Comparación de Algoritmos de Ordenamiento

Se comparó el desempeño de cinco algoritmos de ordenamiento: *Selection Sort, Insertion Sort, Bubble Sort, Quick Sort,* y *Merge Sort* en cuanto al número de comparaciones realizadas y el tiempo de ejecución.

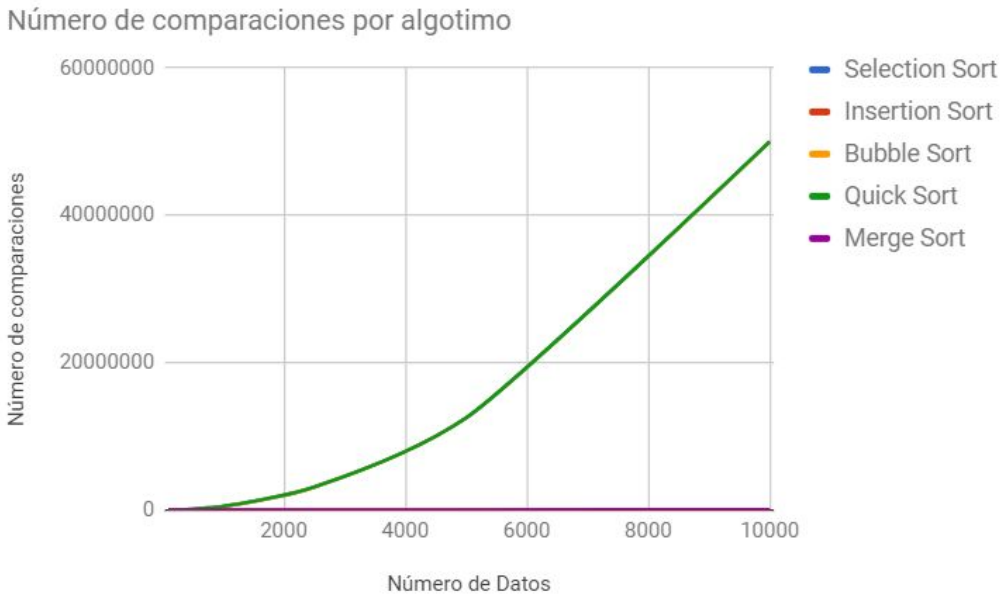
Las comparaciones se realizaron para tres escenarios diferentes: con los datos ordenados desde un comienzo, con los datos dados en orden inverso, y con los datos dados en un orden aleatorio.

Los datos que se usaron son títulos de un banco de películas.

Con los datos en orden

Comparaciones	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Insertion Sort	99	249	499	749	999	2459	4999	9999
Bubble Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Quick Sort	5148	31623	125748	282373	501498	3128748	12507498	50014998
Merge Sort	317	985	2221	3470	4942	13680	29858	64734

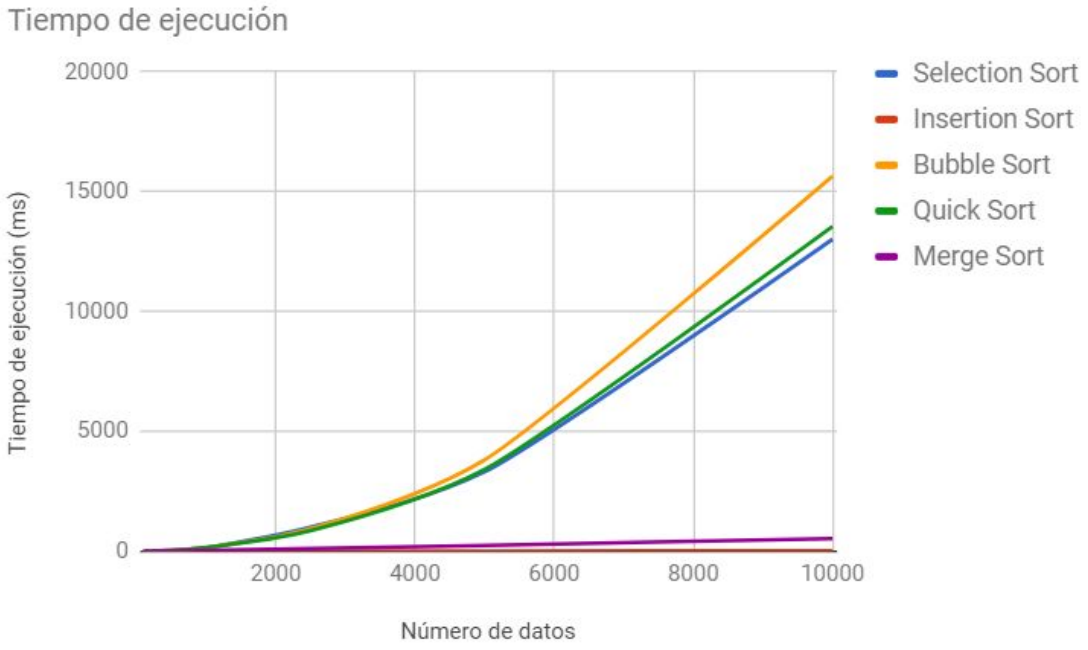
Tabla 1: Número de comparaciones por algoritmo en un arreglo ordenado.



Gráfica 1: Número de comparaciones por algoritmo en un arreglo ordenado.

Tiempo de ejecución (ms)	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	2	10	38	84	137	1013	3315	13020
Insertion Sort	0	1	1	2	2	4	9	15
Bubble Sort	1	9	34	82	146	939	3813	15654
Quick Sort	3	11	42	86	157	863	3416	13548
Merge Sort	10	12	18	28	42	119	245	527

Tabla 2: Tiempo de ejecución por algoritmo en un arreglo ordenado.



Gráfica 2: Tiempo de ejecución por algoritmo en un arreglo ordenado.

Para los algoritmos *Selection Sort, Bubble Sort, Quick Sort* el comportamiento es muy parecido en cuanto al número de comparaciones que hacen, pues los tres tienen un comportamiento cuadrático. En la Gráfica 1 se nota como las curvas de estos tres algoritmos se enciman.

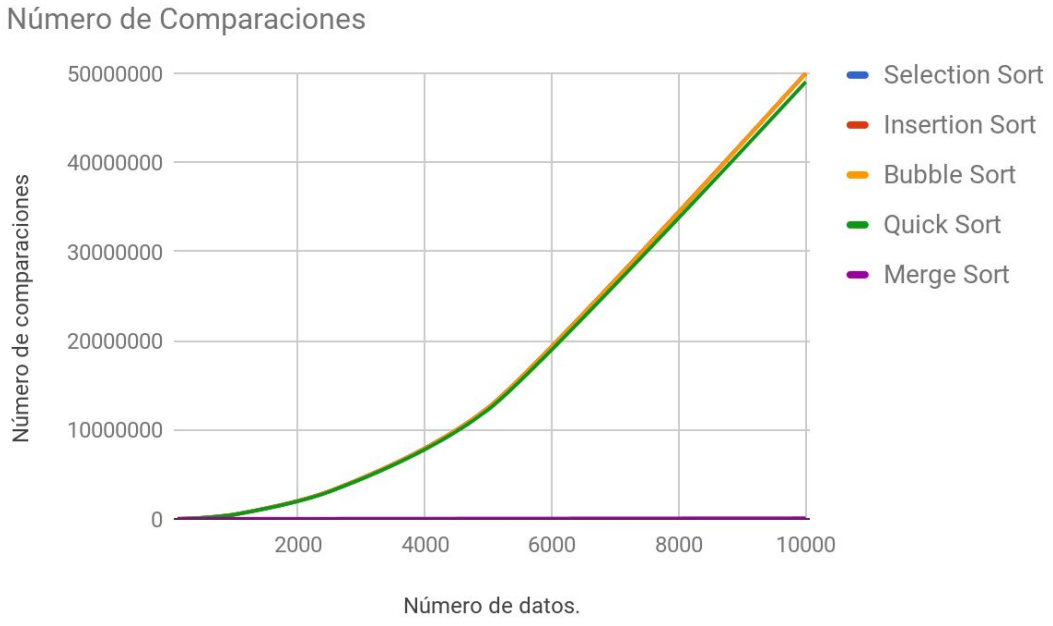
Por otro lado *Merge Sort* tiene un comportamiento muy parecido a lo esperado de este algoritmo, haciendo un número de comparaciones del orden O(NlgN). Y insertion sort hace muy pocas operaciones cuando el arreglo está ordenado, esto ocurre porque en cada iteración del algoritmo sólo se compara cada elemento con el anterior, por lo que hace O(N) comparaciones.

En la Gráfica 2 se ve una mayor diferencia entre los cinco algoritmos, con *Selection y Bubble Sort* teniendo un comportamiento muy parecido, mientras que *Quick Sort* tomá un poco más de tiempo. Esto es interesante pues aunque los tres algoritmos hacen $O(N^2)$ comparaciones, sus tiempos de ejecución son ligeramente diferentes.

Con los datos en orden inverso

Comparaciones	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Insertion Sort	4949	31123	124748	280873	499492	3123738	12497485	49994969
Bubble Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Quick Sort	5016	30613	123069	276315	488152	3066873	12270493	49017832
Merge Sort	356	1011	2272	3769	5044	14752	32004	69008

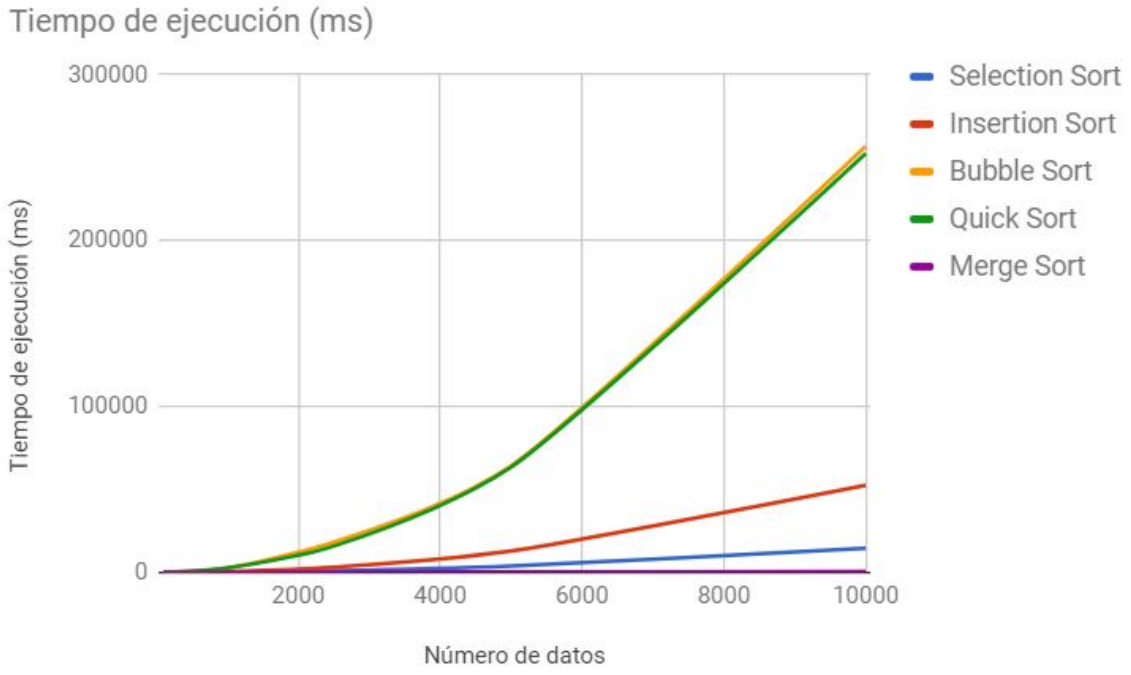
Tabla 3: Número de comparaciones por algoritmo en un arreglo con los datos en orden inverso.



Gráfica 3: Número de comparaciones por algoritmo en un arreglo con los datos en orden inverso.

Tiempo de ejecución (ms)	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	2	10	38	83	153	988	3900	14706
Insertion Sort	5	31	129	290	503	3204	12934	52453
Bubble Sort	24	163	670	1462	2594	18168	64189	256790
Quick Sort	37	190	754	1503	2658	15882	63514	252485
Merge Sort	2	8	18	28	40	119	245	532

Tabla 4: Tiempo de ejecución por algoritmo en un arreglo con los datos en orden inverso.



Gráfica 4: Tiempo de ejecución por algoritmo en un arreglo con los datos en orden inverso.

Lo primero que se puede notar es que *Selection Sort, Insertion Sort, y Quick Sort* que en la prueba anterior tenían un comportamiento cuadrático en cuanto al número de comparaciones, mantienen ese comportamiento. Pero además, *Insertion Sort,* que en la prueba anterior tenía un comportamiento lineal con respecto al número de comparaciones, hace $O(N^2)$ comparaciones cuando el arreglo está en orden inverso. Así que *Merge Sort* es el único de los algoritmos analizados que no tiene un comportamiento cuadrático con respecto al número de operaciones cuando los datos están en orden inverso.

Analizando ahora el tiempo de ejecución, vemos que *Quick Sort y Bubble sort* tienen tiempos de ejecución muy parecidos, mientras que *Selection Sort, e Insertion Sort* son mucho más rápidos que los dos algoritmos anteriormente mencionados. De nuevo esto parece interesante, pues aunque estos cuatro algoritmos hacen $O(N^2)$ comparaciones, tienen tiempos de ejecución diferentes.

La causa de esta disparidad entre los tiempos de ejecución se vuelve evidente cuando comparamos el tiempo de ejecución de *Bubble Sort* cuando los datos están ordenados y cuando los datos están en orden inverso. Tenemos que un mismo algoritmo, que hace el mismo número de comparaciones en ambos casos, es más lento cuando los datos están en orden inverso. Esto se debe a que cuando los datos están en orden inverso *Bubble Sort* debe además hacer $O(N^2)$ intercambios entre elementos del arreglo.

Vemos entonces que el tiempo de ejecución de un algoritmo de ordenamiento no depende exclusivamente del número de comparaciones que realiza, sino también de las manipulaciones que hace sobre el arreglo.

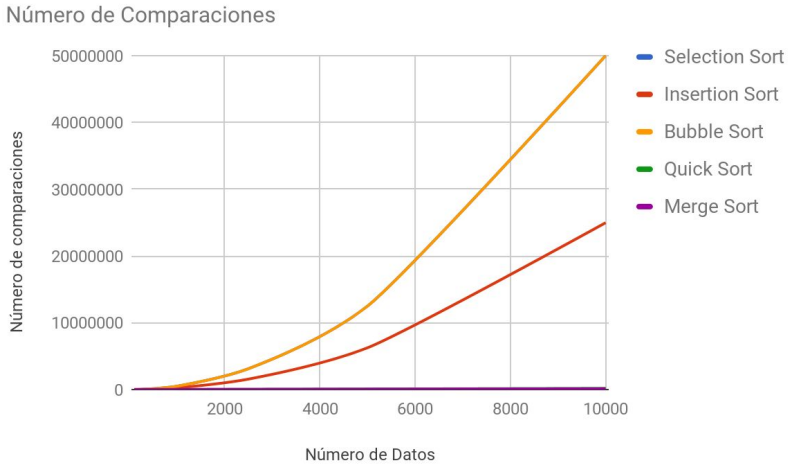
Por último, *Merge Sort* es el que tiene menor tiempo de ejecución entre los cinco algoritmos cuando los datos están en orden inverso.

Con los datos en orden aleatorio

El último análisis se realizó con los datos en orden aleatorio, para esto se probó cada algoritmo con distintos números de datos. Para cada uno de estos análisis se ejecutó el algoritmo con 30 permutaciones aleatorios del arreglo y se tomó el promedio del número de comparaciones y del tiempo de ejecución.

Número de Comparaciones	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Insertion Sort	2.577	15.739	62.729	142539	251601	1.562.220	6253120	24976200
Bubble Sort	4950	31125	124750	280875	499500	3123750	12497500	49995000
Quick Sort	780	2407	5493	8777	12301	35167	77932	167357
Merge Sort	540	1677	3854	6234	8710	25117	55226	120450

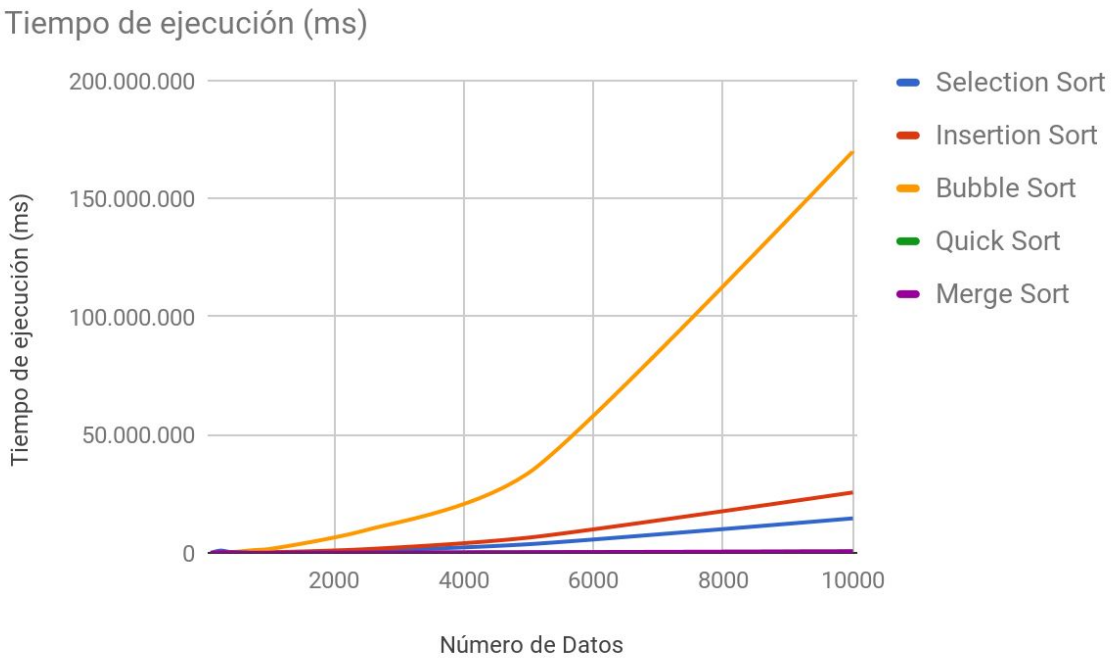
Tabla 5: Número de comparaciones por algoritmo en un arreglo con los datos en orden aleatorio.



Gráfica 5: Número de comparaciones por algoritmo en un arreglo con los datos en orden aleatorio.

Tiempo de ejecución (ms)	Número de Datos							
	100	250	500	750	1000	2500	5000	10000
Selection Sort	1.933	976.667	37.433	835.667	145.233	917.967	3.673.375	14.626.515
Insertion Sort	2.663	15.700	62.800	142.300	252.333	1.579.930	6.463.630	25.600.400
Bubble Sort	18.220	116.100	484.967	1.140.130	1.618.230	9.666.470	33.818.621	170.013.252
Quick Sort	2.866	8.315	19.466	30.067	42.600	116.033	261.533	535.715
Merge Sort	2.733	8.266	19.800	31.112	43.253	120.145	277.251	704.667

.Tabla 6: Tiempo de ejecución por algoritmo en un arreglo con los datos en orden aleatorio.

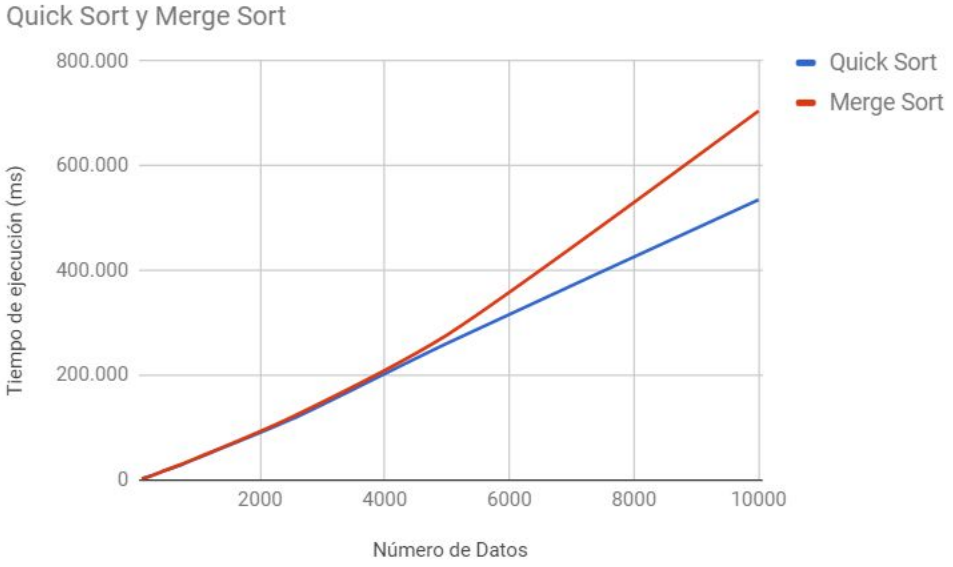


Gráfica 6: Tiempo de ejecución por algoritmo en un arreglo con los datos en orden aleatorio.

Observando el número de comparaciones realizadas por cada algoritmo, se puede ver, primero que el número de comparaciones que hacen *Selection Sort y Bubble Sort* es constante para un número de datos dato, y sin embargo el tiempo de ejecución promedio de *Bubble Sort* es mucho mayor que el del resto de los algoritmos. *Selection Sort* y *Bubble Sort* siempre hacen $O(N^2)$ comparaciones, sin importar las características del arreglo.

Por otro lado, *Insertion Sort* hace una cantidad diferente de operaciones dependiendo de las características del arreglo. Vimos que hace O(N) operaciones cuando el arreglo está ordenado, $O(N^2)$ comparaciones cuando el arreglo está en orden inverso, y por los resultados obtenidos en este último análisis, vemos que en general hace la mitad de las comparaciones que hace *Selection Sort o Bubble Sort,* sin embargo su tiempo promedio de ejecución es mayor que el de *Selection Sort.*

Queda por último analizar *Merge Sort y Quick Sort* ya que en la gráfica no se aprecia bien la diferencia entre ambos, vale la pena agregar una gráfica comparativa entre estos dos algoritmos.



Gráfica 7: Comparación entre Quick Sort y Merge Sort

Si vemos además la tabla de número de comparaciones, notaremos que *Merge Sort* hace en general menos comparaciones que *Quick Sort,* sin embargo el tiempo de ejecución de *Quick Sort* se vuelve menor que el de *Merge Sort* después de cierto punto. *Merge Sort* es quizá el algoritmo más constante en cuanto a número de comparaciones y tiempo de ejecución se refiere, pues por la forma en que está diseñado el algoritmo, el comportamiento del mismo depende muy poco de las características del arreglo.

Por otro lado, *Quick Sort* es un caso muy especial. Tiene un comportamiento cuadrático en los dos primeros ejemplos analizados, pero en el caso más general en el que los datos están en un orden aleatorio, tiene un comportamiento muy parecido al de *Merge Sort,* superándolo incluso en tiempo de ejecución. *Quick Sort,* es de hecho el algoritmo utilizado en la librería estándar de C++ en su método *sort,* ya que, como vimos en este experimento, puede llegar a ser muy rápido, en especial si se toman ciertas precauciones para elegir el pivote, y evitar los casos en que tiene un comportamiento cuadrático.

Notas:
Se puede consultar la implementación del proyecto aquí:
https://github.com/CharlyGaleana/Estructuras-de-Datos/tree/master/ComparingSortAlgorithms