Jorge Loredo Meléndez A01284185

Actividad 1.3

Tanto el tiempo de ejecución como la complejidad de un algoritmo son de suma importancia al momento de estructurar un programa o aplicación. Estas dos variables determinan la eficiencia del mismo, así como de otros factores trascendentes para la interacción con un usuario, manejo de eventos y consumo de recursos. Los principales algoritmos aplicados a estructuras de datos son los métodos de búsqueda y ordenamiento.

Los métodos de ordenamiento engloban varios procesos como el cambio de posición de un valor o la creación de nuevas estructuras de datos. Cabe resaltar que el caso promedio de ordenamiento es el que normalmente se toma en cuenta para asignar cada complejidad O(n). Algunos método que entran a esta categoría por complejidad son:

- $O(n^2)$:
    - Bubble Sort: Se detecta si el siguiente valor es mayor al actual y en caso de que sea cierto los intercambia.
    - Exchange Sort: Se compara el primer elemento del arreglo con los demás y va intercambiando por el menor valor en caso de ser necesario
    - Selection Sort: Busca el menor elemento de la lista y lo intercambia por la primera posición. Después vuelve a hacer la misma comparación sin tomar en cuenta la posición anterior.
- O(nlog(n)):
    - Merge Sort: Se divide la lista en unidades. Posteriormente se comparan dos elementos adyacentes y se ordenan. Se repite el proceso hasta que la lista completa esté ordenada.
    - Quick Sort: Se escoge un pivote, se localizan todos los elementos mayores al pivote de lado derecho y a los menores en el lado izquierdo. Finalmente se escoge otro pivote y se repite el proceso.

Por otro lado, los métodos de búsqueda tienen la siguiente complejidad:

- O(n):
    - Búsqueda Secuencial: Se itera por todos las posiciones de un arreglo hasta encontrar el valor buscado.
- O(log(n)):
    - Búsqueda Binaria: En un arreglo ordenado se apunta al valor intermedio, y se compara si el valor buscado es mayor o menor al valor apuntado, en cualquier caso se ve a la mitad del arreglo que contenga el valor y omite la sección sobrante.

Cada método tiene un mejor y peor caso, dependiendo de la naturaleza del problema puede que algunos resalten sobre otros. Por ejemplo, si se tiene una lista cuyos únicos elementos desordenados se encuentran al principio, el Bubble Sort o Exchange Sort pueden tener un mejor tiempo de ejecución que algoritmos más complejos. Por otro lado, utilizando el caso

promedio un Quick Sort tiene una mayor eficiencia que un Selection Sort. En nuestro problema optamos por usar un Quick Sort al momento de ordenar los registros ya que la lista contenía muchos objetos a ordenar y el tiempo de ejecución lo exigía. Asimismo a través de los boundaries (low - up) utilizamos búsqueda binaria para regresar al usuario el rango de registros solicitados.