Estructuras de datos avanzadas

Tarea 1

Comparación de algoritmos de ordenamiento

Diego Hernández Delgado

Clave única: 176262

Grupo: 001

Maestra: Fernando Esponda

Fecha de entrega: 18-septiembre-2019

Índice

OBJETIVO

El objetivo de esta tarea es determinar empíricamente el desempeño de los algoritmos de ordenamiento vistos en clase.

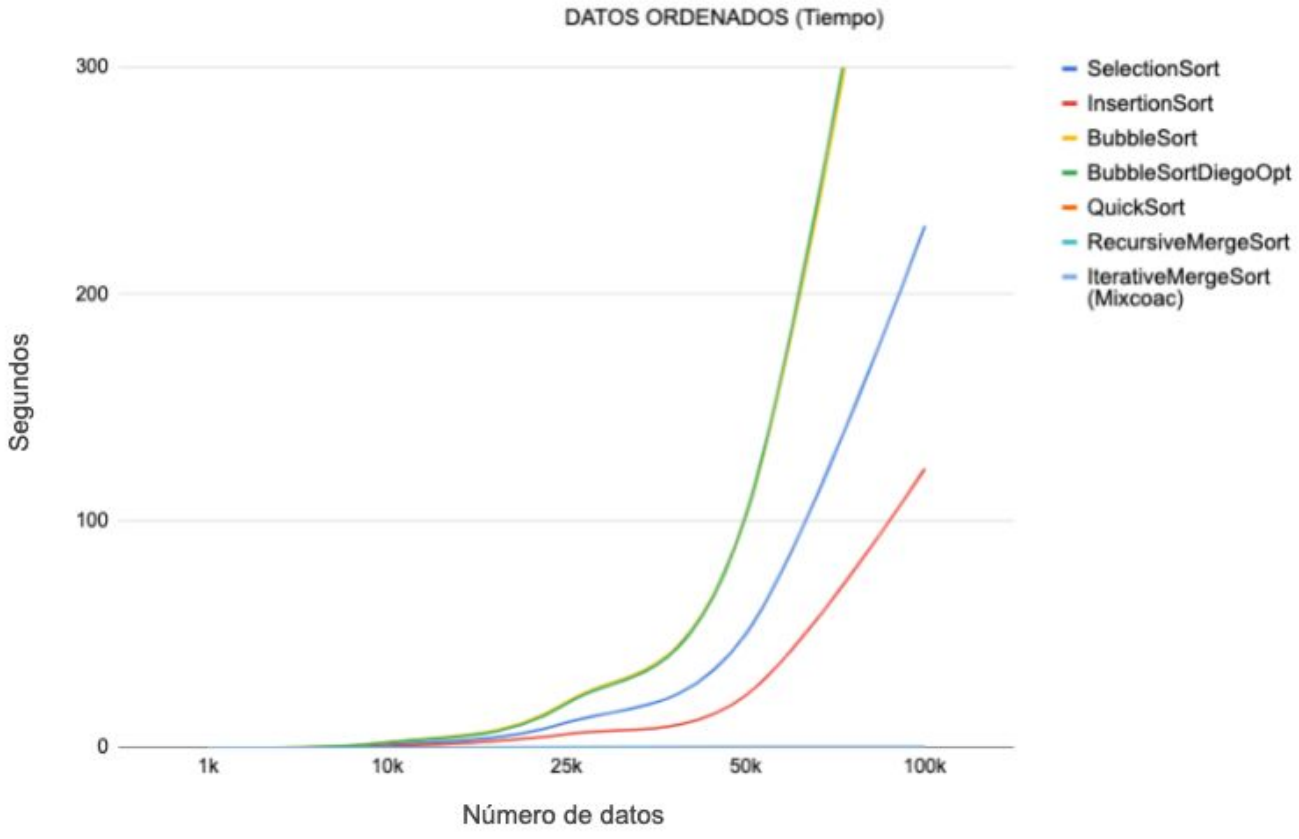
INTRODUCCIÓN

En la presente tarea se realizó una comparación empírica de la eficiencia de los distintos métodos de búsqueda en arreglos computacionales. Para esta experimentación se leyó, en una aplicación de JAVA, un archivo en formato JSON (JavaScript Object Notation) con múltiples objetos que, a su vez, contenían otros objetos y atributos. Se codificó una clase “Business” y una clase específica para leer el archivo, convertir el objeto JSON y ejecutar los distintos métodos de búsqueda. Los parámetros de comparación establecidos fueron el tiempo de ejecución y la cantidad de veces que se repitieron las comparaciones.

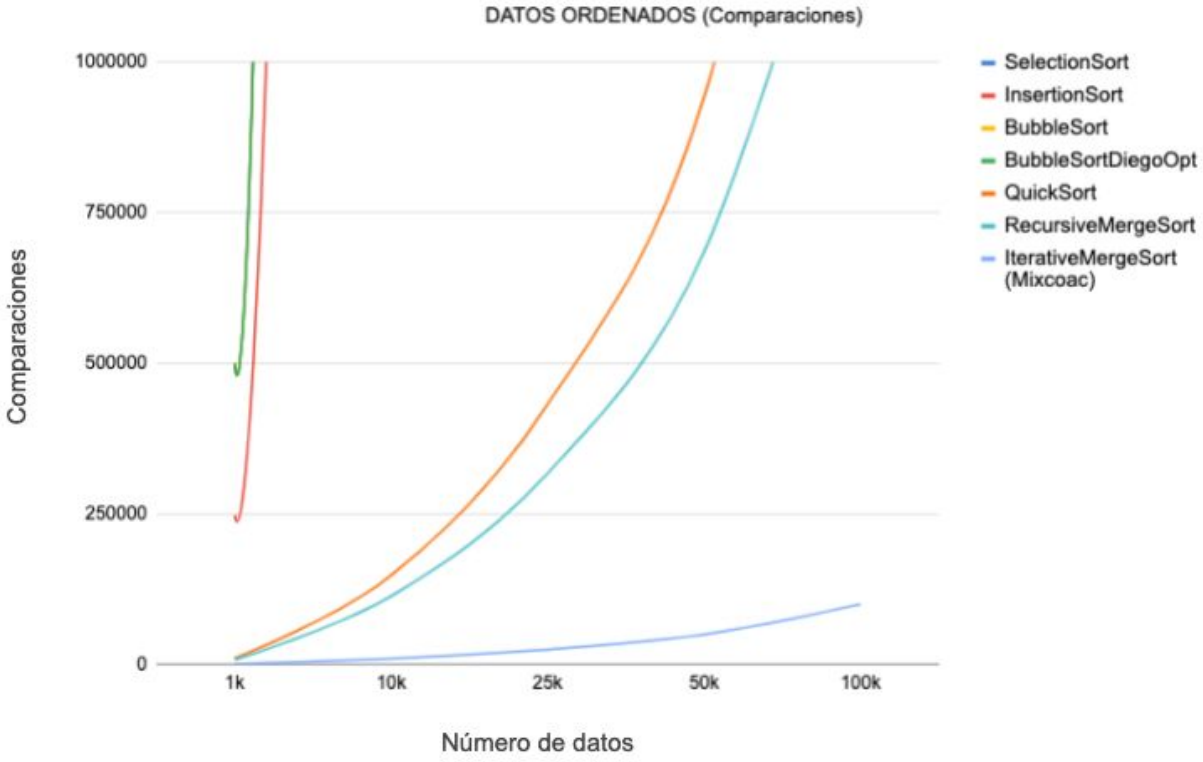
Nota: al percatarme de que algunos métodos se ejecutaban en 0 segundos cuando ordenaban 10,000 datos, decidí crear un archivo con 100,000 datos con el objetivo de tener un rango más amplio de tiempo para que las diferencias fueran más evidentes. Así mismo, implementé una variable de tipo *long* que guardaba el tiempo del momento específico en milisegundos antes de que se ejecutara el método de ordenamiento y, posteriormente, obtenía la diferencia para así obtener los datos precisos en milisegundos y convertirlos a segundos con mayor exactitud.

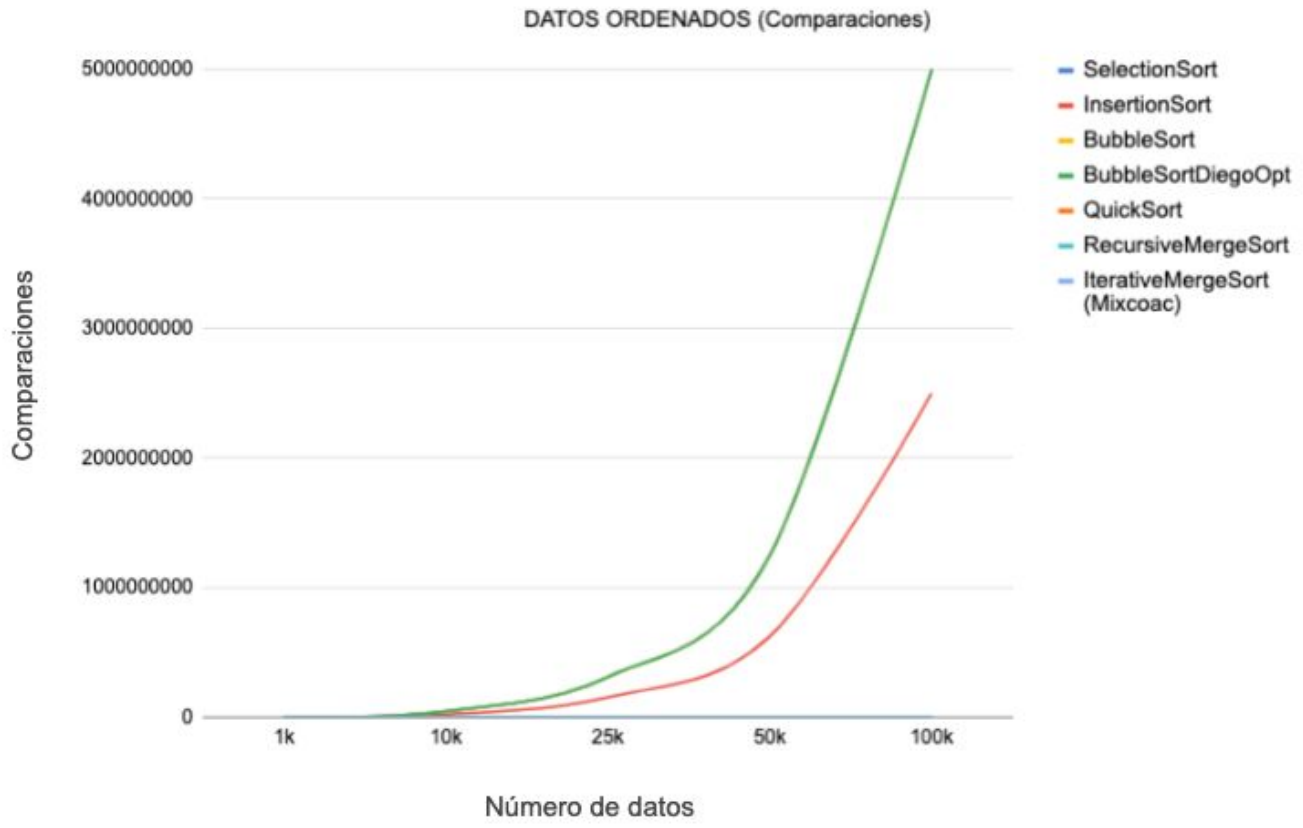
DATOS ORDENADOS

Datos ordenados (TIEMPO en segundos)					
	1k	10k	25k	50k	100k
SelectionSort	0.027	1.2	11	50	230 (3’50’’)
InsertionSort	0.02	0.57	6	23	123 (2’3’’)
BubbleSort	0.038	2.31	20	101.649 (1’45’’)	498 (8’10’’)
BubbleSortDiegoOpt	0.033	2.25	19.321	102 (1’45’’)	509.578
QuickSort	0.003	0.02	0.044	0.077	0.119
RecursiveMergeSort	0.005	0.023	0.074	0.156	0.19
IterativeMergeSort (Mixcoac)	0.004	0.037	0.072	0.133	0.215



Datos ordenados (# CONDICIONES)					
	1k	10k	25k	50k	100k
SelectionSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
InsertionSort	254,244	25,004,850	156,152,794	624,974,254	2,499,823,509
BubbleSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
BubbleSortDiegoOpt	498,554	49,989,747	312,473,472	1,249,966,999	4,999,937,754
QuickSort	11,772	166,273	460,149	986,516	2,228,570
RecursiveMergeSort	7,976	113,612	317,227	684,459	1,468,923
IterativeMergeSort (Mixcoac)	1,000	10,000	25,000	50,000	100,000

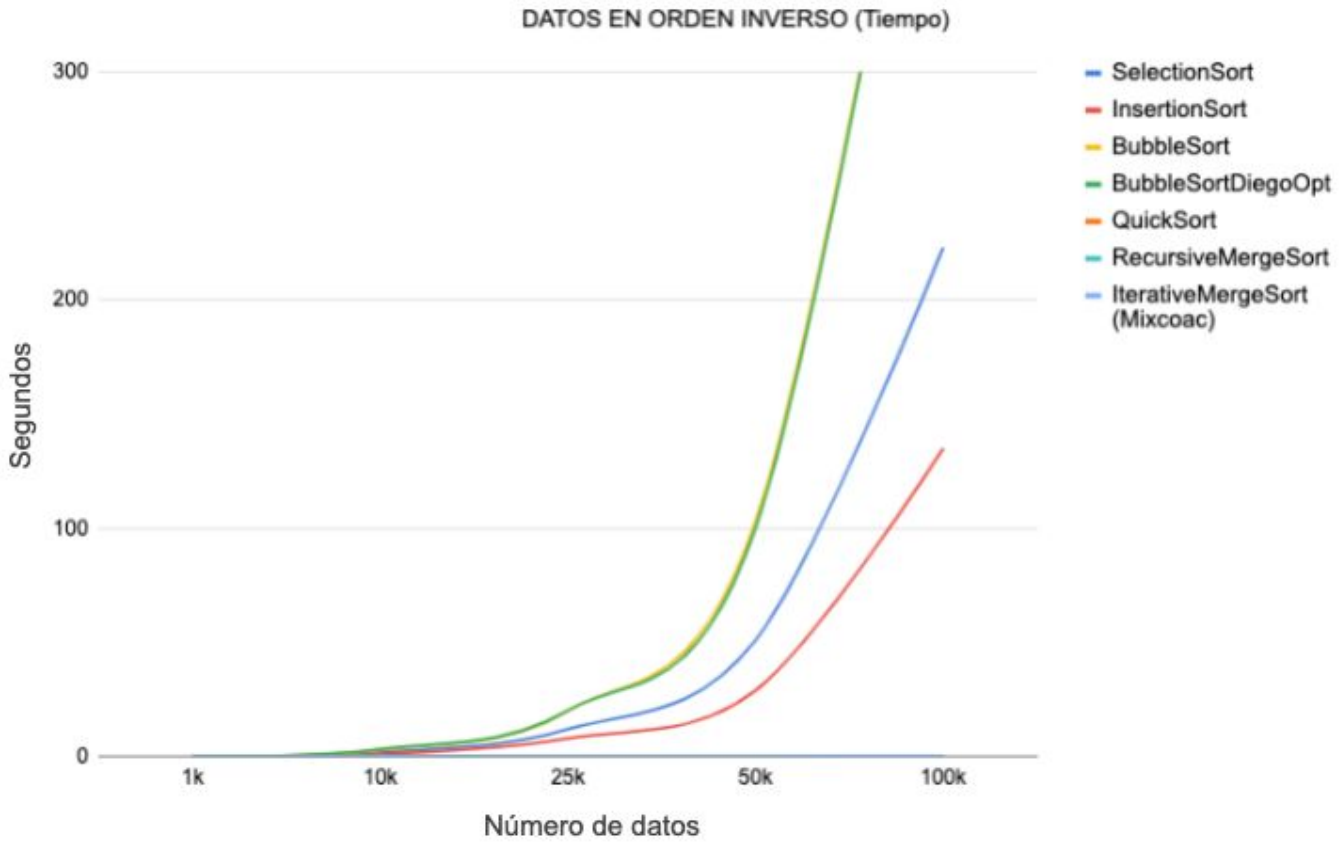




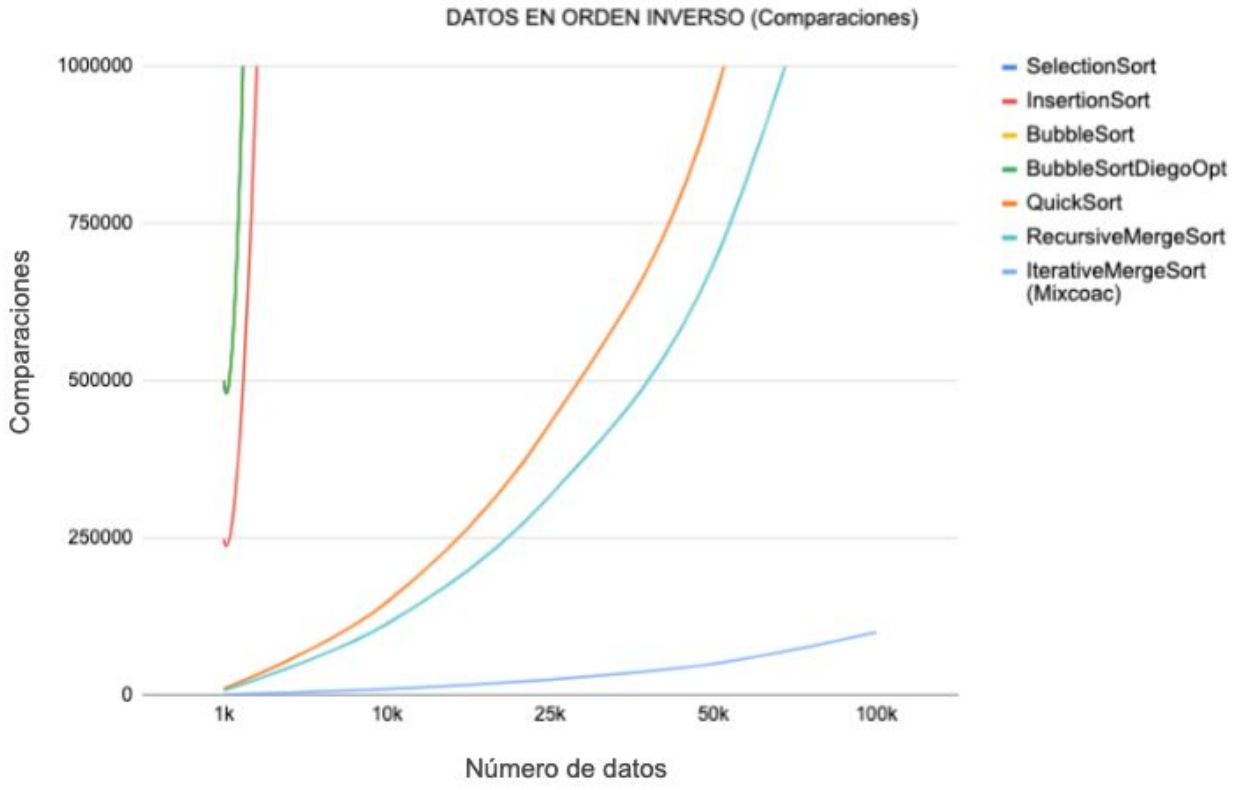
“Bubble Sort “es el método más tardado, seguido de “Selection Sort”, a pesar de que tengan el mismo número de comparaciones. El método que desarrollé de “Bubble Sort” utilizando un ciclo “while” se tarda más que el típico “Bubble Sort”, pero tiene menor número de repeticiones. Es posible que al comparar si la bandera cumple la condición y asignarle un valor cada vez que ingresa al ciclo, provoque un alentamiento del método a pesar de que reduzca el número de comparaciones entre los datos. comparados.

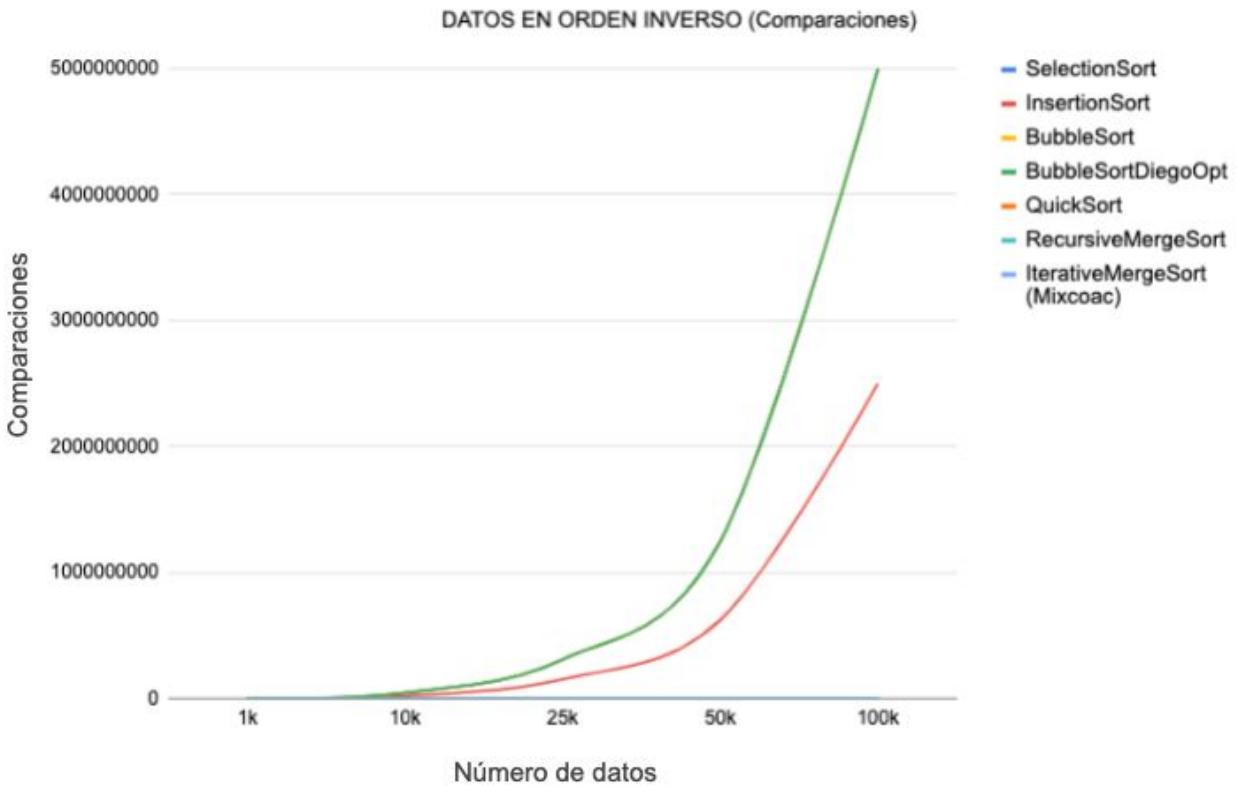
DATOS EN ORDEN INVERSO

Datos en orden inverso (TIEMPO en segundos)					
	1k	10k	25k	50k	100k
SelectionSort	0.03	2	12	51	223
InsertionSort	0.021	1	8	29	135
BubbleSort	0.038	3	20	103	496
BubbleSortDiegoOpt	0.045	3.283	20.277	99.97	494.767
QuickSort	0.002	0.032	0.061	0.089	0.117
RecursiveMergeSort	0.004	0.028	0.081	0.14	0.202
IterativeMergeSort (Mixcoac)	0.003	0.035	0.079	0.141	0.218



Datos en orden inverso (# CONDICIONES)					
	1k	10k	25k	50k	100k
SelectionSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
InsertionSort	247,254	25,010,148	156,364,704	625,000,744	2,499,876,489
BubbleSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
BubbleSortDiegoOpt	499,200	49,989,435	312,458,339	1,249,900,695	4,999,775,655
QuickSort	10,887	147,952	431,631	941,090	2,104,118
RecursiveMergeSort	7,976	113,612	317,227	684,459	1,468,923
IterativeMergeSort (Mixcoac)	1,000	10,000	25,000	50,000	100,000

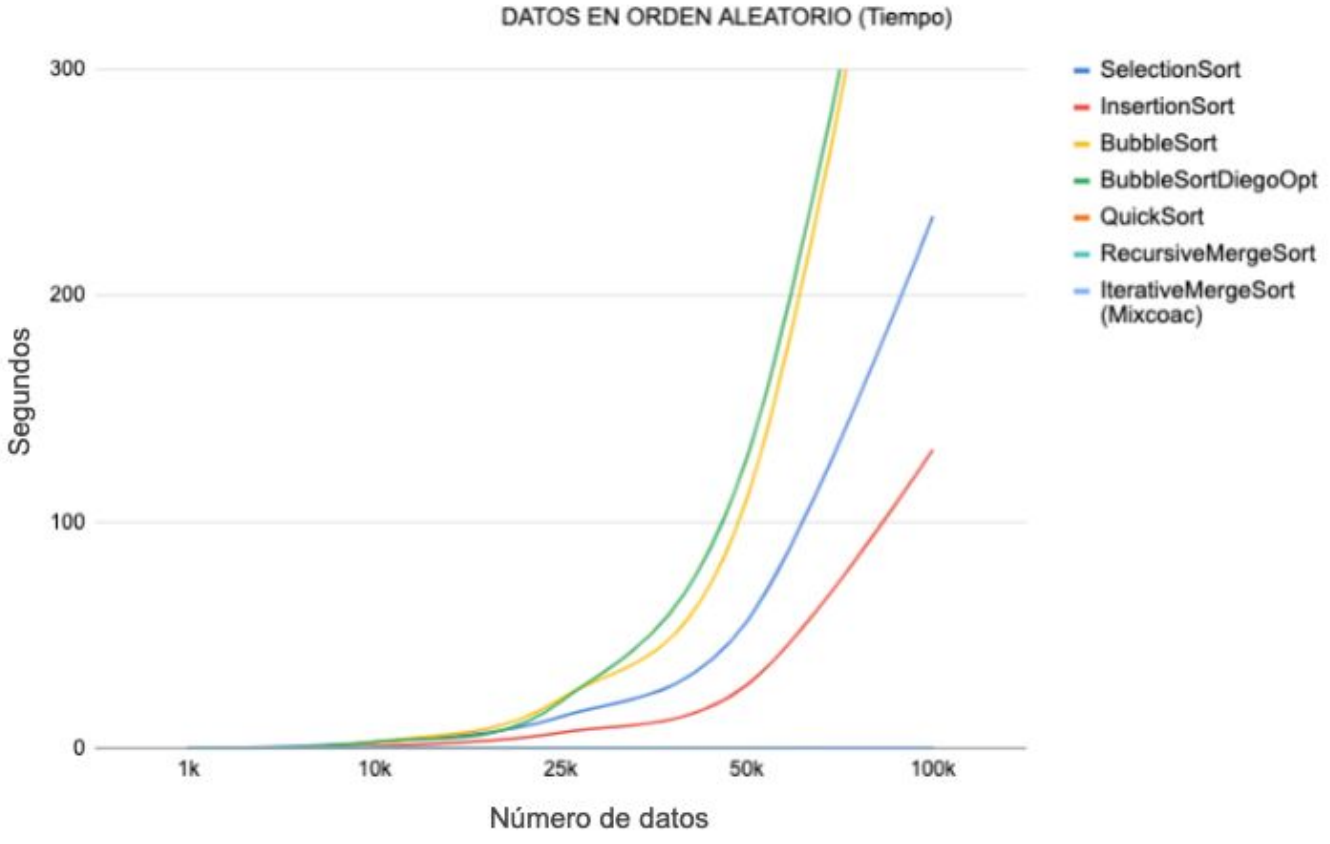




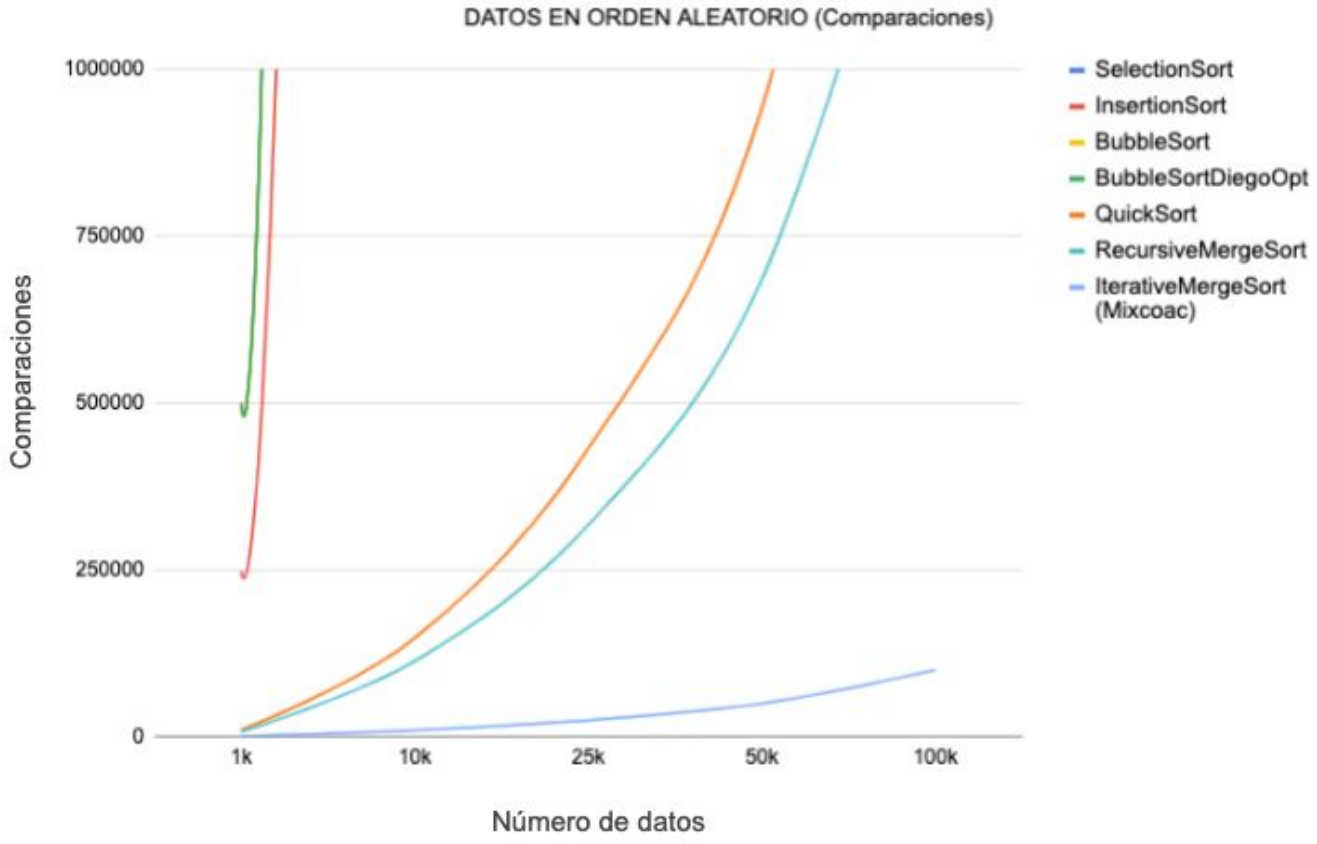
“Insertion Sort” es dos veces más rápido que el “Selection Sort” y cuatro veces más rápido que el “Bubble Sort”.

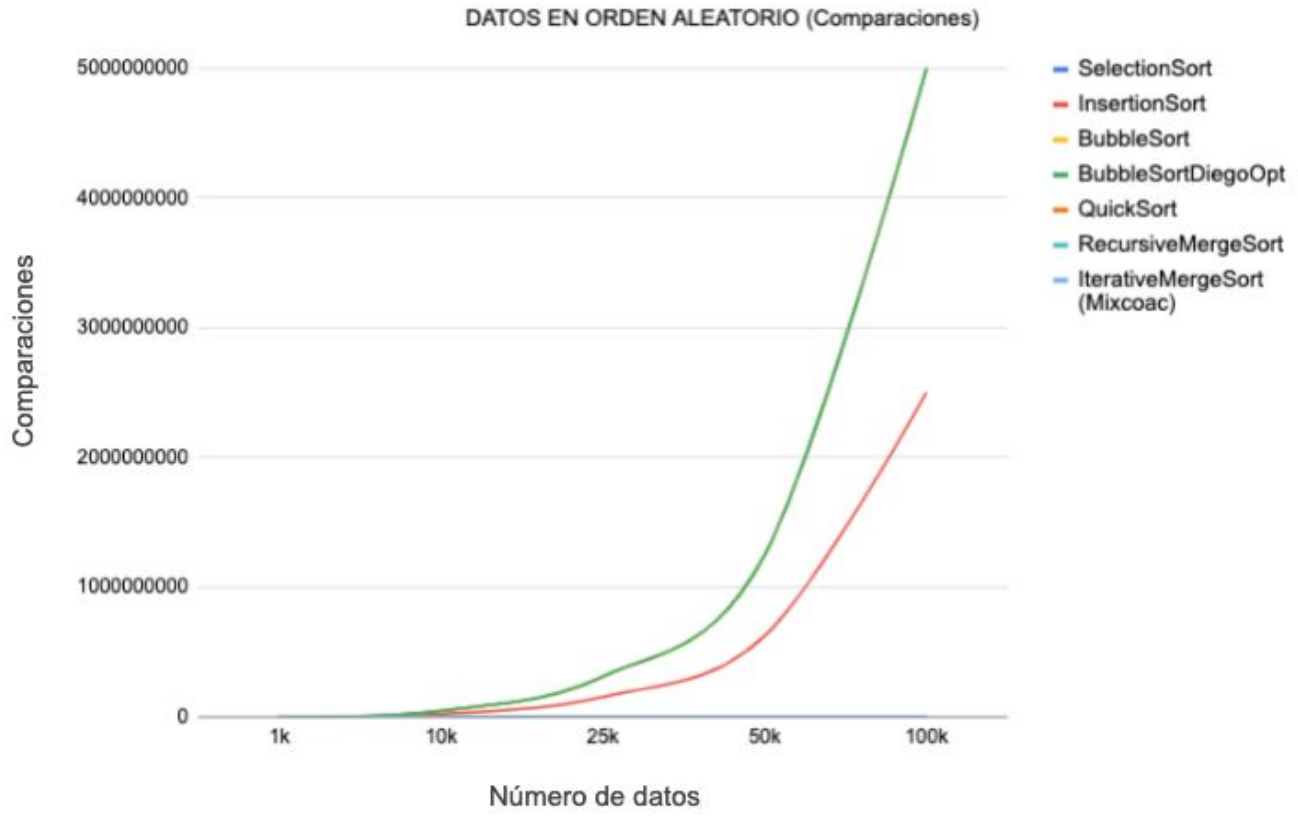
DATOS EN ORDEN ALEATORIO

Datos en orden aleatorio (TIEMPO en segundos)					
	1k	10k	25k	50k	100k
SelectionSort	0.03	3	14	56	235
InsertionSort	0.021	1	7	28	132
BubbleSort	0.032	3	22	110	504
BubbleSortDiegoOpt	0.037	2.8	20.72	128.09	509.679
QuickSort	0.006	0.025	0.063	0.081	0.103
RecursiveMergeSort	0.004	0.028	0.064	0.148	0.226
IterativeMergeSort (Mixcoac)	0.004	0.039	0.086	0.149	0.237



Datos en orden aleatorio (# CONDICIONES)					
	1k	10k	25k	50k	100k
SelectionSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
InsertionSort	245,426	24,952,258	156,365,108	621,797,084	2,500,207,784
BubbleSort	499,500	49,995,000	312,487,500	1,249,975,000	4,999,950,000
BubbleSortDiegoOpt	498,372	49,994,297	312,475,410	1,249,957,609	4,999,875,695
QuickSort	10946	149,427	432,571	921,680	2,151,653
RecursiveMergeSort	7,976	113,612	317,227	684,459	1,468,923
IterativeMergeSort (Mixcoac)	1000	10,000	25,000	50,000	100,000





Si bien el cambio de orden en los tres casos provoca una fluctuación en el tiempo y en el número de comparaciones, ésta es despreciable porque el cambio en las cifras es extremadamente pequeño.

Ahora bien, es importante señalar una última observación: en el número de comparaciones, “Merge Sort Mixcoac” es el más bajo, seguido de “Merge Sort Recursivo” y seguido, a su vez, por “QuickSort”. Sin embargo, en tiempo de ejecución, el “QuickSort” fue más veloz que los otros dos mencionados. Es muy probable que esta incongruencia se deba a la forma en la que están codificados los métodos.

CONCLUSIÓN

Esta tarea ha permitido que visualicemos de una manera más cercana las diferencias, ventajas y desventajas de los diversos algoritmos de ordenamiento vistos en clase. Así mismo, ha permitido que desarrollemos un carácter más crítico frente a la eficiencia de los algoritmos, pues al trabajar con un gran número de datos, la diferencia en tiempo de espera y de la memoria que utiliza la computadora puede ser brutal.

De manera particular en esta tarea, es evidente que los algoritmos más veloces son el “Quick Sort” y los dos tipos de “MergeSort”, así mismo, el algoritmo con menor número de repeticiones

es el “Quick Sort Mixcoac”. Por otro lado, el algoritmo más lento y con mayor número de repeticiones es el “Bubble Sort”.

REFERENCIAS

Lewis, J & Chase, J. (2006) Estructuras de datos con Java. Diseño de estructuras y algoritmos. Pearson-Addison Wesley.

Cormen, T., Leiserson C., Rivest, R., Stein, C. (2009). Introduction to Algorithms. 3a Edición. The MIT Press.

Cairó, O. y Guardati, S. (2006). Estructuras de Datos. 3era. edición. Mc. Graw Hill.