UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico – Programación I

Análisis de algoritmos: eficiencia y optimización

Alumnas:

Giselle Chaumont Mohr - gisellechaumont@gmail.com

Daiana Ghisio - daianaghisio@gmail.com

Materia: Programación I

Profesor: Bruselario, Sebastián

Fecha de Entrega: 09 de junio de 2025

1. Marco teórico

La eficiencia algorítmica se refiere al uso que hace un algoritmo de los recursos disponibles, principalmente el **tiempo de ejecución** y la **memoria**. Analizar la eficiencia permite estimar cuán bien resuelve un problema en términos de velocidad o consumo de recursos. Algunas medidas no siempre pueden ser comparadas directamente, entonces según el objetivo, se puede priorizar que el algoritmo sea más rápido (menor complejidad temporal) o que utilice menos memoria (menor complejidad espacial). Esto depende de la medida de eficiencia que se esté considerando como prioridad, por ejemplo, la prioridad podría ser obtener la salida del algoritmo lo más rápido posible, o que minimice el uso de la memoria, o alguna otra medida particular.

1.1 Complejidad temporal

Se refiere a la cantidad de tiempo que tarda un algoritmo en ejecutarse, generalmente expresado en función del tamaño de la entrada (n).

Para lograr una optimización de la complejidad temporal, se suelen aplicar estrategias como:

- Reducir bucles anidados
- Aplicar técnicas de divide y vencerás (El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar)
- Elegir algoritmos con mejor rendimiento asintótico
  Un ejemplo de este último caso sería usar Merge Sort en lugar de Bubble Sort. La explicación es la siguiente:

  Bubble Sort recorre la lista muchas veces comparando elementos que están uno al lado del otro. Si están desordenados, los intercambia. Esto se repite hasta que toda la lista queda ordenada. El problema es que, incluso si una parte ya está ordenada, igual sigue comparando y haciendo swaps, lo que conlleva un mayor uso de tiempo y hace que no sea muy eficiente.

  Merge Sort funciona distinto: primero divide la lista en mitades más chicas, las sigue dividiendo hasta que quedan partes muy pequeñas (de uno o dos elementos), y después las vuelve a unir en orden. Como ordena en el momento de combinar las partes, realiza una cantidad menor de comparaciones, y eso hace que funcione mucho más eficientemente, sobre todo cuando la lista tiene muchos elementos.

  A esto se lo llama eficiencia asintótica porque, si lo analizáramos matemáticamente, podríamos ver que la cantidad de operaciones crece de manera distinta según el algoritmo, y eso permite comparar cuál escala mejor frente a entradas más grandes.

1.2 Complejidad espacial

Se refiere a la cantidad de memoria que se requiere para la ejecución.

Para optimizar la complejidad espacial, se busca reducir la cantidad de memoria que el algoritmo necesita durante su ejecución. Algunas estrategias comunes incluyen:

- reutilizar variables en lugar de crear estructuras nuevas -
- evitar copias innecesarias de datos
- preferir estructuras de datos más livianas.

  Por ejemplo, al recorrer una lista, usar un bucle que procese los elementos uno por uno es más eficiente en memoria que generar una nueva lista con los resultados si no es necesario.

  También, para continuar con los ejemplos de algoritmos que estamos analizando previamente, Quick Sort, tiene buen rendimiento en tiempo y además utiliza menos memoria que otros como Merge Sort, y esto se debe a que no requiere estructuras auxiliares para combinar los resultados.

2. Caso práctico

Link al repositorio en github: https://github.com/daianaghisio/TP-Programacion-I

Pruebas realizadas:

```
n.exe "c:/Users/Daiz_/Desktop/TP Programacio
Tiempo Bubble Sort: 0.038430 segundos
Tiempo Merge Sort:  0.001611 segundos
PS C:\Users\Daiz_\Desktop\TP Programacion I>
n.exe "c:/Users/Daiz_/Desktop/TP Programacio
Tiempo Bubble Sort: 0.037558 segundos
Tiempo Merge Sort:  0.001312 segundos
PS C:\Users\Daiz_\Desktop\TP Programacion I>
n.exe "c:/Users/Daiz_/Desktop/TP Programacio
Tiempo Bubble Sort: 0.036683 segundos
Tiempo Merge Sort:  0.001427 segundos
```

3. Conclusión

Este trabajo nos permitió comparar el rendimiento de dos algoritmos de ordenamiento y comprender mejor los distintos tipos de eficiencia. A través del caso práctico, pudimos comprobar que Merge Sort, aunque más complejo, resulta mucho más eficiente que Bubble Sort al trabajar con listas grandes, tal como sugería nuestra investigación teórica.

4. Referencias

- Wikipedia. (s. f.). *Algoritmo divide y vencerás*. Recuperado de https://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s
- GeeksforGeeks. (2025, 25 de abril). *Merge Sort - Data Structure and Algorithms Tutorials*. Recuperado de https://www.geeksforgeeks.org/merge-sort/
- GeeksforGeeks. (2025, 21 de enero). *Bubble Sort Algorithm*. Recuperado de https://www.geeksforgeeks.org/bubble-sort-algorithm/
- Python Software Foundation. (s. f.). *time — Time access and conversions (Python 3.13.4 documentation)*. Recuperado de https://docs.python.org/3/library/time.html