Reflexión Actividad 1.3

Programación de estructuras de datos y algoritmos fundamentales (TC1031.570)
Fausto Alejandro Palma Cervantes (A01639224)
11 de julio de 2022

Dentro de la Ingeniería en Tecnologías Computacionales, los algoritmos de ordenamiento y búsqueda son de gran utilidad debido a que facilitan la interacción e interpretación de conjuntos de datos. La importancia de uso de estos algoritmos se ve especialmente en conjuntos de información a gran escala, ya que es ahí cuando las capacidades de la computadora que se usa para su análisis se convierte rápidamente en un factor importante. La capacidad de una computadora en conjunto con la eficacia temporal y espacial de sus algoritmos son cuestiones relevantes para un programador al momento de escoger los métodos con los cuales analizar información. (UtkarshPandey6, 2022) Es considerando todo lo anterior por lo cual hoy en día se utiliza la notación *Big-O* para analizar la complejidad computacional de un algoritmo. Este tipo de notación lleva a cabo un análisis algorítmico utilizando el siguiente procedimiento:

1. Determinar el tamaño del conjunto de datos de entrada
2. Determinar el máximo número de operaciones que se lleva a cabo con dicho conjunto
3. Identificar las operaciones de mayor valor
4. Eliminar las operaciones de menor valor y otras constantes

El resultado de este análisis es la determinación de la operación de mayor valor (operación básica) del algoritmo, y es esta complejidad algorítmica la cual determina su complejidad computacional. (SoumyadeepDebnath, 2022) Una vez teniendo esa complejidad computacional, el programador ahora puede tomar una decisión informada al momento de seleccionar el algoritmo correcto para sus necesidades. Además, esta complejidad permite comparar todos los algoritmos entre sí para determinar su eficacia dentro de un programa. Las comparaciones entre algoritmos se hacen evaluando sus límites hacia infinito tal como se haría con cualquier otra función matemática. La siguiente figura muestra la relación entre las complejidades más comunes en algoritmos de ordenamiento y búsqueda y el tamaño del conjunto de información al que interpreta:

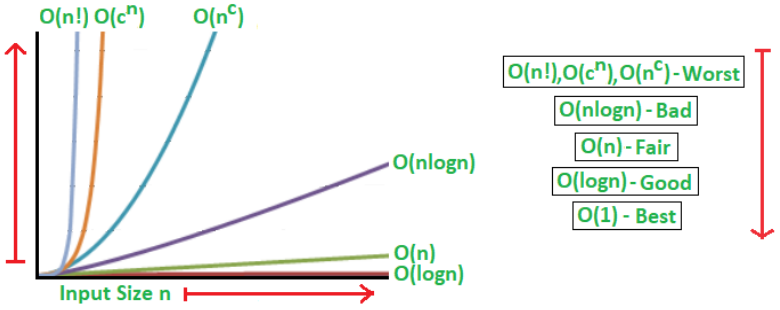

Figura 1. Comportamiento de complejidades computacionales en notación *Big-O*
(SoumyadeepDebnath, 2022)

Justo como se mencionó previamente, existen dos tipos de algoritmos que pueden utilizarse para el análisis de conjuntos: los de búsqueda y los de ordenamiento. El primer tipo de algoritmos, tal y como lo sugiere su nombre, se utilizan para hacer búsquedas de datos dentro de conjuntos de información. (*Searching Algorithms*, 2022) Por otra parte, los algoritmos de ordenamiento se utilizan para organizar datos dentro del mismo tipo de conjuntos en alguna secuencia lógica determinada. Debido a su naturaleza, la complejidad espacial y la estabilidad del algoritmo (su capacidad para preservar el orden de datos lógicamente iguales dentro de su secuencia) son otras propiedades de este tipo de algoritmos además de la complejidad temporal. (*Sorting Algorithms*, 2022)

A pesar de las diferencias entre tipos de algoritmos, cada uno tiene sus diferentes complejidades computacionales las cuales inclusive varían dependiendo de las condiciones de entrada de su conjunto de información. Las siguientes dos tablas muestran las complejidades computacionales de distintos algoritmos importantes divididos por tipo. Cabe mencionar que en ambas tablas se muestran tres distintas complejidades considerando diferentes condiciones de entrada. La columna *Mejor* muestra la complejidad computacional al analizar un conjunto de datos en el orden más eficaz específico al algoritmo (mejor caso). Por otra parte, la columna *Peor* muestra la misma información pero considerando la condición de entrada menos óptima para el algoritmo (peor caso). Por último, la columna *Promedio* muestra la complejidad al analizar un caso promedio, el cual se calcula haciendo un análisis de probabilidad de todos los distintos casos que existen. Cabe mencionar que para la complejidad computacional de un algoritmo se acostumbra usar la complejidad *Peor* ya que suele ser la operación con mayor valor y por ello la más complicada para una computadora.

Algoritmo	**Mejor**	**Promedio**	**Peor**	**Estable**	**Espacio**
Swap sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Sí	O(1)
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	O(1)
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$	Sí	O(1)
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	Sí	O(1)
Merge sort	O(n log n)	O(n log n)	O(n log n)	Sí	O(n)
Quicksort	O(n log n)	O(n log n)	$O(n^2)$	Sí	O(log n)

Tabla 1. Complejidad de algoritmos de ordenamiento
(Rodríguez Tello, E. A., 2022, *sesion05a-AlgoritmosDeBusqueda)*

Algoritmo	**Mejor**	**Promedio**	**Peor**
Búsqueda secuencial	O(1)	O(n)	O(n)
Búsqueda secuencial en arreglos ordenados	O(1)	O(n)	O(n/2)
Búsqueda binaria	O(1)	$O(\log_2 n)$	O(log n)

Tabla 2. Complejidad de algoritmos de búsqueda

(Rodríguez Tello, E. A., 2022, *sesion03a-AnalisisAlgoritmos-Parte02)*

Como parte de la entrega de la Actividad 1.3 (*Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales*) de la unidad de formación *Programación de estructuras de datos y algoritmos fundamentales*, se solicitaron las funcionalidades de organización y búsqueda de registros dentro de una bitácora. En este caso, la organización se llevó a cabo utilizando el algoritmo *Merge Sort* y la búsqueda utilizando búsqueda binaria. *Merge Sort*, el algoritmo de ordenamiento seleccionado, consiste en dividir los datos del conjunto en mitades iguales repetidamente hasta lograr dividir todo el conjunto dato por dato. Una vez teniendo todos los datos individuales, estos se unen siguiendo alguna secuencia lógica determinada (proceso al cual se le conoce como *merge*). (GeeksforGeeks, 2022, *Merge Sort*) Utilizando la información en párrafos anteriores y como se puede apreciar en la primera tabla, además de ser estable *Merge Sort* es el único algoritmo con una complejidad de $O(n \log n)$, mientras que todos los demás algoritmos de la tabla tienen una complejidad de $O(n^2)$. Al consultar la *Figura 1*, podemos determinar que $O(n \log n)$ es la complejidad más simple de todos los algoritmos de ordenamiento en la tabla. Ya que la bitácora está compuesta de 16,807 registros (un conjunto de información grande), se utilizó este algoritmo para su ordenamiento.

Pasando a la funcionalidad de búsqueda, fue el algoritmo *Binary Search*, o búsqueda binaria, el cual se usó para este propósito. Este algoritmo, el cual requiere de un conjunto de datos ordenados, identifica el punto medio del conjunto y compara su valor con el que se está buscando. Si el valor de búsqueda es mayor al del punto medio, se descartan todos los valores menores al punto medio. Si el valor de búsqueda es menor al del punto medio, se descartan todos los valores mayores al punto medio. Lo que hace a este algoritmo tan efectivo es que logra eliminar la mitad de todos los datos de un conjunto en un sólo de sus procesos. Este proceso, similar a lo que ocurre con el de *Merge Sort*, se hace repetidamente hasta encontrar el valor que se está buscando. Al regresar a la segunda tabla se puede ver que este algoritmo tiene una complejidad de $O(\log n)$, mientras que los otros dos algoritmos de la tabla tienen complejidades $O(n/2)$ y $O(n)$. Consultando nuevamente la *Figura 1*, y considerando la complejidad $O(n/2)$ como una variación de $O(n)$, se puede determinar que la búsqueda binaria es la menos compleja de las tres.

A pesar de la misma naturaleza repetitiva en ambos algoritmos, se debe mencionar que el algoritmo *Merge Sort* se implementó a través de una función recursiva (la cual simplifica los procesos y se llama a sí misma repetidamente) mientras que el de la búsqueda binaria se implementó utilizando una función iterativa (la cual hace todos los procesos dentro de sí misma). (*dkp1903*, 2022) Considerando que la situación problema de esta unidad gira en torno a la detección de ataques cibernéticos de redes de bots, tanto el algoritmo de búsqueda como el de ordenamiento son relevantes para dar respuesta a la situación problema. Sólo en esta actividad se logró ordenar y buscar información eficazmente dentro de un conjunto de datos utilizando los algoritmos más computacionalmente simples. Estas funcionalidades son sumamente importantes para la detección de ataques ya que, justo como se menciona en la situación problema, *"una forma de detectar este tipo de situaciones es a través de recopilación de toda la información pertinente que nos permitan hacer búsquedas optimizadas para la detección oportuna de accesos maliciosos a través de redes de robots"*.

Bibliografía:

dkp1903. (2022). *Difference between Recursion and Iteration*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/

GeeksforGeeks et al. (2022). *Linear Search vs Binary Search*. GeeksforGeeks. https://www.geeksforgeeks.org/linear-search-vs-binary-search/.

GeeksforGeeks et al. (2022). *Merge Sort*. GeeksforGeeks. https://www.geeksforgeeks.org/merge-sort/?ref=lbp.

Rodríguez Tello, E. A. . (2022). *sesion03a-AnalisisAlgoritmos-Parte02* [pdf]. Departamento de Computación EIC Región Occidente. Tec de Monterrey.

Rodríguez Tello, E. A. . (2022). *sesion05a-AlgoritmosDeBusqueda* [pdf]. Departamento de Computación EIC Región Occidente. Tec de Monterrey.

Searching Algorithms. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/searching-algorithms/.

Sorting Algorithms. (2022). GeeksforGeeks. https://www.geeksforgeeks.org/sorting-algorithms/.

SoumyadeepDebnath. (2022). *Analysis of Algorithms | Big-O analysis.* GeeksforGeeks. https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/.

UtkarshPandey6. (2022). *Time Complexity and Space Complexity*. GeeksforGeeks. https://www.geeksforgeeks.org/time-complexity-and-space-complexity/.