Reflexión Actividad 1.3

Adrián Becerra Meza A01639813

En esta situación problema nos enfocamos principalmente en usar el Merge Sort, porque intuimos que era el que mejor nos funcionaba en este caso, ya que es uno de los más efectivos de los algoritmos de ordenamiento.

El merge sort en todos sus casos cuenta con una complejidad de O (n log n), este algoritmo se enfoca en dividir por partes el arreglo hasta llegar a arreglos con un único elemento, luego va comparando y decidiendo que valor es mayor que otro, en esta situación problema para determinar que fechas son mayores, las convertimos en números flotantes con lo que el merge sort hizo lo demás.

En esta situación problema no hicimos algoritmos de búsqueda, ya que no nos pedía buscar un solo valor de todas las fechas, sino que determinar un inicio y un final. Donde tuviéramos que demostrar las fechas que se encuentran en esos límites introducidos por el usuario. Sin embargo, los algoritmos de búsqueda también son muy eficientes, dentro de clase utilizamos dos, la búsqueda secuencial y la búsqueda binaria, la secuencial funciona con arreglos desordenados y ordenados, ya que recorre todo el arreglo buscando el valor designado es por esta razón que en el mejor caso tendrá una complejidad de O (1) mientras que en su peor caso tendrá una complejidad de O(n). En cuanto a la búsqueda binaria, tenemos que hacer uso de un arreglo ordenado, ya que vamos a ir revisando las mitades del arreglo hasta encontrar el valor, su mejor caso escenario es O (1) mientras que su peor caso será O (log n).

Otros algoritmos de ordenamiento que vimos en clase fueron selection sort, bubble sort, insertion sort y quick sort. Siendo el quick sort el más eficiente de todos con una complejidad en el mejor de sus casos de O (n log n) y en el peor de sus casos una complejidad de O (n^2). El Selection sort consiste en intercambiar los elementos hasta ordenar el arreglo completo, comparando entre sí los valores que intercambia, la complejidad de este algoritmo es de O (n^2) en todos los casos. El

bubble sort consiste en intercambiar valores haciendo una especie de burbuja entre ellos, por ende, su nombre, su complejidad es de O (n) en su mejor caso, ya que recorre el arreglo completo y O (n^2) cuando tiene que recorrer el arreglo completo dos veces. El insertion sort consiste en hacer uso de un pivote en el cual vamos moviendo los elementos que son mayores o menores en sus posiciones designadas. Su complejidad es de O (n) en su mejor caso y O(n^2) en el peor de los casos.

Como lo veo, pudimos haber utilizado cualquiera de estos algoritmos de ordenamiento en la situación problema, sin embargo, nosotros nos quedamos con el merge sort, que nos pareció el más eficiente y que se acoplaba mejor a nuestra actividad, sin contar que también hace uso de la recursividad en su diseño. Debido a que los datos a los que tenemos que ordenar son muy grandes es importante buscar la mayor eficiencia en cuanto a complejidad temporal, ya que esto nos podrá ahorrar procesamiento de la memoria, ahorrando operaciones complejas en nuestros algoritmos.