Pablo A. Ruz Salmones
151218

Complejidad algorítmica: algoritmos de ordenamiento

En el presente trabajo se llevará a cabo un breve análisis, basado en evidencia empírica, del tiempo de ejecución de diversos algoritmos, así como del número de operaciones requeridas para llevar a cabo los mismos. La lista de prueba consta de 17770 elementos, mismos que fueron introducidos a los algoritmos de tres maneras distintas: ordenados, en orden inverso y en orden aleatorio. Para cada uno de estos casos, se presentan gráficas y explicaciones comparativas para:

- Bubble sort
- Optimised Bubble sort
- Selection sort
- Insertion sort
- Quick sort
- Merge sort
- Merge sort Mixcoac

Breve explicación de los algoritmos

Bubble sort

El algoritmo de ordenamiento por burbujeo consiste en, para cada elemento del arreglo, verificar si el elemento que le sucede es mayor o menor a él. Si es menor, entonces se intercambian posiciones. Si no, se continúa iterando. Esto garantiza que los elementos más pesados quedarán al final del arreglo.

Complejidad temporal: O(n^2)

Optimised Bubble sort

Si los elementos más pesados ya están al final del arreglo, ¿cuál es el punto de compararlos con un nuevo elemento? El algoritmo por burbujeo optimizado lidia con este problema, y se diferencia del previo en que los ciclos iterativos que se aplican a cada uno de los elementos varían en -1 en longitud por cada vez que se repite el proceso, de suerte que no se revisen aquellos elementos que ya quedaron ordenados.

Complejidad temporal: O(n^2)

Insertion sort

El algoritmo por inserción es considerada una solución viable cuando hay pocos elementos para ordenar, y funciona de la manera siguiente: se trata de ordenar los elementos situados en las posiciones de 0 a p (donde p va de 1 a N-1), en donde p es incrementado en uno en cada etapa. Esto es, cuando la matriz está ordenada de 0 a p, entonces el elemento p+1 tiene que integrarse a un arreglo que ya está ordenado. Tiene que buscar su lugar e incorporarse.

Complejidad temporal: O(n^2)

Pablo A. Ruz Salmones
151218

Quick sort

Este algoritmo consta de los cuatro pasos siguientes:

a) Si el número de elementos de un arreglo S es 0 o 1, entonces regresar.
b) Seleccionar cualquier elemento s en S, denominado pivote.
c) Partir S-{s} en dos grupos distintos, a saber: L={x en S-{s}| x <= s} (esto es, los elementos menores o iguales que pivote), y R = {x en S-{s}|x > s} (los elementos mayores estrictos que pivote)
d) Devolver el resultado de manera recursiva para quicksort(L) y luego quicksort(R).

Este algoritmo tiene la peculiaridad de que el pivote a elegir es un número aleatorio, en tanto que, de no hacerlo así, si el algoritmo está inversamente ordenado, entonces L tiene una dimensión de 0 y no se acaba ganando nada. Por otra parte, es importante hacer notar que, más allá de la parte recursiva, la clave se encuentra en la manera de hacer la partición.

Complejidad temporal: O(nlogn) → Para pivote aleatorio

Merge sort

El algoritmo de ordenación por mezcla consta de tres etapas:

a) Si el número de elementos a ordenar es 0 o 1, entonces volver.
b) Ordenar recursivamente por separado la primera y la segunda mitad del arreglo.
c) Mezclar las dos mitades para obtener el total ordenado.

Complejidad temporal: O(nlogn)

Merge sort Mixcoac

La principal diferencia entre este algoritmo y el previo se encuentra en la partición del arreglo. Mientras que Merge sort lo lleva a cabo de manera recursiva, la variante Mixcoac comienza considerando a cada elemento del arreglo mayor como un “arreglito” de dimensión 1, mismo que va creciendo. Por ello, existe un ahorro de llamadas recursivas y, como consecuencia, del tiempo de ejecución del algoritmo y sobre todo de la memoria utilizada pues, si se hace adecuadamente, no es necesario crear más que un arreglo auxiliar para llevar a cabo el ordenamiento.

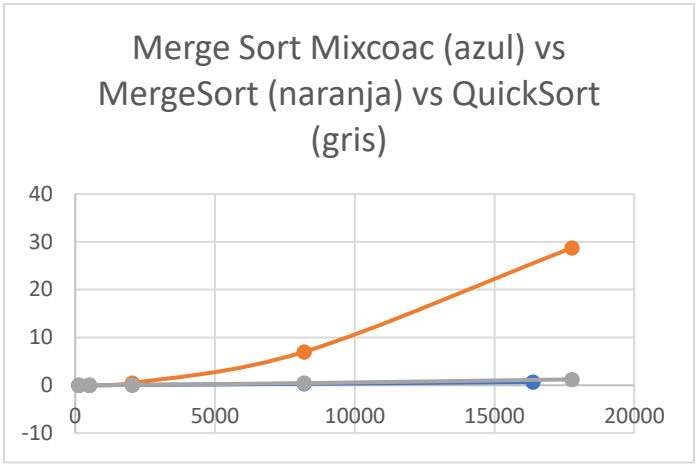
Complejidad temporal: O(nlogn)

1. Ordenamiento Inverso

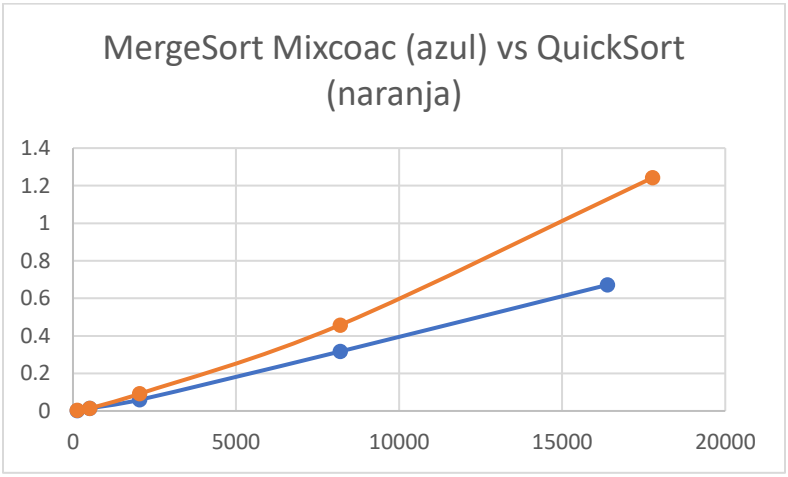
Resultados gráficos (análisis temporal [A])

A continuación se presentan las gráficas que contrastan número de elementos contra tiempo para cada uno de los algoritmos mencionados previamente.

Pablo A. Ruz Salmones
151218

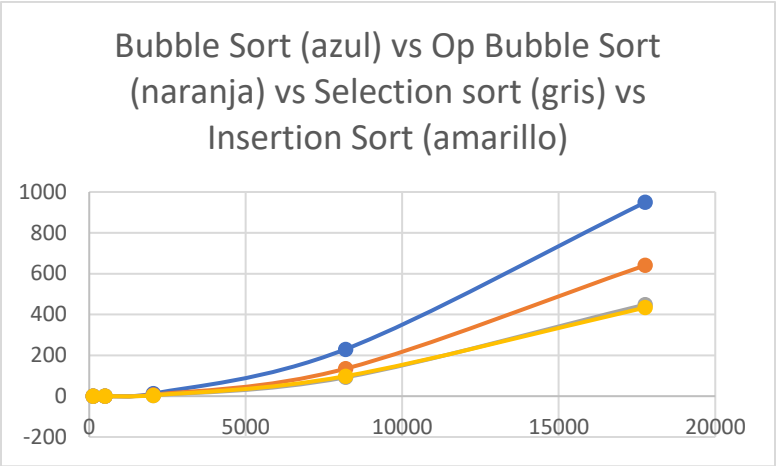


En esta figura se aprecia claramente la superioridad en el desempeño temporal del Mixcoac y el QuickSort sobre el Mergesort común y corriente. Es posible que esta diferencia se deba a la cantidad de memoria que la computadora se ve forzada a crear para operar el Mergesort. En efecto, en el caso de los otros dos algoritmos, la creación de arreglos es mínima. De hecho, en el Mixcoac es nula.

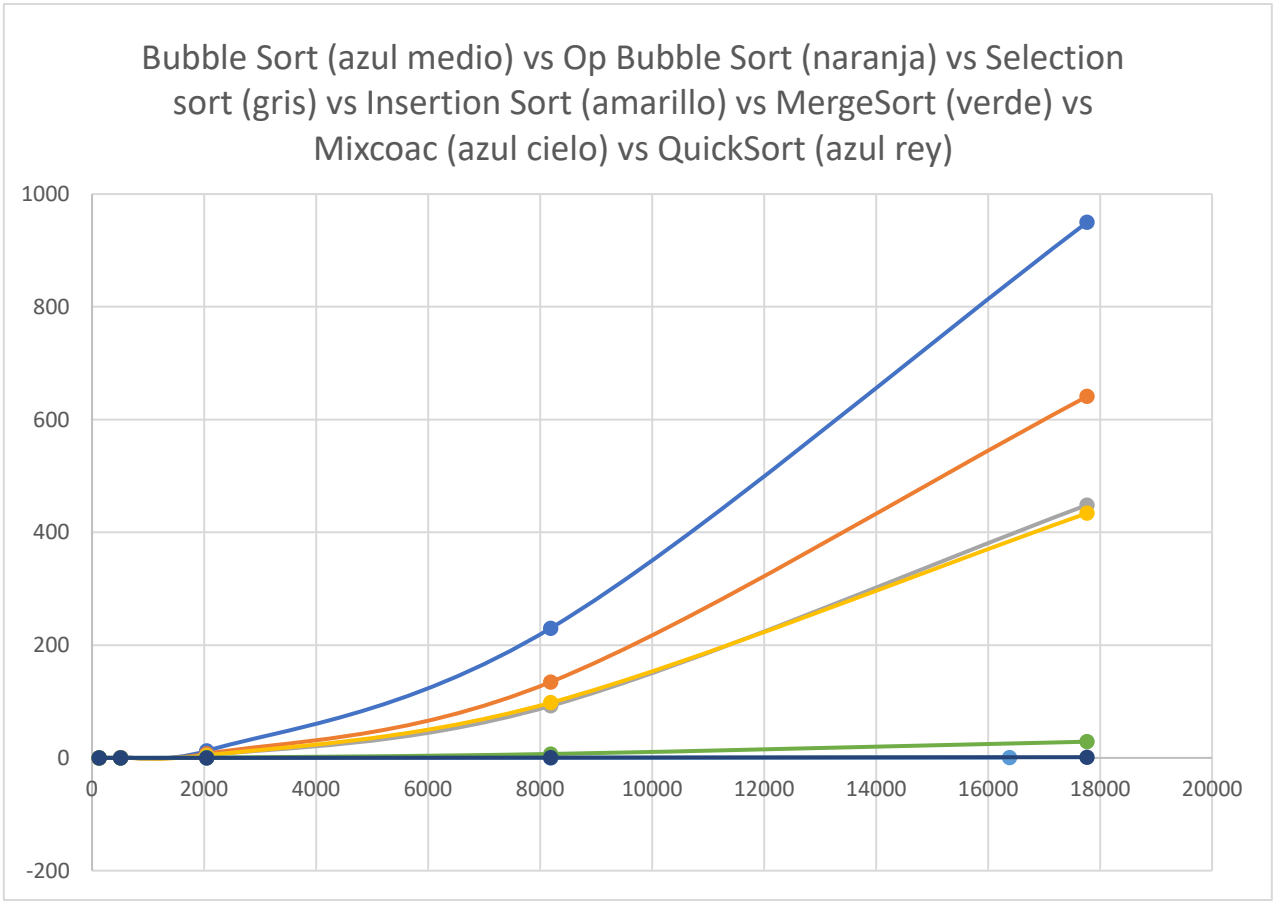


Aunque en el gráfico Mixcoac vs MergeSort vs QuickSort es difícil apreciar la diferencia de desempeño entre el primero y el tercero, aquí se puede observar claramente que sí hay una superioridad del Mixcoac sobre el QuickSort.

Pablo A. Ruz Salmones
151218



Finalmente, aquellos algoritmos que tienen una complejidad O(n^2) son claramente inferiores, pero entre sí también existen diferencias. La primera diferencia notable es la existente entre el BubbleSort normal y la versión optimizada del mismo. No obstante, ambos resultan inferiores al algoritmo por inserción y al algoritmo por selección, que entre sí son muy parecidos.

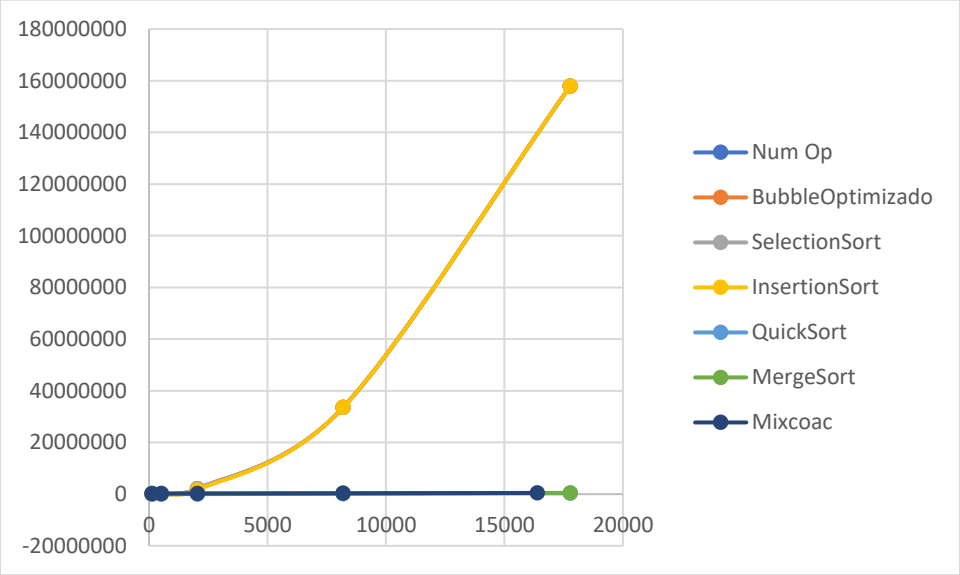


En la última figura se aprecia claramente la diferencia que existe en el tiempo de ejecución de cada uno de los algoritmos. Véase cómo obedece perfectamente la tendencia O(n^2) para

BubbleSort, Optimised Bubble Sort, Selection Sort e Insertion Sort, mientras que para el resto se observa la tendencia O(n log n). Los algoritmos podrían, así, dividirse en dos bloques.

Resultados gráficos: análisis de operaciones [B]

Aquí vale la pena presentar un único gráfico con todos los algoritmos en él, pues resulta muy impactante observar la diferencia clara entre el comportamiento el grupo O(n^2) y del O(nlogn)

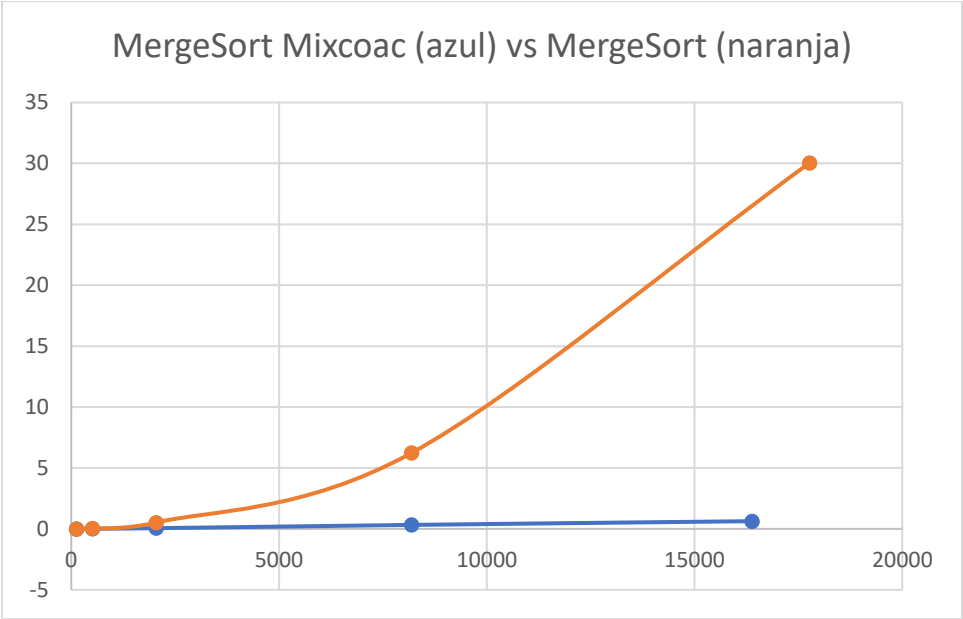


Obsérvese cómo desde el Bubble Sort (NumOp en la nomenclatura de la gráfica) hasta el Insertion Sort, crecen todos ellos al mismo ritmo, mientras que el QuickSort, MergeSort y Mixcoac se mantienen casi pegados al eje x. Así pues, no es de extrañarse que la diferencia en los tiempos de ejecución sea tan marcada. Se aprecia claramente que existe una correlación muy alta entre el número de operaciones que tiene que realizar la computadora para ordenar el algoritmo y el tiempo que le tomará hacerlo.

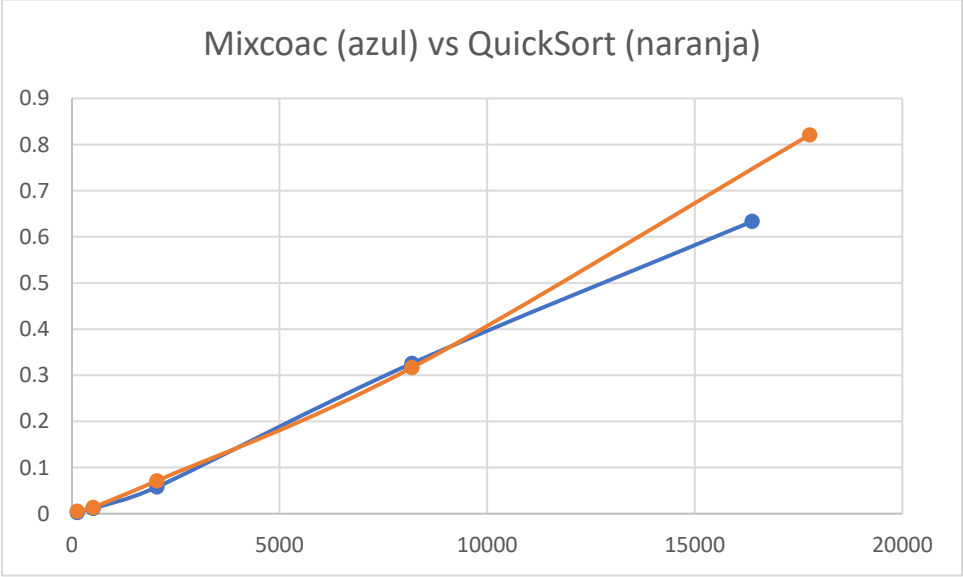
Pablo A. Ruz Salmones
151218

2. Promedio de ordenamientos aleatorios

Resultados gráficos: promedios temporales [A]

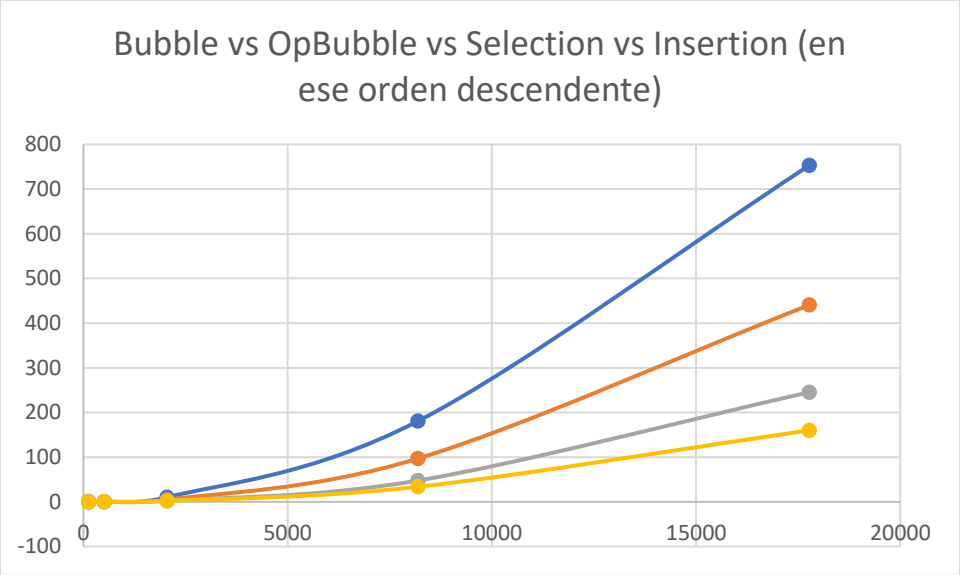


Una vez más, aún cuando los ordenamientos sean aleatorios, se muestra un promedio de tiempo de ejecución del Mixcoac muy por debajo del MergeSort normal, comparados únicamente entre ellos.

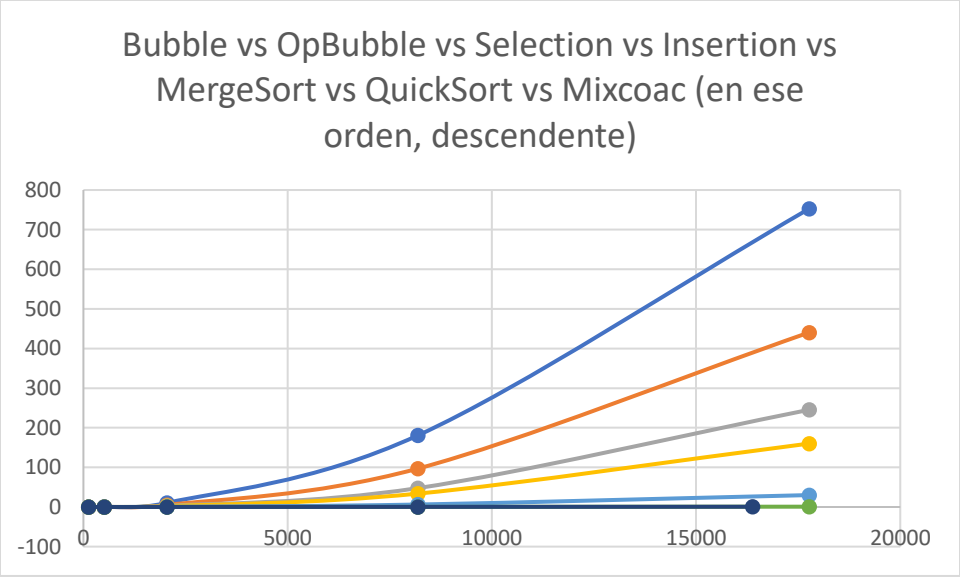


Aquí hay una diferencia clave con respecto al orden inverso. En la gráfica Mixcoac vs QuickSort del orden inverso, el Mixcoac se muestra claramente superior para cada valor de los datos. En este caso, sin embargo, aunque se puede apreciar una tendencia ganadora, a lo largo de la trayectoria se observa una variación en cuanto a la eficiencia de los algoritmos.

Pablo A. Ruz Salmones
151218



Obsérvese que incluso dentro de la categoría O(n^2) hay variación. En efecto, el ordenamiento por burbujeo optimizado tiene una eficiencia considerablemente mayor con respecto a su hermano y, al igual que en el caso anterior, el selection sort y el insertion sort son mejores que los de burbujeo.

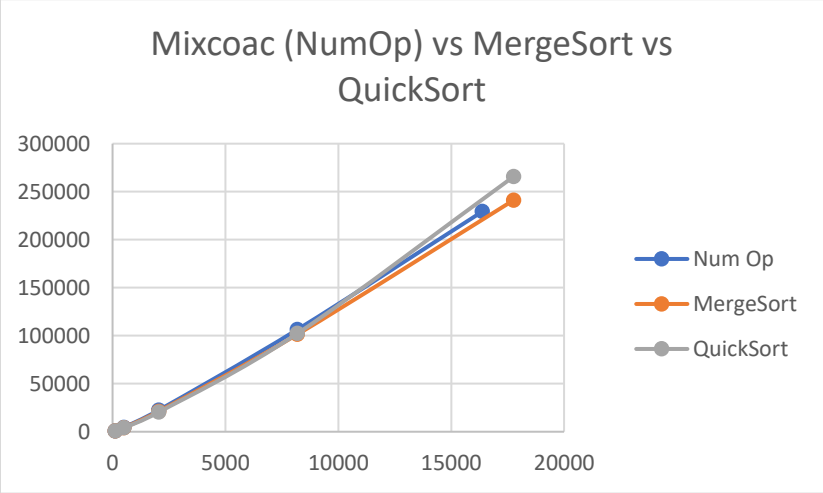


En esta gráfica final pueden apreciarse perfectamente los dos “bloques” de algoritmos: aquellos que tiene una complejidad O(n^2) y los O(nlogn)

Pablo A. Ruz Salmones
151218

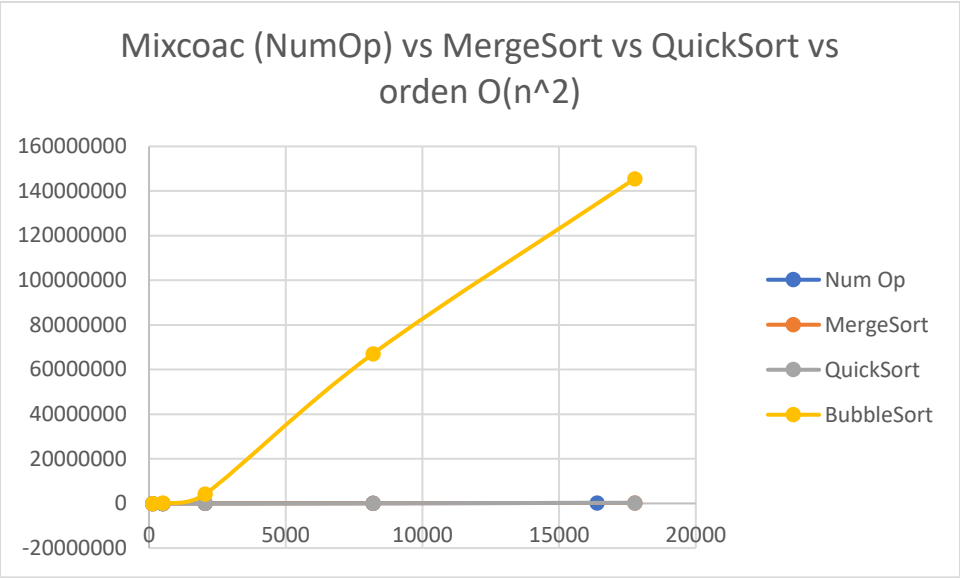
Resultados gráficos: número de operaciones [B]

A continuación se presenta el resultado gráfico del número de operaciones para los algoritmos O(nlogn), a saber: Mixcoac, QuickSort y MergeSort



Obsérvese que de esta gráfica previa se obtiene una conclusión por demás interesante. El Mixcoac (NumOp) no es el que realza una menor cantidad de operaciones de comparación y, sin embargo, resulta ser el más veloz. Esto quiere decir que la diferencia entre uno y otro radica probablemente en la manera en la que se llevó a cabo la integración de los arreglos. Es decir, el número de iteraciones es casi idéntico y, sin embargo, la creación de arreglos de apoyo para el MergeSort causó que el tiempo de ejecución del mismo fuese mayor, en comparación con el Mixcoac. De cualquier manera, se nota perfectamente que los tres siguen una misma tendencia.

Compárense ahora con los algoritmos O(n^2):

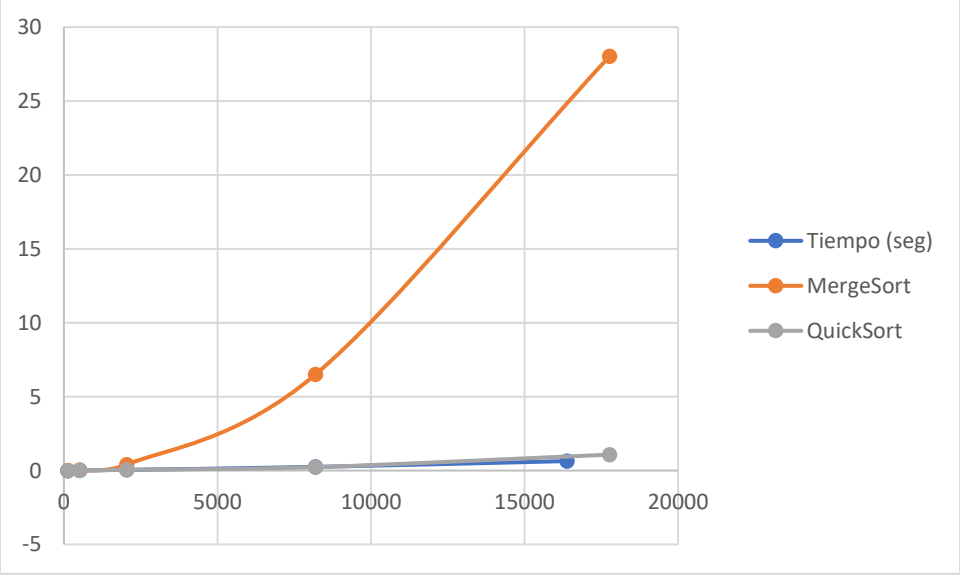


Aunque la diferencia no es tan abismal como en el caso de orden inverso (obsérvese que el Bubble Sort se encunetra casi 2 000 000 de operaciones por debajo del orden inverso), sigue siendo en extremo notoria con respecto al resto de los algoritmos O(nlgn)

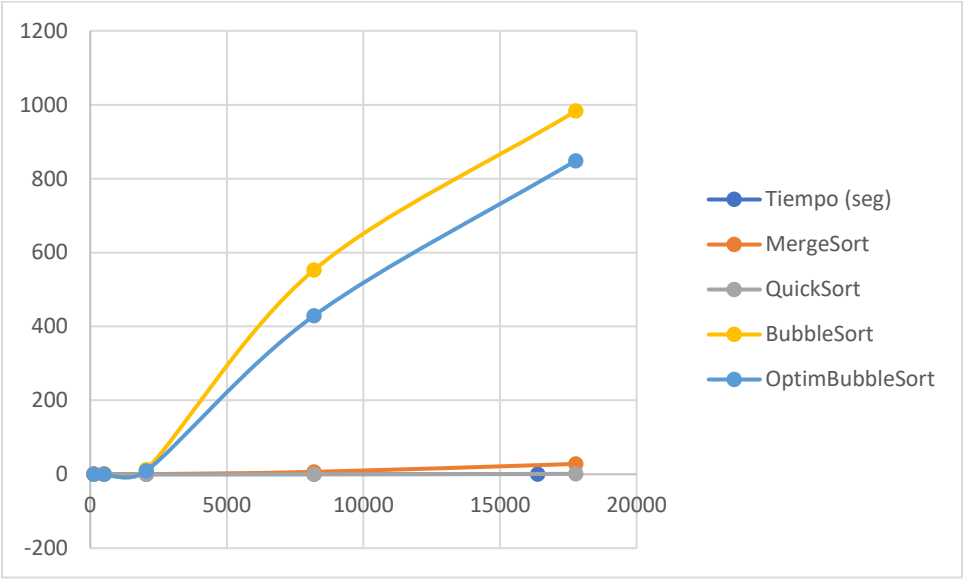
3. Ordenamiento total

Resultado gráfico: eficiencia temporal [A]

A continuación se muestra la comparación de eficiencia temporal entre los tres algoritmos de ordenamiento O(nlogn)

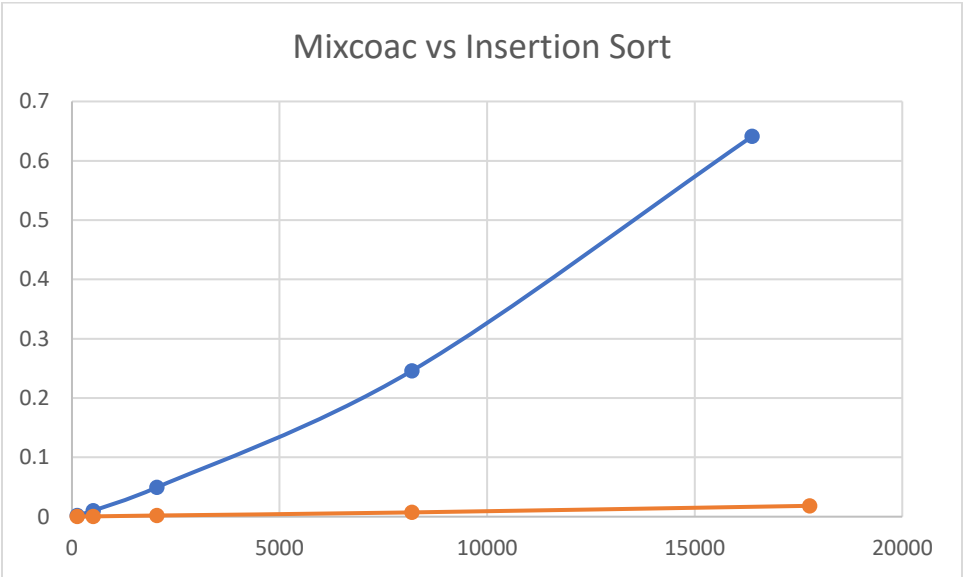


Una vez más, el Mixcoac (Tiempo(seg)) supera a los otros dos. Nótese cómo a partir de cierto valor el Merge Sort se dispara. El QuickSort, sin embargo, se mantiene a la misma altura que el Mixcoac.

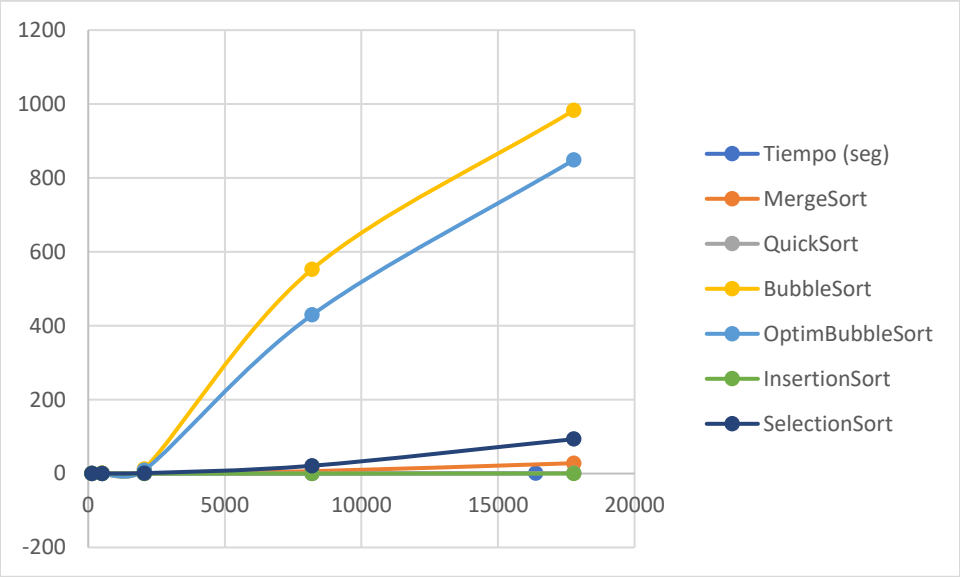


Sin embargo, aunque el MergeSort es inferior a sus dos compañeros, nótese la enorme diferencia que existe en comparación con el BubbleSort y el Optimised Bubble Sort. Es únicamente para valores muy bajos que todos los algoritmos tienen una complejidad similar.

Antes de presentar la gráfica temporal final para el ordenamiento total, obsérvese la siguiente figura:



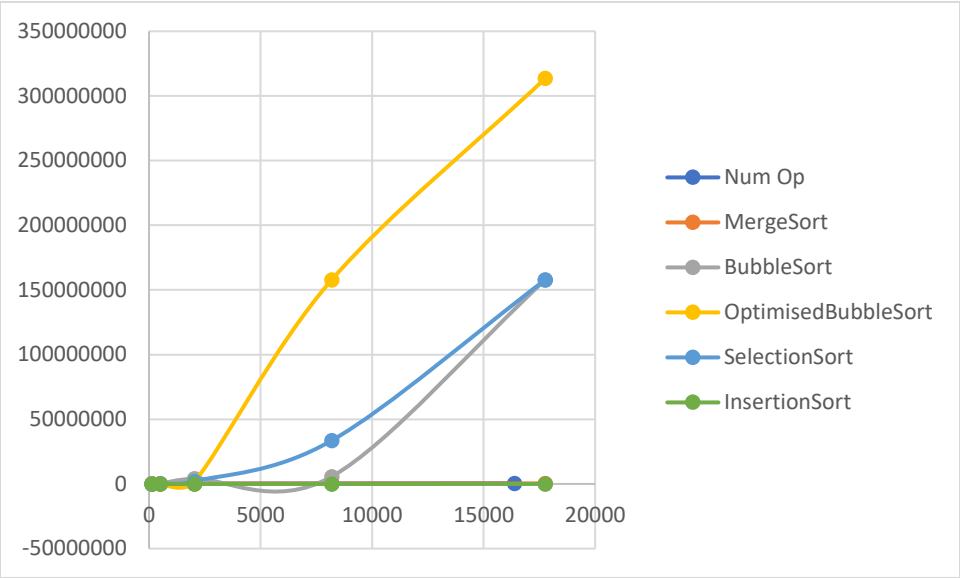
A diferencia de lo que ocurre en los casos anteriores, aquí el Insertion Sort es, por mucho, el más rápido de todos los algoritmos. En efecto, si la lista ya se encuentra ordenada, el insertion sort no entra a su segundo ciclo, pues la condición falla. Por ello, el número de comparaciones que hace es mínimo.



Véase como el hecho de que el algoritmo esté ordenado, modifica completamente el rendimiento de los diferentes algoritmos, sobre todo de aquellos que pertenecen al que hemos denominado grupo O(n^2). Aunque el Bubble Sort y el Optimised Bubble Sort mantienen su total ineficiencia, el insertion sort y el selectionSort mejoran sobremanera. De hecho, como vimos previamente, el más rápido para verificar el ordenamiento de un determinado arreglo es el InsertionSort.

Resultado gráfico: análisis de operaciones[B]

La gráfica de abajo representa el número de operaciones que cada algoritmo realiza de acuerdo con la cantidad de datos ingresados.



Obsérvese que los Bubble siguen siendo los más ineficientes. No obstante, el insertionSort se convierte, al igual que en el análisis temporal, en el mejor algoritmo, y por la misma razón ya enunciada: la condición de entrada al segundo bucle falla para cada elemento.

Anexos

A continuación se muestran los anexos del análisis de Excel y posteriormente las referencias bibliográficas de la introducción.

Orden inverso

Merge sort

Datos	Tiempo(seg)	Num op
128	0.004724085	242116
512	0.038625628	245828
2048	0.473694856	263748
8192	6.997980417	347716
17770	28.73656701	492772

Bubble sort

Datos	Tiempo (seg)	Num Op
128	0.060348605	8128
512	0.705558055	130816
2048	12.48380792	2096128

8192	230.1529724	33550336
17770	949.9609088	157877565

Bubble sort optimizado

Datos	Tiempo (seg)	Num Op
128	0.025802203	8128
512	0.428011433	130816
2048	7.189389909	2096128
8192	134.7024813	33550336
17770	641.1138983	157877565

Selection sort

Datos	Tiempo (seg)	Num Op
128	0.016769282	8256
512	0.281034238	131328
2048	4.409109717	2098176
8192	92.30170346	33558528
17770	448.5891736	157895335

Insertion sort

Datos	Tiempo (seg)	Num Op
128	0.01962474	8128
512	0.197788229	81280
2048	4.232616513	1642571
8192	97.99384013	33550336
17770	434.0266889	157877565

Quick sort

Datos	Tiempo (seg)	Num Op
128	0.00313392	814
512	0.014386721	4998
2048	0.091825085	27678

8192	0.458029651	142726
17770	1.242221991	351150

Promedio de aleatorios

Merge sort Mixcoac

Datos	Tiempo (seg)	Num Op
128	0.002274108	896
512	0.011734004	4608
2048	0.057839026	22528
8192	0.325869571	106496
16384	0.6333218	229376

Merge sort

Datos	Tiempo(seg)	Num op
128	0.004096324	815
512	0.037215242	4298
2048	0.524590404	21259
8192	6.235900412	101419
17770	30.04360228	241220

Bubble sort

Datos	Tiempo(seg)	Num op
128	0.05239094	16256
512	0.63486678	261632
2048	10.35071963	4192256
8192	180.8245503	67100672
17770	752.7763006	145554070

Bubble sort optimizado

Pablo A. Ruz Salmones
151218

Datos	Tiempo(seg)	Num op
128	0.019362889	
512	0.324640918	
2048	4.982102702	
8192	96.99820368	
17770	440.4947827	

Selection sort

Datos	Tiempo(seg)	Num op
128	0.013130222	
512	0.196698319	
2048	2.982321075	
8192	47.6811594	
17770	245.3615928	

Insertion sort

Datos	Tiempo(seg)	Num op
128	0.008476485	
512	0.149964068	
2048	2.098095857	
8192	33.77328786	
17770	160.1216342	

Quick sort

Datos	Tiempo(seg)	Num op
128	0.005244371	722
512	0.013861063	4127
2048	0.071390361	20389
8192	0.316638308	102370
17770	0.821052421	265835

Orden total

Pablo A. Ruz Salmones
151218

Merge sort Mixcoac

Datos	Tiempo (seg)	Num Op
128	0.002054761	242116
512	0.010299694	245828
2048	0.049501846	263748
8192	0.246107631	347716
16384	0.641153185	470596

Merge sort

Datos	Tiempo(seg)	Num op
128	0.003335199	241668
512	0.031047107	243524
2048	0.410184597	252484
8192	6.502273622	294468
17770	28.03433345	369747

Bubble sort

Datos	Tiempo (seg)	Num Op
128	0.060133251	16256
512	0.800053152	261632
2048	12.33933884	4192256
8192	552.6892624	5666048
17770	982.8383987	157877565

Bubble sort optimizado

Datos	Tiempo (seg)	Num Op
128	0.019415679	8128
512	0.35134986	130816
2048	8.880240103	2096128
8192	429.2448624	157877565
17770	848.4897247	313659002

Pablo A. Ruz Salmones
151218

Selection sort

Datos	Tiempo (seg)	Num Op
128	0.003871117	8256
512	0.065108938	131328
2048	1.208600193	2098176
8192	21.12924521	33558528
17770	93.10303114	157895335

Insertion sort

Datos	Tiempo (seg)	Num Op
128	0.000170498	127
512	0.000477783	511
2048	0.001987832	2047
8192	0.007220485	8191
17770	0.018319903	17769

Quick sort

Datos	Tiempo (seg)	Num Op
128	0.002014701	241742
512	0.010850757	244304
2048	0.050651358	257618
8192	0.233983763	323156
17770	1.076385205	439479

Referencias bibliográficas

- *Estructuras de datos en Java.* Allen Weiss, M. 2013. Pearson Education. 4a edición. Madrid, España.
- *Algoritmos a Fondo con implementaciones en C y en Java.* Sznajdleder, Pablo A. 2012. AlfaOmega. 1ª Edición. México.