

# LABORATORIO 2.

## ESTRUCTURAS DATOS Y ALGORITMOS 1.

**Tomas Marin Aristizábal**

**Colombia**

**Eafit**

[tmarina@eafit.edu.co](mailto:tmarina@eafit.edu.co)

**Juan Andrés Vera**

**Colombia**

**Eafit**

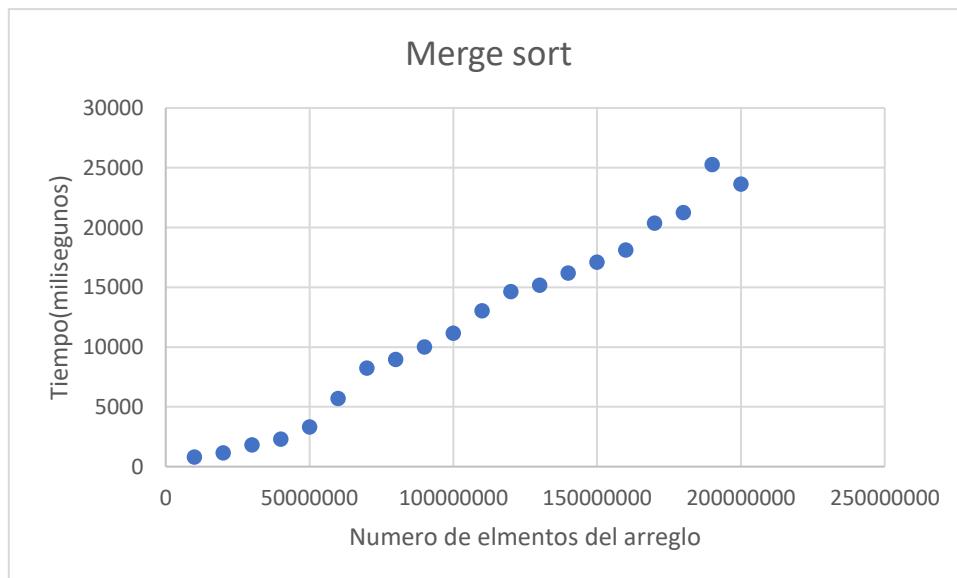
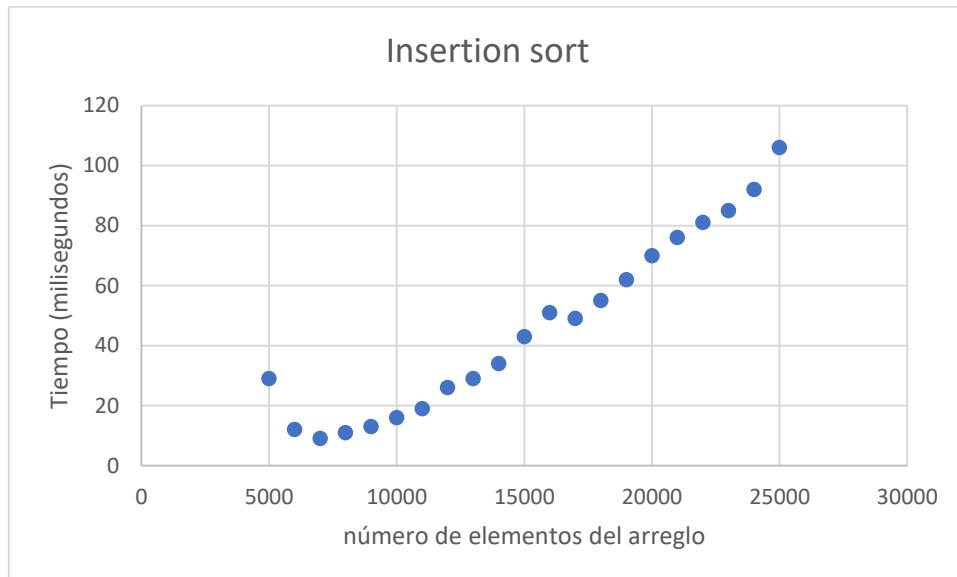
[javeraa@eafit.edu.co](mailto:javeraa@eafit.edu.co)

### 1.1 Tiempo de ejecución insertion sort y merge sort.

Insertion Sort		Merge	sort
Número de elementos del arreglo	Tiempo en milisegundos	Número de elementos del arreglo	Tiempo en milisegundos
5000	29	10000000	810
6000	12	20000000	1151
7000	9	30000000	1836
8000	11	40000000	2321
9000	13	50000000	3318
10000	16	60000000	5710
11000	19	70000000	8244
12000	26	80000000	8978
13000	29	90000000	10016
14000	34	100000000	11173
15000	43	110000000	13048
16000	51	120000000	14642
17000	49	130000000	15193
18000	55	140000000	16203
19000	62	150000000	17114
20000	70	160000000	18133

21000	76	170000000	20388
22000	81	180000000	21254
23000	85	190000000	25269
24000	92	200000000	23647
25000	106		

### 3.2



### 3.3

Para videojuegos que usan gran volumen de datos e información no es adecuado usar el insertion sort ya que comparando tiempos podemos notar que tenemos una gran diferencia en tiempos de ejecución ya que el merge sort es capaz de ejecutarse con cantidades de 10000000 en segundos mientras que el insertion sort se demora mucho más y al ser un video juego no sería adecuado que tomara mucho tiempo en ejecutarse en tiempo real.

**3.4** Se genera un logaritmo en merge sort porque en una parte del algoritmo solo se ejecuta en ciertos casos por ende nos muestra que la complejidad es  $O(n \log(n))$

### 3.7

#### Complejidad Array 2

1.  $T(n): c_1 + c_2 + c_3 * n + c_4 + c_5 * n + c_6 + c_7$  Complejidad  $O(n)$
2.  $T(n): c_1 + c_2 * n + c_3 * n + c_4 + c_5 * n + c_6 + c_7 * n + c_8 + c_9 * n + c_{10} * n + c_{11} * n + c_{12} * n + c_{13} + c_{14} + c_{15} * n + c_{16}$  Complejidad  $O(n^2)$
3.  $c_1 + c_2 + c_3 * n + c_4 * n + c_5 + c_6 * n + c_7 * n + c_8 + c_9$  Complejidad  $O(n)$
4.  $c_1 * n + c_2 * n + c_3 + c_4 * n + c_5 * n + c_6 * n + c_7$  Complejidad  $O(n)$
5.  $c_1 + c_2 + c_3 + c_4 * n + c_5 + c_6 * n + c_7 + c_8 + c_9 * n + c_{10} + c_{11} + c_{12}$  Complejidad  $O(n)$

#### Complejidad Array 3

1.  $c_1 + c_2 + c_3 * n + c_4 * n + c_5 * n + c_6 + c_7 * n + c_8 + c_9 * n + c_{10} + c_{11} * n + c_{12} + c_{13} * n + c_{14} * n + c_{15}$  Complejidad  $O(n^2)$
2.  $c_1 + c_2 * n + c_3 + c_4 * n + c_5 + c_6 * n + c_7 + c_8 * n + c_9 + c_{10} * n + c_{11} * n + c_{12} + c_{13} * n + c_{14} * n + c_{15} + c_{16} * n + c_{17} + c_{18} + c_{19} + c_{20} + c_{21} * n + c_{22} * n + c_{23} + c_{24} * n + c_{25} + c_{26}$  Complejidad  $O(n^2)$
3.  $c_1 + c_2 + c_3 * n + c_4 + c_5 * n + c_6 + c_7 + c_8 * n + c_9 + c_{10} * n + c_{11} + c_{12} * n + c_{13} + c_{14}$  Complejidad  $O(n^2)$
4.  $c_1 + c_2 + c_3 + c_4 * n + c_5 + c_6 + c_7 * n + c_8 + c_9 * n + c_{10} * n + c_{11} + c_{12} + c_{13} + c_{14} * n + c_{15} * n + c_{16} + c_{17} + c_{18} * n + c_{19} + c_{20}$  Complejidad  $O(n)$
5.  $c_1 * n + c_2 * n^2 + c_3 + c_4 + c_5 * n + c_6 + c_7 * n + c_8 * n + c_9 + c_{10}$  Complejidad  $O(n^2)$

### 3.8

1.  $n$  significa el numero del tamaño del arreglo

- 2.** n significa el número del tamaño del arreglo
- 3.** n significa el número del tamaño del arreglo
- 4.** n es un numero que se va a usar en el programa
- 5.** n significa el número del tamaño del arreglo
- 6.** n significa el número del tamaño del arreglo
- 7.** n significa el número del tamaño del arreglo
- 8.** n significa el número del tamaño del arreglo
- 9.** n y m significan el número de tamaños de dos arreglos
- 10.** n es un número que se va a usar en el programa

**4.2** B

**4.5.1** D

**4.5.2** B (No)

**4.6**  $T(n) = c * n^2$  que despejando seria  $T(n)/n^2 = c$  dando como resultado  $c = 0.1\text{ms}$ .

**4.7.1** falsa

**4.7.2** Verdadera

**4.7.3** falsa

**4.7.4** verdadera

**4.9.** A

**4.14.** C