Laboratorio Nro. 2
Notación O grande

Simón Marín Giraldo
Universidad Eafit
Medellín, Colombia
smaring1@eafit.edu.co

Miguel Fernando Ramos García
Universidad Eafit
Medellín, Colombia
mframosg@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

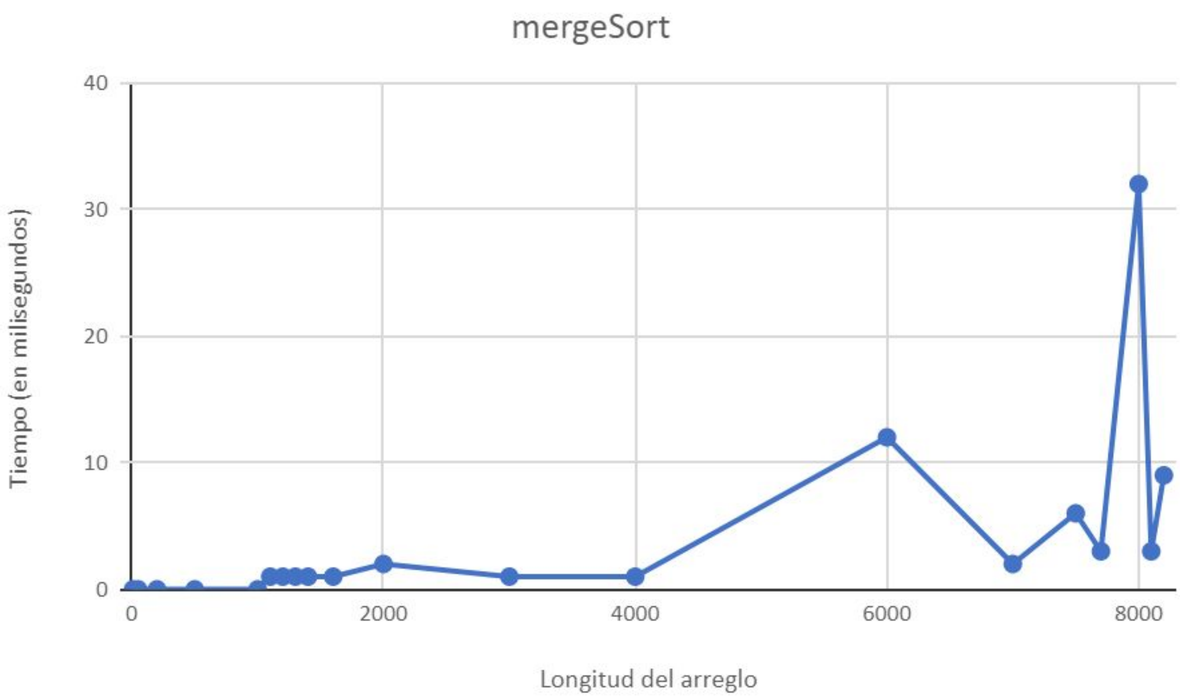
3.1

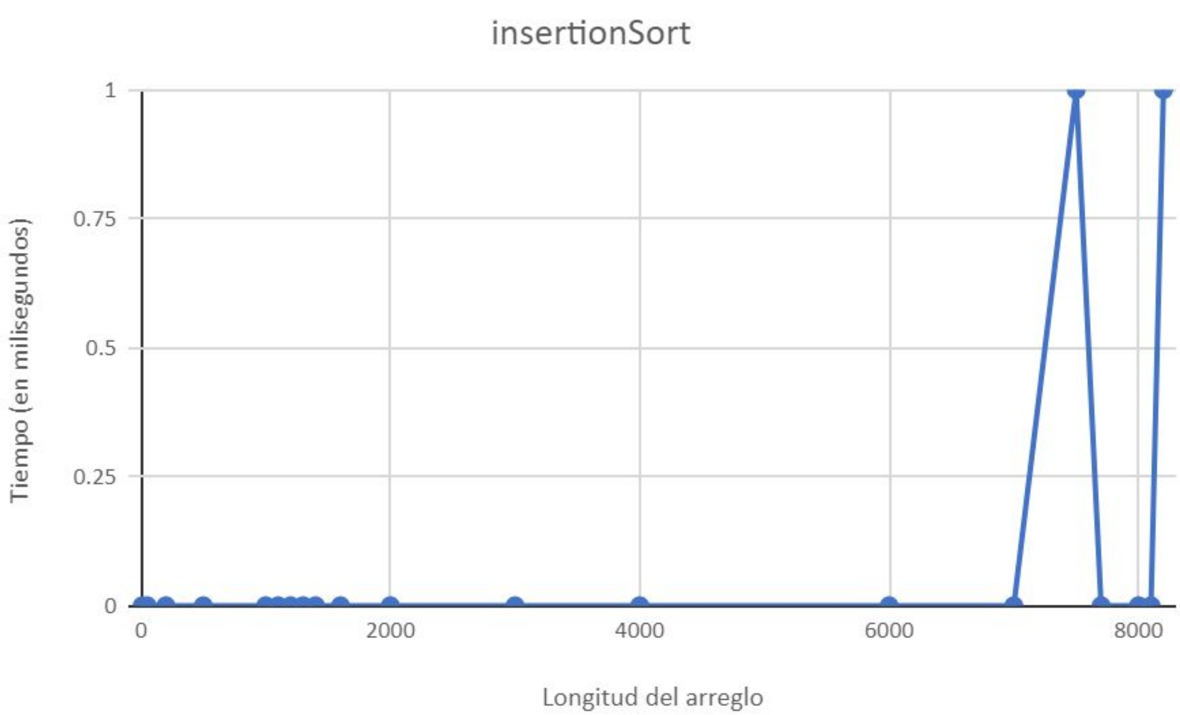
mergeSort	
LONGITUD	TIEMPO (EN MILISEGUNDOS)
10	0
50	0
200	0
500	0
1000	0
1100	1
1200	1
1300	1
1400	1
1600	1
2000	2
3000	1
4000	1
6000	12
7000	2
7500	6
7700	3
8000	32
8100	3
8200	9

insertionSort	
LONGITUD	TIEMPO (EN MILISEGUNDOS)
10	0
50	0
200	0
500	0
1000	0
1100	0
1200	0
1300	0
1400	0
1600	0
2000	0
3000	0
4000	0
6000	0
7000	0
7500	1
7700	0
8000	0
8100	0
8200	1

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.2





PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.3 Dado que la complejidad para merge sort es de O(n log n) y la de insertion sort es de O(n^2), merge sort es menos ineficiente que insertion sort.

3.4 Para un videojuego de millones no es apropiado usar insertion sort dado que tiene complejidad O(n^2) razón por la cual, cada vez que aumentemos en 1 la cantidad de elementos, el algoritmo tardará el doble para ejecutar todo el proceso. Si ponemos un millón de elementos en el arreglo, el algoritmo tomará 555.5 segundos para ejecutarse, casi 10 minutos. Es totalmente ineficiente para usarse en un juego.

3.5 Para todos los casos, merge sort es más eficiente, razón por la cual no hay casos en los que sea preferible usar insertion sort. Se puede usar sin tener diferencias significativas en el rendimiento únicamente para arreglos de tamaños pequeños. Ya si la eficiencia es prioridad, lo más conveniente es merge sort.

3.6 En este problema nos piden que consideremos las apariciones a toda la izquierda y a toda la derecha de un valor en el arreglo. Tomaremos el span o lapso como la cantidad de elementos que hay entre dos elementos cualquiera, incluyendo a ambos límites. Un único valor solo posee un lapso o span de 1. Para resolver este problema, inicialmente se debe evaluar la condición de que si la longitud del arreglo es cero, por lo tanto no existirán lapsos y se retornará inmediatamente cero. Posteriormente se evalúa si el primer elemento del arreglo posee el mismo valor que el último elemento del arreglo. En este caso se retornará el lapso, que es el mayor caso posible, es decir la longitud del arreglo, que es igual al lapso desde el primer elemento hasta el último, incluídos ambos. Por último, si ninguna de las dos anteriores condiciones se cumple, se retorna la última posición del arreglo.

3.7 countEven = T(n) = C_1 + C_2 + C_3 * n + C_4 * n + C_5 * n + C_6 = O(C_3 * n) = O(n)

lucky13 = T(n) = C_1 + C_2 + C_3 * n + C_4 + C_5 * n + C_6 * n + C_7 + C_8 + C_9 = O(C_6 * n) = O(n)

only14 = T(n) = C_1 + C_2 + C_3 * n + C_4 + C_5 * n + C_6 + C_7 + C_8 = O(C_5 * n) = O(n)

zeroMax = T(n) = C_1 + C_2 + C_3 * n + C_4 * n + (C_5 + C_6 * n) * n + C_7n * n + C_8 * n * n + C_9 * n + C_10 * n + C_11 = O(C_8 * n * n) = O(n*n) = O(n^2)

sum13 = T(n) = C_1 + C_2 + C_3 * n + C_4 * n + C_5 * n + C_6 + C_7 * n + C_8 * n + C_9 * n + C_10 = O(C_9 * n) = O(n)

3.8 “n” Indica la cantidad de procesos a realizar en un algoritmo. En algunas ocasiones aparece una variable “m” que tiene un funcionamiento muy similar, casi igual a la “n”.

4) Simulacro de Parcial

4.1 c
4.2 b
4.3 b
4.4 b
4.5 d

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4.6 a
4.7 T(n-1)+C_1 y complejidad asintótica O(n).
4.8 a
4.9 d
4.10 c
4.11 c
4.12 b
4.13 c
4.14 a

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473