n log n), aunque consume un poco más de memoria.

En conclusión, ambos algoritmos ordenan correctamente todas las listas dentro de la matriz, y la elección depende de si se prioriza velocidad (QuickSort) o estabilidad y seguridad del peor caso (MergeSort).

Algoritmos de búsqueda

Descripción del algoritmo

Para buscar dentro de la matriz usamos Búsqueda Binaria. Este funciona siempre y cuando la lista esté ordenada. Lo que hace es:

Agarra el elemento del medio y lo compara con el que estamos buscando.

Si es igual, ya lo encontró.

Si el valor buscado es menor, entonces busca en la parte izquierda.

Si es mayor, busca en la parte derecha.

En la matriz lo que hicimos fue correr la búsqueda binaria en cada sublista hasta encontrar el valor o llegar al final sin hallarlo.

También hicimos la Búsqueda Lineal como comparación. Esa es la más sencilla: recorre todos los elementos uno por uno y revisa si alguno es el que buscamos. Es lenta, pero siempre funciona, aunque la lista no esté ordenada**.**

Análisis de complejidad

Búsqueda Binaria:

Mejor caso: si el valor está justo en el medio desde el inicio (O(1)).

Caso promedio y peor caso: cada vez divide la lista a la mitad, así que termina siendo O(log n).

La función es:

$$T(n)=\log_{f0} nT(n) = \log nT(n)=\log n$$

Búsqueda Lineal:

Mejor caso: si lo encuentra en la primera posición (O(1)).

Peor caso: si el valor está al final o no está, tiene que revisar todo (O(n)).

Caso promedio: más o menos la mitad de la lista, entonces también anda en O(n).

La función es:

$$T(n)=nT(n) = nT(n)=n$$

Resultados

```
Pruebas de búsqueda:
Para busquedaMatriz en 10x10: 0.000130 s, 0.22 KB, ¿Existe?=False

Para busquedaLineal en 10x10: 0.000037 s, 0.09 KB, ¿Existe?=False

Para busquedaMatriz en 100x100: 0.001168 s, 0.80 KB, ¿Existe?=False

Para busquedaLineal en 100x100: 0.000678 s, 0.09 KB, ¿Existe?=True

Para busquedaMatriz en 500x500: 0.002452 s, 3.93 KB, ¿Existe?=True

Para busquedaLineal en 500x500: 0.010601 s, 0.09 KB, ¿Existe?=True
```

Comparación de eficiencia con otro algoritmo.

Comparando los dos algoritmos, la búsqueda lineal va como piña para matrices chicas (tipo 10x10 o 100x100) porque es super rápida y gasta poca memoria. Pero cuando la matriz se pone grande (500x500 para arriba), la búsqueda matricial le pasa el trapo porque está optimizada para trabajar con matrices. Eso sí, la matricial gasta más memoria.

Link de acceso al Github del trabajo

https://github.com/HeldyisAE/Tarea-01---Analisis-de-Algoritmos---2025.git