Reflexión Actividad Integral 1.3

Fernanda Monserrat Galván Romero
A00344712

Sinceramente esta actividad fue un reto a realizar por toda la parte de la lectura y escritura de archivos, además de que meterle 16 mil datos a mi computadora fue un poco estresante. Por esta misma razón es que es tan importante saber qué algoritmos son mejores o más aptos para llevar a cabo tareas muy largas, ya que muchas veces se busca que corran en menos tiempo para hacer más eficiente el proceso.

Primeramente quiero decir que usar el algoritmo de merge fue la mejor opción por su efectividad, en la actividad pasada comprobé que tenía menos iteraciones que el bubble o el swap. Y aunque si intento implementarlos en el código al momento de tomar tiempo se notaba la diferencia entre el merge y el bubble o el swap, entonces use el que tenía el menor tiempo.

La diferencia de estos tiempos radica en la forma en la que operan: (profe aqui estoy explicando como funciona cada algoritmo si quiere no lo lea), el bubble hace comparaciones inmediatas para saber que numero es mayor y este a su vez va “recorriendo” este número que detecto como el mayor al final de la lista, de manera que queda ordenado de forma ascendente. Ahora, el swap se maneja de una forma un poco más tardada, este también hace comparaciones inmediatas pero cuando encuentra que el primer número es mayor al número siguiente los cambia de lugar, esto puede llegar a provocar que haya cambios innecesarios lo que a la larga afecta el rendimiento del algoritmo.

Por último tenemos al merge, este método divide el arreglo en 2, y esa mitad vuelve a partirse hasta que solo quedan pares, que después compara y acomoda para después volver a unir todo dejándonos un arreglo ordenado.Como podemos observar en el peor de los casos obtenemos O(n log n), pero a diferencia de los demás algoritmos, estos tienen un valor de n^2. Por lo que podemos observar esa notable diferencia en la siguiente tabla

Merge Sort		
Mejor	Promedio	Peor
$O(n\ log\ n)$	$O(n\ log\ n)$	$O(n\ log\ n)$

Swap Sort		
Mejor	Promedio	Peor
$O(n^2)$	$O(n^2)$	$O(n^2)$

Bubble Sort		
Mejor	Promedio	Peor
$O(n)$	$O(n^2)$	$O(n^2)$

Para tener una mejor visualización de cómo funcionan los datos de arriba los sustituiremos con nuestro número de datos. N = 16807

	Mejor	Promedio	Peor
Merge Sort	*O*(71,019)	*O*(71,019)	*O*(71,019)
Bubble Sort	*O*(16,807)	*O*(282′475,249)	*O*(282′475,249)
Swap Sort	*O*(282′475,249)	*O*(282′475,249)	*O*(282′475,249)

Y para el algoritmo de búsqueda use el de búsqueda binaria por su eficiencia, ya que no gasta iteraciones en pasar por todos los elementos, sino que como ya sabemos parte de la mitad del arreglo y va haciendo eso hasta llegar a la posición deseada. A diferencia de la búsqueda secuencial que consiste en buscar diagonalmente el dato que necesitas. Si no encuentra el dato requerido en la diagonal, se pasa al siguiente dato y busca de la misma forma.

Podemos notar cual es la diferencia entre ambos algoritmos a simple vista, pero por si acaso le voy a hacer la sustitución.

Búsqueda Binaria		
Mejor	Promedio	Peor
$O(1)$	$O(log_2 n)$	$O(log\, n)$

Búsqueda Secuencial		
Mejor	Promedio	Peor
$O(1)$	$O(\,n)$	$O(\,n)$

	Mejor	Promedio	Peor
Búsqueda Binaria	O(1)	O(14.037)	O(4.225)
Búsqueda Secuencial	O(1)	O(16,807)	O(16,807)

La diferencia más notable está en el caso promedio y en el peor, por lo tanto es muy fácil decidir cual usar en este código que tiene miles de datos para correrlo de manera eficiente

Creo que esta actividad si me dejo ver porque es importante la eficiencia y ahorro de recursos, y aunque más que nada es porque a mi me encanta tener cosas que no hacen nada en mi codigo, ya aprendí que eso gasta memoria y hace que se trabe mi computadora, asi que esta vez aprendi por las malas por que no tenemos que hacer cosas extras. Además de que era muy evidente cuando había cosas que alentaban el programa, pasaba de completarse en 30000 ms a completarse en 350 ms y aparte como no tengo paciencia pues se hacía aún más difícil hacer casos de prueba.