Tecnológico de Monterrey

Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales

Programación de Estructuras de Datos y Algoritmos Fundamentales

Instituto Tecnológico y de Estudios Superiores de Monterrey

Profesor: Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega: 08 de julio de 2023

Estudiante: Alyson Melissa Sánchez Serratos - A01771843

Actividad Integral: investigación y reflexión

En la era actual, la creación de código eficiente ha adquirido una importancia crítica en el campo del desarrollo de software. En un entorno en el que la tecnología desempeña un papel central en múltiples aspectos de la sociedad, la eficiencia en la programación se convierte en un factor determinante que marca la diferencia entre el éxito y el fracaso de una aplicación o sistema. Este aspecto es especialmente relevante considerando que millones de usuarios, incluyendo empresas, gobiernos, instituciones financieras, hospitales y otros, dependen de estos sistemas para llevar a cabo sus operaciones diarias. Un código eficiente permite un rendimiento óptimo, una mejor experiencia de usuario y la capacidad de aprovechar al máximo los recursos disponibles. Por otro lado, un código ineficiente puede generar retrasos, errores y costos innecesarios. En este contexto, la eficiencia en la programación se convierte en un factor imprescindible para el éxito y la competitividad en el ámbito tecnológico actual.

En situaciones problema en las que se enfrenta la necesidad de ordenar o buscar información en una base de datos extensa, es crucial que el programador cuente con el conocimiento, las herramientas y la habilidad necesarios para implementar soluciones eficientes. Para lograrlo, resulta fundamental realizar una elección adecuada de algoritmos eficaces, cuya complejidad temporal sea baja, con el fin de asegurar un procesamiento rápido y eficiente de la información.

Según han señalado Naeem Akhter, Muhammad Idrees, y Furqan-ur-Rehman (2016) la clasificación es crucial para optimizar la utilidad de los datos. En la vida diaria, se encuentran numerosos ejemplos de clasificación, como la búsqueda de artículos en una tienda o centro comercial, donde se almacenan en categorías específicas. Asimismo, una palabra en un diccionario resulta sencillo debido al ordenamiento alfabético. De manera similar, localizar un número de teléfono, nombre o dirección en una guía telefónica es fácil gracias a la clasificación. En este sentido, en informática, la clasificación es una operación fundamental y de gran importancia debido a sus múltiples aplicaciones.

En el contexto de la primera fase de la actividad integral llevada a cabo en el marco de la asignatura de Programación de Estructuras de Datos y Algoritmos Fundamentales, se procedió a la implementación de un total de dos algoritmos de ordenamiento: el algoritmo de Burbuja (Burbuja Sort) y el algoritmo de Mezcla (Merge Sort). El objetivo de esta implementación radicó en la evaluación y comparación de su eficiencia y complejidad temporal. Es importante resaltar que el proceso de ordenamiento se realizó utilizando una base de datos que constaba de 16,807 líneas, por lo cual el tamaño de la entrada fue grande.

De acuerdo con Naeem Akhter, Muhammad Idrees, y Furqan-ur-Rehman (2016):

> Bubble Sort no es una mala opción cuando el tamaño de entrada es pequeño (menos de 100) porque es un algoritmo fácil de implementar. Pero a medida que aumenta el tamaño de entrada, se requiere más eficiencia para la clasificación. La ordenación de burbuja es fácilmente dominada por algoritmos más eficientes en un tamaño de entrada más grande. (p.935)

A su vez, Yomna Ben Jmaa, Rabie Ben Atitallah, David Duvivier y Maher Ben Jemaa (2019) comentan que Bubble Sort es ineficiente para clasificar un gra número de elementos porque su complejidad es muy importante O (n^2) en la media y en el peor caso.

Por otro lado, en relación con el segundo algoritmo de ordenamiento implementado, según Naeem Akhter, Muhammad Idrees, y Furqan-ur-Rehman (2016):

> El algoritmo Merge Sort es más complejo que Bubble Sort. Tiene un rendimiento deficiente en entradas pequeñas, pero mejora su rendimiento en paralelo con el incremento en el tamaño de entrada. (p.935)

Tras haber observado los análisis realizados por medio de un estudio comparativo realizado por el *International Journal of Computer Science and Information Security*, se procederá a comprobar lo citado a través del número de comparaciones e intercambios realizados por ambos algoritmos. En la **tabla 1** se observan los valores.

Tabla 1 Comparación de eficacia de algoritmos de comparación

	Bubble Sort	Merge Sort
Comparaciones	141229221	214693
Intercambios	70166913	**No aplica**

Como se puede apreciar, el algoritmo de ordenamiento Bubble Sort exhibió un número de comparaciones significativamente mayor en comparación con el algoritmo Merge Sort. Además, su tiempo de ejecución fue notablemente más prolongado y evidente. De esta manera, se confirma que, en el contexto de esta situación específica, Merge Sort se mostró superior en términos de complejidad temporal.

A continuación en la **tabla 2** se muestra un resumen de la complejidad de los algoritmos implementados de acuerdo con su mejor, peor y caso promedio.

Tabla 2 Complejidad de algoritmos de ordenamiento por mejor, peor y caso promedio. Fuente: Khan, M., Shaheen, S., & Qureshi, F. A (2014)

Algorithms	Best Case			Average Case			Worst Case		
	Small Data Set()	Average Data Set	Large Data Set	Small Data Set()	Average Data Set	Large Data Set	Small Data Set()	Average Data Set	Large Data Set
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)	O(nlogn)

Se aprecia que la complejidad temporal promedio, así como para su mejor y peor caso en Merge Sort es O (n log n), esto significa que el algoritmo se comporta de manera consistente y predecible en términos de tiempo de ejecución, independientemente de la disposición de los elementos en la lista a ordenar.

Por otra parte, en el peor, mejor y promedio de los casos, la complejidad temporal de Bubble Sort es $O(n^2)$. Esto implica que el tiempo de ejecución del algoritmo aumenta de manera cuadrática con el tamaño de los datos de entrada. Asimismo, su desempeño no mejora incluso en casos promedio o mejores escenarios. Esto se puede deber a su estrategia de comparar y hacer intercambios adyacentes de elementos repetidamente a lo largo de la lista. Esto implica que incluso cuando los elementos ya están en su posición correcta, el algoritmo seguirá realizando comparaciones y permutaciones innecesarias, lo que conduce a un mayor tiempo de ejecución.

Adicional a esto, el programa realizado durante la primera fase de la actividad integradora también implementó un algoritmo de búsqueda binario con la finalidad de verificar la existencia de fechas en la bitácora ordenada utilizada. Dicho algoritmo fue utilizado ya que es considerado como uno de los más eficientes para buscar elementos en una lista ordenada. Se utiliza ampliamente debido a su rapidez y efectividad en la búsqueda de datos en conjuntos de gran tamaño. Asimismo, una de las principales razones por las que el algoritmo de búsqueda binaria fue bueno para la situación problema se debe a su complejidad temporal. Su complejidad es logarítmica, es decir, O (log n). Esto significa que a medida que aumenta el tamaño de la entrada, el tiempo requerido para buscar un elemento en ella crece de manera mucho más lenta que en algoritmos con complejidades lineales o cuadráticas. Esta propiedad fue un factor clave en la elección del algoritmo, ya que se pudo anticipar que el tiempo de ejecución del programa no se vería significativamente afectado por el aumento del tamaño de entrada. A su vez, la búsqueda binaria se puede aplicar en diferentes contextos y problemas en los que se requiera encontrar un elemento específico en una entrada ordenada. Esto incluye la búsqueda de un número en un rango, implementación la cual se utilizó en el programa desarrollado.

Cabe recalcar que no existe un algoritmo de ordenamiento o de búsqueda inherentemente superior a otro en todas las situaciones. La elección del algoritmo adecuado depende del contexto específico y de los requisitos del problema en cuestión. Cada algoritmo de ordenamiento tiene sus propias fortalezas y debilidades, y su eficacia puede variar según las características de los datos y las restricciones del sistema.

Es fundamental considerar factores como el tamaño de los datos, la naturaleza de los elementos a ordenar y las limitaciones de tiempo y recursos disponibles al seleccionar el algoritmo más apropiado. Al evaluar y comparar diferentes algoritmos en relación con una situación particular, se debe determinar cuál es el más eficiente y adecuado para lograr los objetivos específicos del problema.

Por lo tanto, se llega a la conclusión de que es crucial que el programador realice un análisis detallado del problema al que se enfrenta y considere cuidadosamente qué solución es digna de implementar. No se debe tomar decisiones basadas únicamente en la eficiencia temporal. Es importante encontrar un equilibrio entre la eficiencia y otros aspectos relevantes para lograr una solución óptima en general.

Fuentes de información:

Ben Jmaa, Y., Ben Atitallah, R., Duvivier, D., & Ben Jemaa, M. (2019). A comparative study of sorting algorithms with FPGA acceleration by High Level synthesis. Computación y Sistemas, 23(1), 213–230. https://doi.org/10.13053/cys-23-1-2999

Furqan-ur-Rehman, N. A. M. (2016). *Sorting Algorithms – A Comparative Study*. International Journal of Computer Science and Information Security. https://www.researchgate.net/publication/315662067_Sorting_Algorithms_-_A_Comparative_Study

Khan, M., Shaheen, S., & Qureshi, F. A. (2014). Comparative analysis of five sorting algorithms on the basis of Best Case, Average Case, and worst case. Iteejournal.org. http://www.iteejournal.org/archive/vol3no1/v3n1_1.pdf