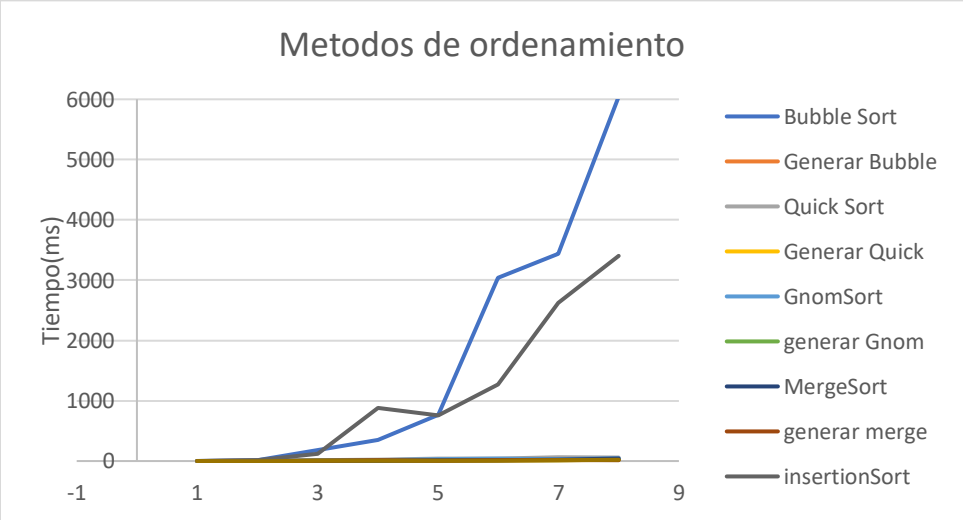
Universidad del Valle de Guatemala
Algoritmos y estructuras de datos - Sección 20
1er. semestre 2022
Integrantes
Wilfredo Gallegos 20399
Guillermo Furlán 20713
Alejandro Pallais 20093

Rendimiento de ordenamiento

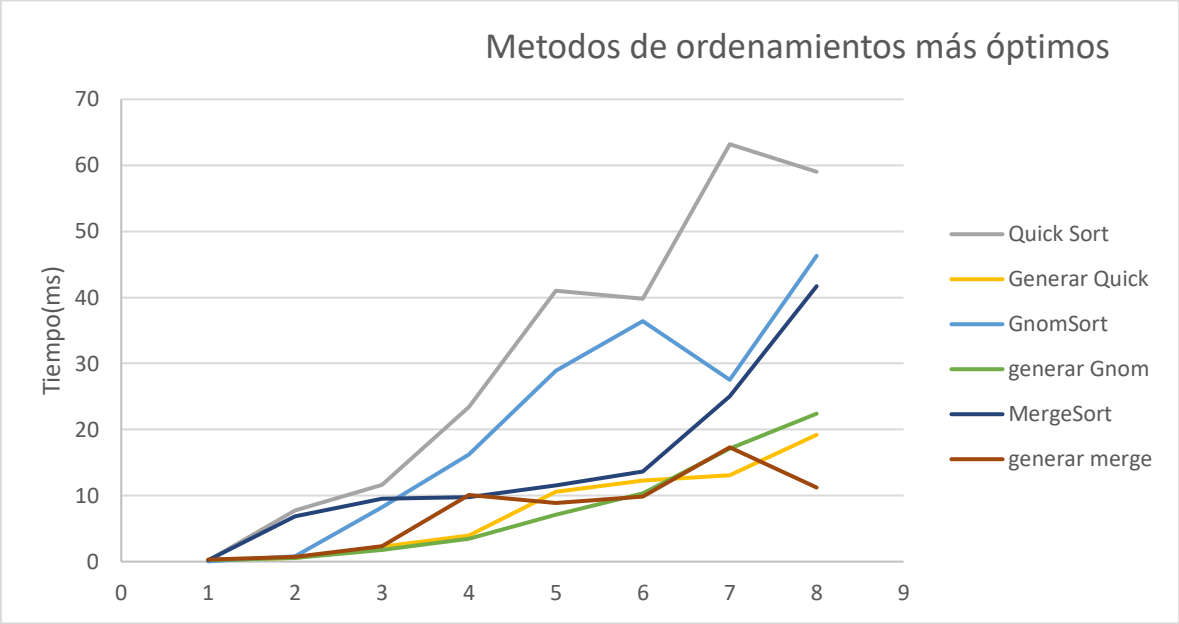
Para cada uno de los métodos de ordenamiento se realizó distintas formas de hacerlo(bubbleSort, QuickSort …) a distintas cantidades, entre ellas está 10, 1000, y hasta 3000 números aleatorios. A continuación, se puede observar en una tabla los valores obtenidos en milisegundos para cada método de ordenamiento, y cuanto tiempo tardo en crear los números aleatorios.

números	10	100	500	1000	1500	2000	2500	3000
Bubble Sort	1.33	15	179	347	762	3035	3437	6044
Generar Bubble	0.479	1.24	2.68	3.1	10.1	14.5	12.9	22.7
Quick Sort	0.165	7.73	11.6	23.4	41	39.8	63.2	59
Generar Quick	0.243	0.534	2.23	3.92	10.6	12.3	13.1	19.2
GnomSort	0.029	0.797	8.22	16.2	28.9	36.4	27.5	46.3
generar Gnom	0.224	0.548	1.73	3.49	7.1	10.3	17.1	22.4
MergeSort	0.211	6.83	9.48	9.73	11.5	13.6	25	41.7
generar merge	0.307	0.752	2.3	10.1	8.85	9.87	17.3	11.2
insertionSort	0.575	13.1	122	879	754	1268	2620	3403
generar insertion	0.406	0.911	3.56	5.72	3.46	5.1	13.5	20.6

Para representar la eficiencia de cada uno de estos de una forma más sencilla de entender se presenta la siguiente grafica.



Pero de esta grafica se puede observar fácilmente que los métodos de ordenamiento BubbleSort e insertionSort son poco eficiente, por lo que se consideró apropiado eliminarlos y comparar el resto de métodos.



Ignorando los 2 métodos eliminados por no ser suficientemente eficientes, de los que quedan se puede observar que el método MergeSort es el mas eficiente en este caso, aunque el método GnomeSort tuvo al final un valor muy parecido. La variación de estos tiempos se debe a la aleatoriedad de los números generados al principio, si de por si ya estaban algunos ordenados que redujeron el tiempo.

Rendimiento Teórico

Por otro lado, se investigó el rendimiento teórico de cada uno de los 5 métodos utilizando Big(O). los rendimientos teóricos fueron los siguientes:

BubbleSort: $O(n^2)$, poco eficiente.

QuickSort: O(n*log((n)), en peores casos $O(n^2)$, también es considerado método mediocre en los peores casos.

GnomeSort: O(n) en algunos casos $O(n^2)$, tampoco suficientemente eficiente.

MergeSort: O(n*log((n)), considerado método excepcional.

InsertionSort: O(n) en algunos casos $O(n^2)$, mal rendimiento. (Big-O, 2020)

Nótese que estas funciones nos devuelven la cantidad de operaciones realizadas a partir de la cantidad de datos.

Las pruebas unitarias se adjuntaron a la carpeta donde se encuentra el programa.

Para obtener los tiempos que se tardan en operar cada uno de los métodos se utilizó la función Profiler de VisualVM y los resultados se obtuvieron de una ventana como la que se muestra a continuación.

Sort.**BubbleSort**(Object[])	1.33 ms (50.4 %)	0.0 ms (- %)	1
Self time	0.723 ms (27.3 %)	0.0 ms (- %)	1
CompareInt.**compare** (Object, Object)	0.609 ms (23 %)	0.0 ms (- %)	45
lector.**Escribir** (String)	0.709 ms (26.8 %)	0.0 ms (- %)	1
GeneradorNum.**Generar** (int)	0.479 ms (18.1 %)	0.0 ms (- %)	1
GeneradorNum.**listarNum** (String, int)	0.125 ms (4.7 %)	0.0 ms (- %)	1

Figura 1: datos obtenidos

Figura 1: Profiler Utilizado

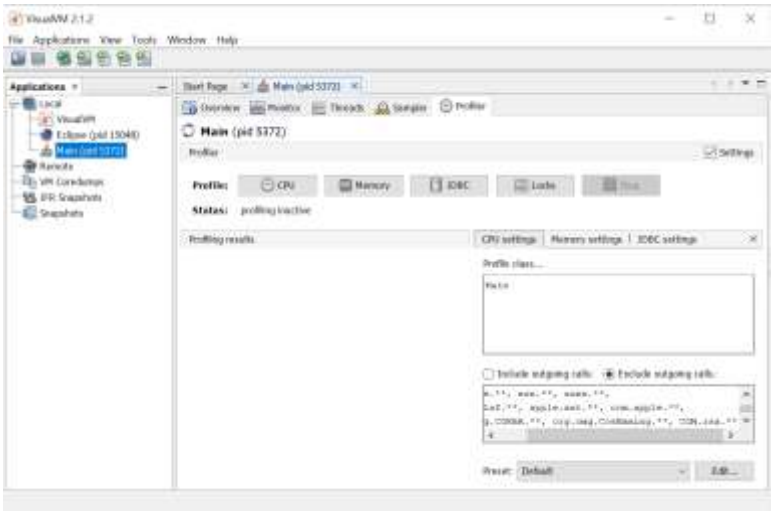


Figura 1: Ventana de visualización

Fuentes de Consulta

Big-O. (2020). *Big-O.* Obtenido de big-o.io