Actividad 1.3
Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales

Samuel Sánchez García
A00831772

Existen diversos algoritmos de ordenamiento y búsqueda que podemos emplear en nuestras aplicaciones de acuerdo a sus requerimientos. Cada uno cuenta con complejidad y eficiencia diferente, por lo que es de gran importancia tener muy claras las características y funcionalidades que definen nuestro programa.

Hasta este momento, los algoritmos de ordenamiento que más frecuentamos en nuestros ejercicios son Mergesort y Quicksort, ambos centrados en “dividir y conquistar”, y BubbleSort, InsertionSort, SwapSort, etc. En el caso de Mergesort, con complejidad temporal de O(n log n) en su mejor caso, no se hace uso de un elemento pivote, ni necesita hacer intercambios para ordenar el vector. Esto puede significar algunas ventajas sobre Quicksort, por ejemplo, tomando en cuenta una gran cantidad de elementos a ordenar, un ordenamiento por fusión haría el trabajo de una forma más eficiente y es más estable. Por otro lado, Quicksort es un algoritmo muy rápido en su mejor caso y su complejidad es igualmente O(n log n), sin embargo, con una lista de elementos muy larga donde no sabemos qué tan ordenados se encuentran estos, Quicksort puede llegar a alargar el proceso de ordenamiento más de lo necesario. Es cierto que Quicksort es una mejor opción cuando disponemos de poca memoria, pero en esta aplicación el uso de memoria adicional no es un problema, y evitamos que la clasificación sea dependiente de la aleatoriedad del vector.
En nuestro programa implementamos el método de SwapSort para aprovechar su simplicidad y funcionalidad. Y en cuanto a los algoritmos de búsqueda, entre búsqueda lineal y binaria elegimos usar esta última. En el caso de un vector con pocos elementos, ambos algoritmos tienen un rendimiento relativamente igual, no obstante, en el desarrollo de esta aplicación observamos que una búsqueda binaria será mucho más rápida, considerando el funcionamiento y complejidad de este método: divide repetidamente a la mitad la porción de la lista que podría contener al elemento hasta reducir las posibilidades a solo una, y su complejidad en promedio es O(log n).

En la siguiente tabla se muestra un resumen de estas comparaciones de algoritmos.

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)