TEC | Tecnológico de Costa Rica

Tarea Corta #1

IC-3002 - Análisis de Algoritmos

Implementación y Análisis de Algoritmos

Estudiantes:

Tayler Wynta Rodriguez

Maikel Flores Navarro

Axel Lopez Cruz

Profesor:

Joss Pecou Johnson

II Semestre, 2025

Parte 1: Implementación y Análisis de Ordenamiento

Objetivo: Implementar y analizar el algoritmo de ordenamiento que se aplique a una estructura de lista de listas presentando cada lista ordenada en el resultado.

1. Descripción del algoritmo.

El algoritmo seleccionado para este propósito es Quicksort. Este es uno de los algoritmos de ordenamiento más utilizados por su eficiencia práctica y relativa simplicidad de implementación.

El funcionamiento de Quicksort se basa en la estrategia de **“**divide y vencerás”:

1. Se selecciona un pivote (generalmente el elemento central o uno aleatorio de la lista).
2. Se dividen los elementos en tres sublistas:
    - Los menores que el pivote.
    - Los Iguales que el pivote
    - Los mayores que el pivote.
3. Se aplica recursivamente el mismo proceso sobre cada sublista.
4. Finalmente, se unen los resultados de forma ordenada: primero los menores, luego el pivote y por último los mayores.

En el caso de una lista de listas (matriz), se aplica Quicksort a cada fila, haciendo que cada sublista quede ordenada individualmente. Al final, se obtiene una matriz donde cada fila tiene sus elementos en orden creciente.

2. Análisis de complejidad.

El análisis de complejidad del algoritmo Quicksort considera tres escenarios:

- **Mejor caso**: Ocurre cuando el pivote divide la lista en partes de tamaño relativamente equilibrado.
- **Caso promedio**: En la mayoría de las ejecuciones, aunque las divisiones no sean perfectas, el algoritmo mantiene un buen funcionamiento.
- **Peor caso**: Ocurre cuando los pivotes elegidos son siempre los más grandes o los más pequeños, lo que genera divisiones desequilibradas. Este escenario es poco común, pero puede presentarse si la lista ya se encuentra ordenada o casi ordenada y no se aplican técnicas de pivoteo aleatorio.

3. Función T(n).

Complejidad	**T(n)**	**Grado**
Omega Ω (mejor caso)	nlogn	Lineal-log
Theta Θ (promedio)	nlogn	Lineal-log
Big O (peor caso)	n^2	Cuadrática

4. Resultados.

Las pruebas realizadas con el algoritmo Quicksort sobre matrices de distintos tamaños muestran que su comportamiento es coherente con lo esperado teóricamente. En estructuras pequeñas, el tiempo de ejecución es prácticamente inmediato, mientras que en matrices más

grandes el tiempo aumenta de manera progresiva, manteniéndose dentro de márgenes adecuados para el volumen de datos procesados.

El consumo de memoria también refleja un crecimiento proporcional al tamaño de la matriz, incrementándose de manera gradual desde unos pocos kilobytes en pruebas iniciales hasta valores más altos en estructuras de mayor escala. Este patrón confirma que el uso de recursos está directamente asociado al número de elementos y al proceso de partición interna del algoritmo.

En conclusión, los resultados experimentales respaldan que Quicksort es un algoritmo eficiente para el ordenamiento de listas de listas, logrando un buen equilibrio entre tiempo de ejecución y consumo de memoria, incluso en escenarios con grandes cantidades de datos.

5. Comparación de eficiencia con otro algoritmo.

Quicksort

Algotimo Quicksort(Desarrollado por nosotros).		
Tamaño de la Matriz	Duracion de la Prueba (Seg).	Memoria Consumida (KB).
10x10	0.000326	2.53
32x32	0.002794	11.60
100x100	0.028077	84.93
317x317	0.205555	801.25
1000x10000	1.088346	7872.73

BubbleSort

Algotimo Bubble Sort		
Tamaño de la Matriz	Duracion de la Prueba (Seg).	Memoria Consumida (KB).
10x10	0.000127	0,14
32x32	0.001336	0.14
100x100	0.030053	0.14
317x317	1.551662	0.29
1000x1000	319.319918	0.32

Las pruebas realizadas muestran que ambos algoritmos mantienen un comportamiento coherente con el aumento del tamaño de la matriz. En el caso de Quicksort, los tiempos de ejecución se mantienen bajos en matrices pequeñas y crecen de manera progresiva a medida que la cantidad de datos aumenta, lo que evidencia su capacidad para manejar estructuras de mayor tamaño con un costo de memoria proporcionalmente mayor.

Por otro lado, Bubble Sort presenta una escalada de tiempo mucho más notable conforme crece el tamaño de la matriz, alcanzando valores elevados en estructuras grandes, mientras que el consumo de memoria se mantiene prácticamente constante y bajo en todas las pruebas realizadas.

En conclusión, los resultados muestran cómo el tiempo de ejecución y la memoria utilizada por cada algoritmo responden según el tamaño de los datos de entrada, confirmando su robustez con las características teóricas de cada técnica de ordenamiento.

Parte 2: Implementación y análisis de búsqueda

Objetivo: Implementar y analizar un algoritmo de búsqueda que se pueda aplicar en una estructura de tipo lista de listas.

1. Descripción del algoritmo.

Se utiliza búsqueda secuencial (lineal) sobre la matriz: se recorre fila por fila y, dentro de cada fila, elemento por elemento, retornando True al primer hallazgo y False si se termina el recorrido sin coincidencias.

2. Análisis de complejidad.

- **Mejor caso**: Encuentra el valor en la primera comparación.
- **Caso promedio:** En promedio, se inspecciona aprox. la mitad de los N elementos.
- **Peor caso:** No está el valor (o está al final), se revisan todos los elementos.

3. Función T(n).

Complejidad	**T(n)**	**Grado**
Omega Ω (mejor caso)	1	Constante
Theta Θ (promedio)	N	Lineal
Big O (peor caso)	N	Lineal

4. Resultados.

Los resultados obtenidos muestran que la búsqueda lineal presenta un desempeño coherente con lo esperado: su tiempo de ejecución crece de manera proporcional al número de elementos a recorrer, siendo eficiente únicamente en estructuras pequeñas o cuando el valor buscado se encuentra al inicio. A medida que el volumen de datos aumenta, el proceso se vuelve más costoso en términos de tiempo, ya que requiere comparar cada elemento.

En conclusión, la búsqueda lineal es un algoritmo sencillo, de bajo requerimiento de memoria y fácil implementación, pero su eficiencia disminuye en grandes cantidades de información, donde otras estrategias de búsqueda pueden resultar más adecuadas.

5. Comparación de eficiencia con otro algoritmo.

Búsqueda lineal

Algoritmo Búsqueda Lineal (Desarrollado por nosotros).		
Tamaño Matriz	Duración (seg.)	Memoria Usada(kb)
10*10	0.006410 segundos	10578 KB
100*100	0.004410 segundos	10120 KB
500*500	0.009006 segundos	10120 KB
1000*1000	0.017757 segundos	10120 KB
3000*3000	0.019591 segundos	10120 KB

Búsqueda binaria

Algoritmo Búsqueda Binaria		
Tamaño Matriz	Duración (seg.)	Memoria Usada(kb)
10*10	0.005096 segundos	11026 KB
100*100	0.005850 segundos	127071 KB
500*500	0.042643 segundos	3096079 KB
1000*1000	0.164588 segundos	12615175 KB
3000*3000	1.383342 segundos	107591748 KB

Las pruebas realizadas muestran un comportamiento coherente con la según cada técnica de búsqueda. En el caso de la búsqueda lineal, los tiempos de ejecución se mantienen bajos y relativamente estables en estructuras pequeñas y medianas, con un consumo de memoria prácticamente constante. Esto indica que el algoritmo únicamente recorre los elementos de la matriz sin requerir estructuras auxiliares ni procesos adicionales.

Por otro lado, la búsqueda binaria tiene un patrón distinto: aunque logra resolver rápidamente en casos iniciales y mantiene su lógica de eficiencia, los resultados experimentales muestran que el consumo de memoria aumenta de manera considerable conforme crece el tamaño

de la matriz. Este comportamiento muestra que, si bien la búsqueda binaria es más eficiente en listas ordenadas, en la práctica su desempeño depende también de las condiciones de implementación y del manejo de estructuras en memoria.

Enlace Github: MaikelFn/Tarea-Corta-1: Repositorio para la Tarea Corta #1 del curso análisis de algoritmos.