ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 2
Complejidad de algoritmos

Valentina Moreno Ramírez
Universidad Eafit
Medellín, Colombia
vmorenor@eafit.edu.co

Alejandra Palacio Jaramillo
Universidad Eafit
Medellín, Colombia
apalacioj@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

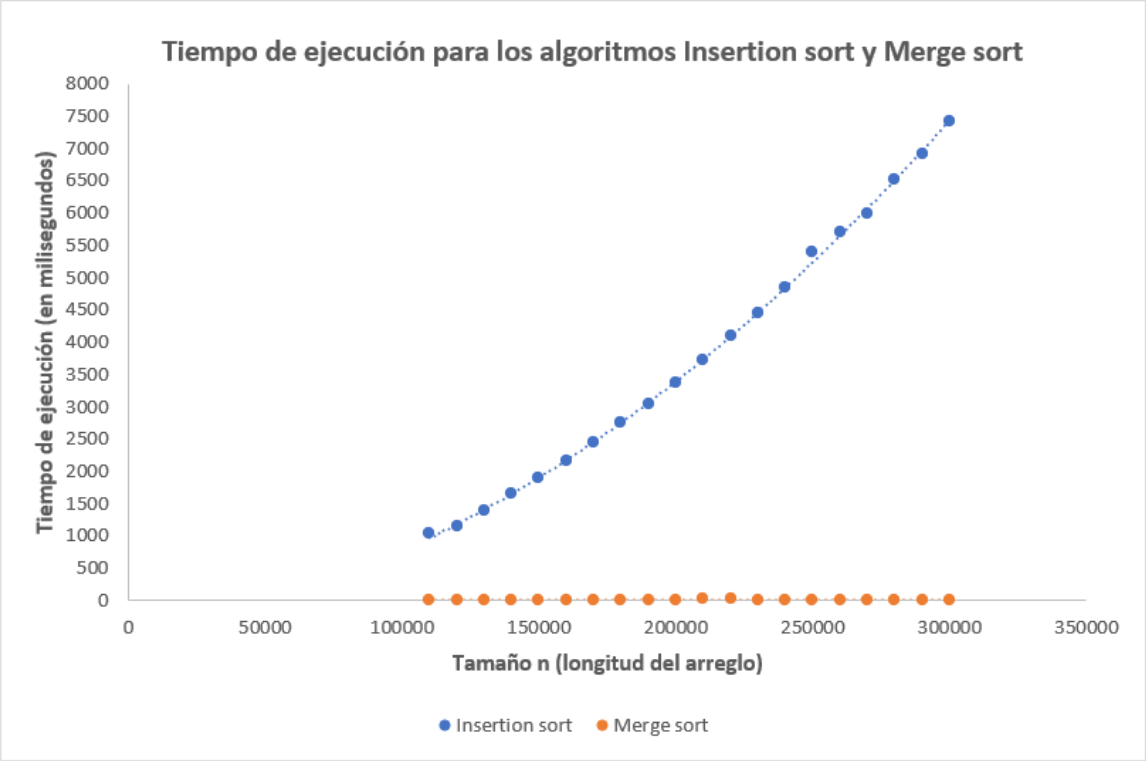
3.1 En la siguiente tabla, se muestran los tiempos de ejecución tomados en milisegundos para tamaños n de un arreglo en los algoritmos Insertion sort y Merge sort. El primer algoritmo ordena los elementos comparando el número a ordenar con los números en posiciones anteriores para determinar la posición en la que este debe ir, mientras que el segundo divide el arreglo original en arreglos más pequeños ("divide y vencerás") para compararlos entre si y ubicarlos nuevamente en un solo arreglo.

	T(n)	T(n)
n	Insertion sort	Merge sort
110000	1032	9
120000	1156	11
130000	1396	12
140000	1659	13
150000	1894	7
160000	2168	7
170000	2444	9
180000	2749	9
190000	3034	9
200000	3375	12
210000	3719	20
220000	4101	21
230000	4451	16
240000	4846	12
250000	5393	12
260000	5715	13
270000	5997	16
280000	6515	15
290000	6928	14
300000	7413	16

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.2 Grafica tiempo de ejecución vs tamaño n de los algoritmos planteados anteriormente.



En esta gráfica se evidencia que la complejidad del insertion sort es $O(n^2)$, mientras que la del merge sort es O(nlogn).

3.3 No es apropiado usar el insertion sort para realizar esta tarea, ya que este presenta una complejidad cuadrática, lo cual implica que cada vez que se tengan más elementos de una escena, este tiempo se incrementará rápidamente. Para este caso es recomendable usar el merge sort debido a que esté tarda menos en ejecutarse que el insertion sort, así se tengan arreglos con millones de elementos.

3.4 El algoritmo de merge sort es el que presenta un logaritmo en su complejidad porque este divide el arreglo en arreglos más pequeños para poder ordenarlos. Cuando esto sucede (dividir el problema en problemas más pequeños) se obtiene una complejidad que implica un logaritmo debido a que se plantea, por ejemplo, en este caso, cuántas veces se puede dividir n (la longitud del arreglo) en 2 antes de que se vuelva 0.

3.5 (Opcional) El peor de los casos del algoritmo insertion sort será cuando todos los datos estén desordenados, por ende, si el usuario ingresa un arreglo que esté casi ordenado, el algoritmo será más eficiente que el merge sort por el hecho de no tener que hacer tantos llamados recursivos y tantas comparaciones como en el merge sort.

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.6 (Opcional) El ejercicio de MaxSpan recibe un arreglo como parámetro. Con base en dos números, uno que está más a la izquierda y otro que está más a la derecha, el algoritmo busca por todo el arreglo cuantos elementos hay en el intervalo de esos dos números y almacena este valor en la variable span. También se tendrá una variable llamada maxSpan que será igual en determinado caso al número de elementos en el intervalo y será el valor que se retornará. Para conseguir esto, se usaron dos bucles *for* que servirán para encontrar el número de elementos entre el número que está más a la derecha y el que está más a la izquierda, luego, desde que los números de la derecha e izquierda sean diferentes, se restan los índices y se suma uno para saber a cuánto equivale span. Por último, si span es mayor que maxSpan en ese caso, este tomará el valor de span, ya que el objetivo del ejercicio es retornar el span más largo del arreglo.

3.7

Array2:

CountEvens:
$T(n) = c_1 + c_2n + c_3n + c_4$
$T(n)$ es $O(c_1 + c_2n + c_3n + c_4)$, aplicando la regla de reflexividad.
$O(n(c_2 + c_3))$, aplicando la regla de la suma.
$O(n)$, aplicando la regla del producto.
La complejidad del algoritmo es O(n), es decir, lineal, donde n es el número de elementos del arreglo.

Sum13:
$T(n) = c_1 + c_2n + c_3n + c_4n + c_5$
$T(n)$ es $O(c_1 + c_2n + c_3n + c_4n * c_5)$, aplicando la regla de regla de reflexividad.
$O(n(c_2 + c_3 + c_4))$, aplicando la regla de la suma.
$O(n)$, aplicando la regla del producto.
La complejidad del algoritmo es O(n), es decir, lineal, donde n es el número de elementos del arreglo.

FizzArray:
$T(n) = c_1 + c_2n + c_3n + c_4$
$T(n)$ es $O(c_1 + c_2n + c_3n + c_4)$, aplicando la regla de reflexividad.
$O(n(c_2 + c_3))$, aplicando la regla de la suma.
$O(n)$, aplicando la regla del producto
La complejidad del algoritmo es O(n), es decir, lineal, donde n es el número de elementos del arreglo que retorna este método.

More14:
$T(n) = c_1 + c_2 + c_3n + c_4n + c_5n + c_6$
$T(n)$ es $O(c_1 + c_2n + c_3n + c_4n + c_5n + c_6)$, aplicando la regla de reflexividad.
$O(n(c_2 + c_3 + c_4 + c_5))$, aplicando la regla de la suma.

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

O(n), aplicando la regla del producto.
La complejidad del algoritmo es O(n), es decir, lineal, donde n es el número de elementos del arreglo.

Sum28:
$T(n) = c_1 + c_2n + c_3n + c_4$
$T(n)$ es $O(c_1 + c_2n + c_3n + c_4)$, aplicando la regla de reflexividad.
$O(n(c_2 + c_3)$, aplicando la regla de la suma.
O(n), aplicando la regla del producto.
La complejidad del algoritmo es O(n), es decir, lineal, donde n es el número de elementos del arreglo.

Array3:

LinearIn:
$T(n) = c_1 + c_2n + c_3mn + c_4mn + c_5mn + c_6mn + c_7 + c_8$
$T(n)$ es $O(c_1 + c_2n + c_3n^2 + c_4n^2 + c_5n^2 + c_6n^2 + c_7 + c_8)$ Como m≤n, entonces en el peor caso m = n
$O(n^2(c_3 + c_4 + c_5 + c_6)$
$O(n^2)$

SeriesUp:
$T(n) = c_1 + c_2 + c_3n + c_4n*n + c_5n*n + c_6n*n + c_7$
$T(n)$ es $O(c_1 + c_2 + c_3n + c_4nm + c_5nm + c_6nm + c_7)$ Como m≤n, entonces en el peor caso m = n
$O(l^2(c_4 + c_5 + c_6)$
$O(l^2)$

MaxSpan:
$T(n) = c_1 + c_2 + c_3 + c_4n + c_5n^2 + c_6n^2 + c_7n^2 + c_8n^2 + c_9$
$T(n)$ es $O(c_1 + c_2 + c_3 + c_4n + c_5n^2 + c_6n^2 + c_7n^2 + c_8n^2 + c_9\, n^2 + c_{10}n^2 + c_{11})$
$O(n^2(c_5 + c_6 + c_7 + c_8)$
$O(n^2)$

Fix34:
$T(n) = c_1 + c_2n + c_3n^2 + c_4n^2 + c_5n^2 + c_6n^2 + c_7n^2 + c_8$
$T(n)$ es $O(c_1 + c_2n + c_3n^2 + c_4n^2 + c_5n^2 + c_6n^2 + c_7n^2 + c_8)$, aplicando regla de reflexividad.
$O(n^2(c_3 + c_4 + c_5 + c_6 + c_7)$, aplicando regla de la suma.
$O(n^2)$, aplicando regla del producto.
La complejidad del algoritmo es $O(n^2)$, es decir, cuadrática, donde n es el número de elementos del arreglo que retorna este método.

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

CanBalance:
$T(n) = c_1 + c_2 + c_3n + c_4n + c_5n(n-1) + c_6n(n-1) + c_7n + c_8n + c_9n + c_{10}$
$T(n)$ es $O(c_1 + c_2 + c_3n + c_4n + c_5n(n-1) + c_6n(n-1) + c_7n + c_8 + c_9n + c_{10})$, aplicando regla de reflexividad
$T(n) = O(c_1 + c_2 + c_3n + c_4n + c_5n^2 - c_5n + c_6n^2 + - c_6n + c_7n + c_8 + c_9n + c_{10})$, haciendo las multiplicaciones implicadas
$O(n^2(c_5 + c_6)$, aplicando regla de la suma.
$O(n^2)$, aplicando regla del producto.
La complejidad del algoritmo es $O(n^2)$, es decir, cuadrática, donde n es el número de elementos del arreglo que retorna este método.

3.8 La n en los ejercicios 1, 2, 4 y 5 de array2 es la longitud del arreglo que se pasa como parámetro. En el caso de del ejercicio fizzArray, n es el número de elementos que tendrá el arreglo que retorna el ejercicio. Y en los ejercicios de array3, en LinearIn, la n será la longitud del arreglo externo y la m del arreglo interno. Y en los otros ejercicios la n será la longitud del arreglo respectivamente.

4) Simulacro de Parcial

4.1 *c*
4.2 *d*
4.3 *b*
4.4 *b*
4.5
a. opción d O(log n).
b. no.
4.6 10000 segundos, en otras palabras, 10 segundos.
Por los datos dados en el enunciado, se sabe que T(100) = 1 segundo, lo cual quiere decir que $c*n^2 = 1$. Si se reemplaza en la función a n por 100, se obtiene que $c*(100)*(100) = 1$, lo cual es igual a $10000*c = 1$ y a su vez es igual a $c = 1/10000$. Como c es una constante entonces no varía, por tanto, se reemplaza en la complejidad para poder hallar T(10000). Esto sería $T(10.000) = (1/10000)*(10000)^2 = 10000$ segundos.
4.7 Todas son verdaderas.
4.8 a
4.9 a
4.10 c
4.11 c
4.12 b
4.13 c
4.14 a

5) Lectura recomendada (opcional)

Mapa conceptual

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

6) Trabajo en Equipo y Progreso Gradual (Opcional)

6.1 *Actas de reunión*
6.2 *El reporte de cambios en el código*
6.3 *El reporte de cambios del informe de laboratorio*

PhD. Mauricio Toro Bermúdez

Correo: | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473