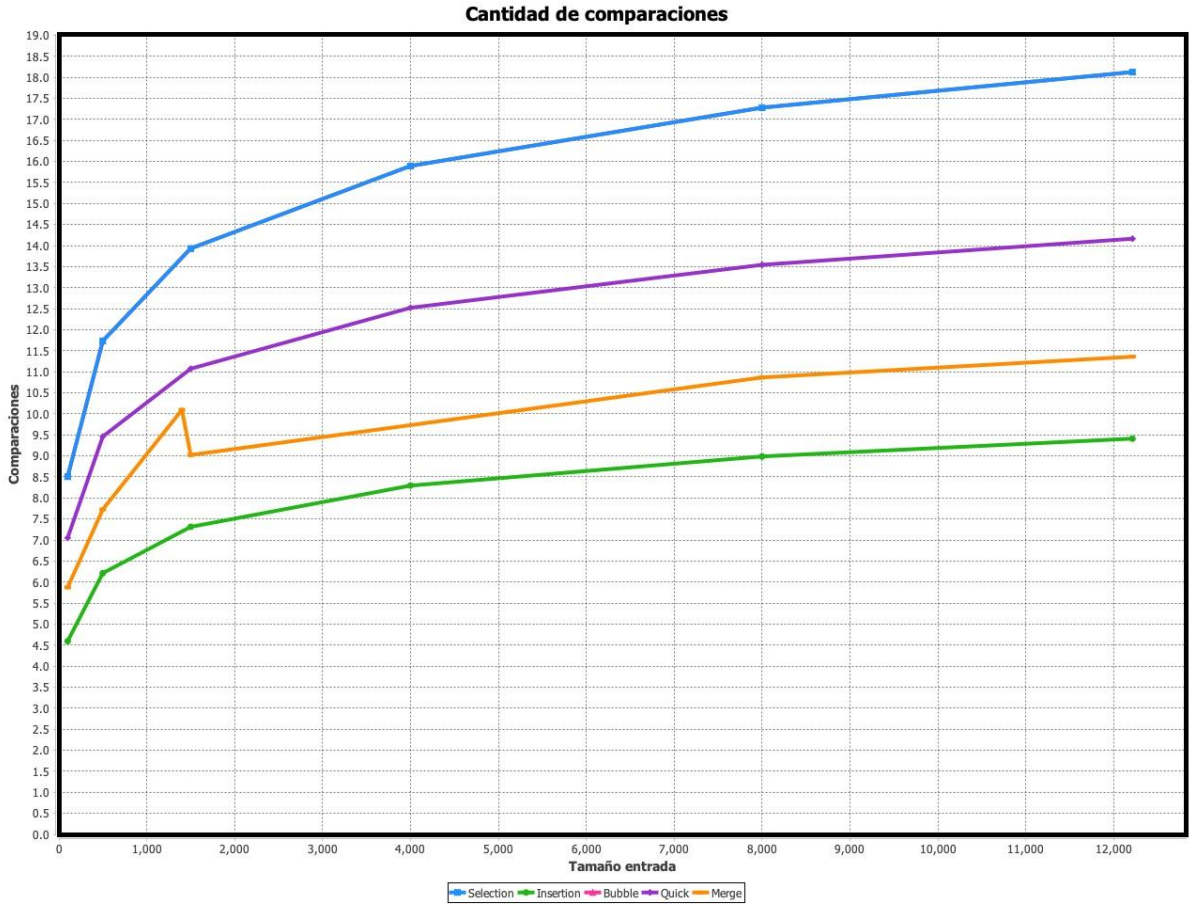
Murillo García Sebastián 191838

Tarea algoritmos de ordenamiento.

Para fines de este documento se implementan los algoritmos de ordenamiento: Selection Sort, Insertion Sort, Bubble Sort, Quick Sort y Merge Sort, para poder comparar su desempeño en tiempo de ejecución y en cantidad de comparaciones. Para ello, los algoritmos se probaron para tres variantes de datos: ordenados [el *mejor de los casos*], en orden inverso [el *peor de los casos*] y en un orden aleatorio [el *caso promedio*]. Además, se corrieron los algoritmos con 6 diferentes tamaños de entrada, a saber: 100, 500, 1500, 4000, 8000 y 1221. Aquí se muestran los resultados. Se debe mencionar que en las gráficas, Merge Sort tiene un pico extraño.

Datos completamente ordenados. [El mejor de los casos]

I. *Figura A*: Cantidad de comparaciones para datos completamente ordenados.



Interpretación.

A los valores de los pasos se les sacó el logaritmo para que se pudieran apreciar mejor las diferencias entre los métodos en la gráfica.

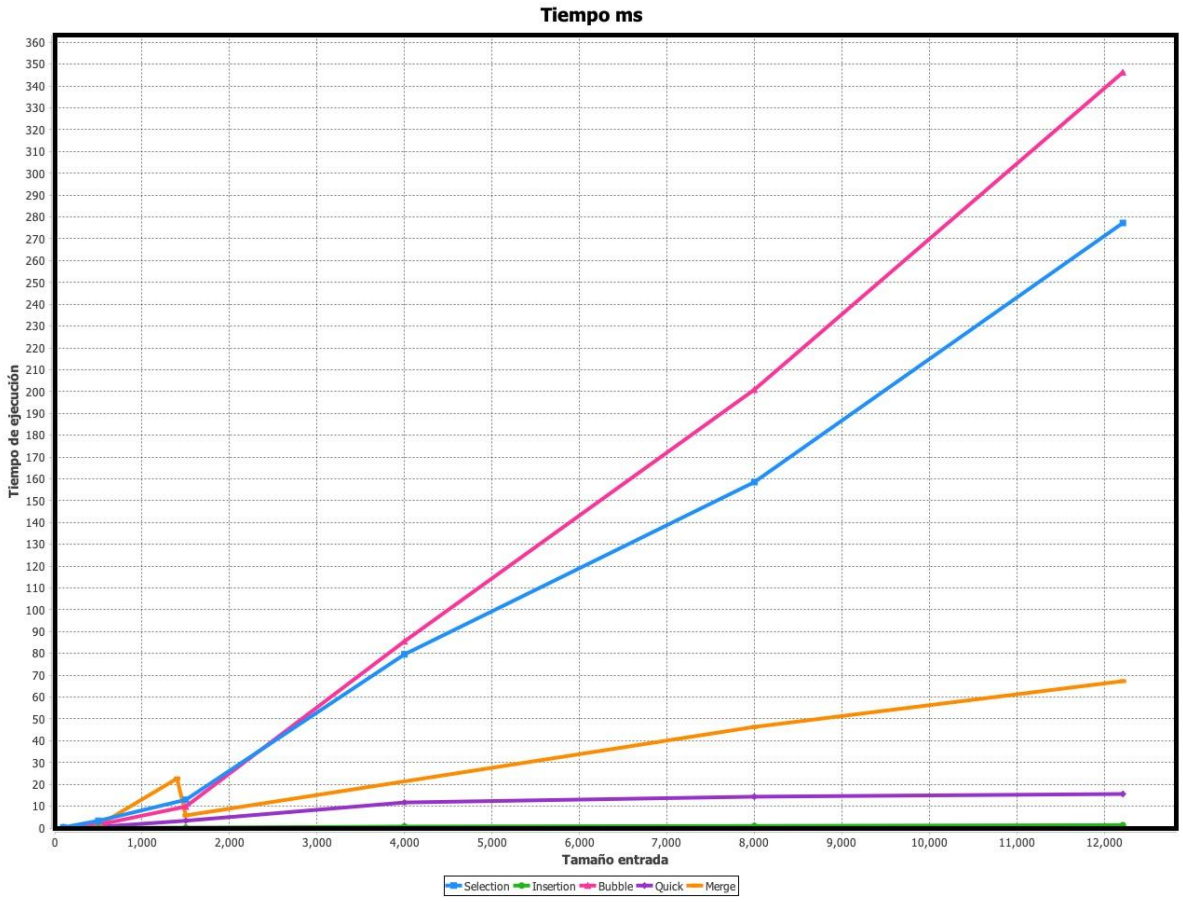
Para empezar, se debe aclarar que tanto Selection Sort como Bubble Sort realizan el mismo número de comparaciones para cualquier tamaño de entrada de datos. Esto se debe a que tanto en el *peor caso* como en el *mejor caso* o inclusive en el *caso promedio*, estos algoritmos de ordenamiento tienen complejidad de $O(n^2)$. De esta manera se tiene como resultado que la cantidad de comparaciones para ambos algoritmos es de $(n^2+n)/2$; es decir, el número de comparaciones no depende del ordenamiento de los datos, sino de la cantidad de los mismos. Es por ello que la curva de Bubble Sort no está visible, porque la curva de Selection Sort la cubre en su totalidad. Además, por esta misma razón, estos son los dos algoritmos que realizan el mayor número de comparaciones por cada tamaño de entrada.

Por otro lado, el caso del algoritmo de ordenación Insertion Sort es un caso particular porque en el *mejor de los casos* (que es el que consideramos ahorita), este algoritmo tiene complejidad de O(n). Esto lo podemos ver claramente en la gráfica debido a que este algoritmo hace n-1 comparaciones por cada tamaño de entrada. Como ejemplo podemos decir que para la entrada de tamaño 4000 hace 3999 comparaciones, para el caso de tamaño 12214 hace 12213 comparaciones, etcétera. Debido a ello, este algoritmo resulta ser el que menor número de comparaciones hace en el *mejor de los casos*.

Para los casos particulares del Quick Sort y Merge Sort tenemos algo interesante. En el caso que estamos considerando, el primer algoritmo antes mencionado es de complejidad $O(n \log n)$ y el segundo de $\Omega(n \log n)$. ¿En qué radica la diferencia? Como veremos en las tres gráficas de cantidad de comparaciones, Merge Sort tiene una menor cantidad de resultado por tamaño de entrada que Quick Sort.

Por lo tanto, el orden de menor a mayor en cuanto a número de comparaciones en *el mejor de los casos* queda como el siguiente: Insertion, Merge, Quick, Bubble y Selection.

II. *Figura B*: Tiempo de ejecución para datos completamente ordenados.



Interpretación.

En cuanto al tiempo de ejecución, podemos ver algo curioso: entre mayor sea la entrada, más distantes se vuelven los valores de tiempos de ejecución de los algoritmos. Por un lado, podemos observar que el algoritmo que menos tiempo se tarda en "ordenar" datos que ya están ordenados es Insertion Sort. Tal vez esto tiene correspondencia con el hecho de que *en el mejor de los casos* este algoritmo es de complejidad O(n). Además, su estructura misma tiene un **while**, esto es de vital importancia debido a que por el hecho de contar con datos ordenados, el método no entra en esa estructura algorítmica. Esto influye en su tiempo de ejecución.

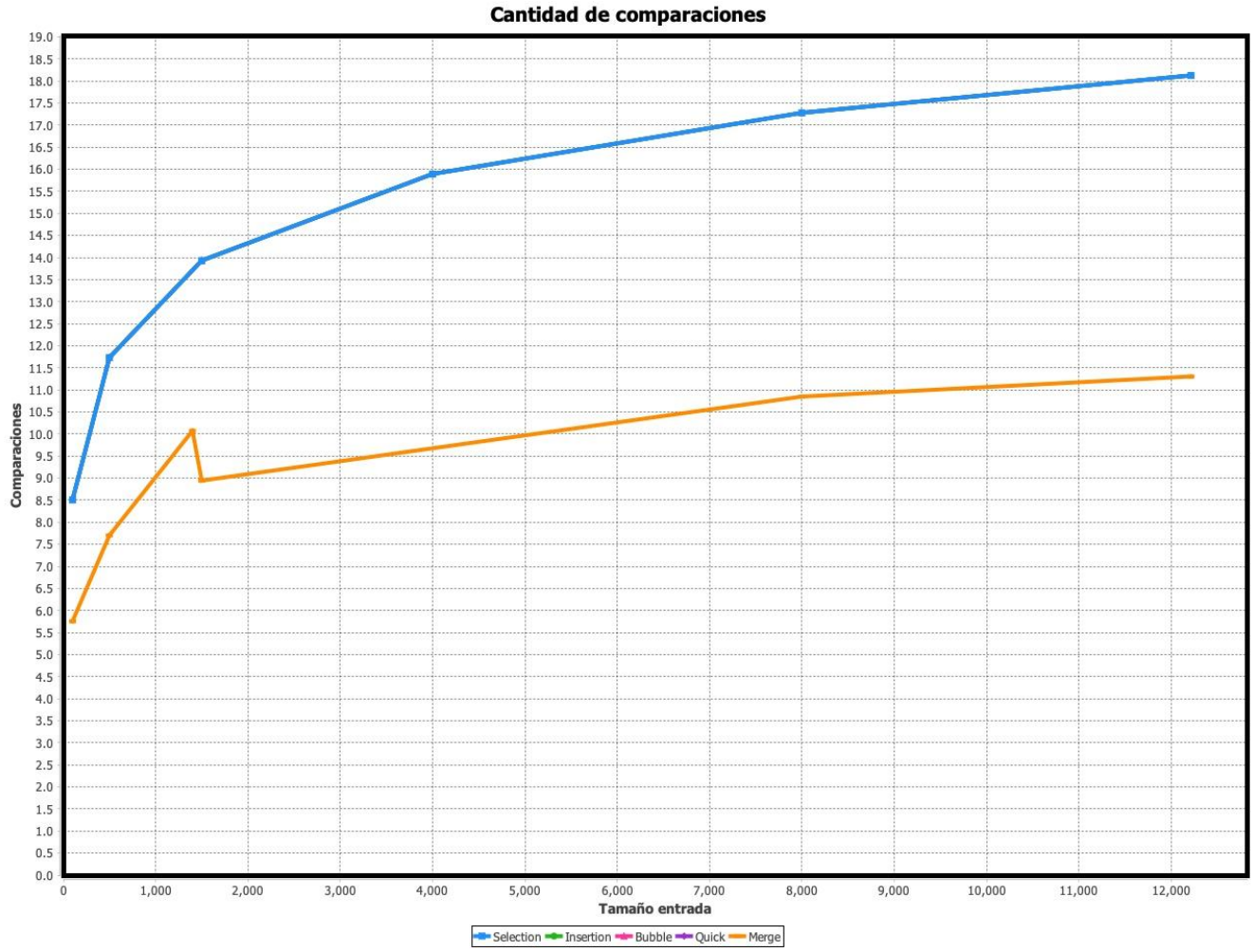
Por otro lado, podemos observar que Bubble Sort es el algoritmo que más tiempo se va a tardar en ejecutar el ordenamiento, y que el algoritmo de Selection Sort le sigue. Esto puede deberse al hecho de que ambos son de complejidad $O(n^2)$, aunque también puede deberse a una cuestión misma del funcionamiento de ambos algoritmos, pues podemos observar que a pesar de que tienen la misma complejidad en este caso, ambos tienen tiempos distintos de ejecución. El hecho de que sean los más

tardados en este caso puede deberse a que a pesar de que los datos están ordenados, ambos algoritmos tienen que pasar por dos **for**'s para completar su funcionamiento.

La diferencia de tiempos entre Quick Sort y Merge Sort puede deberse a la estructura algorítmica con la que funciona cada uno de los métodos. Particularmente diría que Merge tarda más en ejecutarse por el hecho de que separa en mitades al arreglo, mientras que Quick ocupa como pivote (y el más funcional) al primer elemento para hacer los intercambios necesarios. Es decir, diría que Merge tarda más debido a su funcionamiento en el cual parte en mitades y compara los elementos uno por uno de ambas mitades. Por lo tanto, el orden de menor a mayor en cuanto a tiempo de ejecución en *el mejor de los casos* queda como el siguiente: Insertion, Quick, Merge, Selection y Bubble.

Datos en un orden inverso. [El peor de los casos]

III. *Figura A*: Cantidad de comparaciones para datos en un orden inverso.



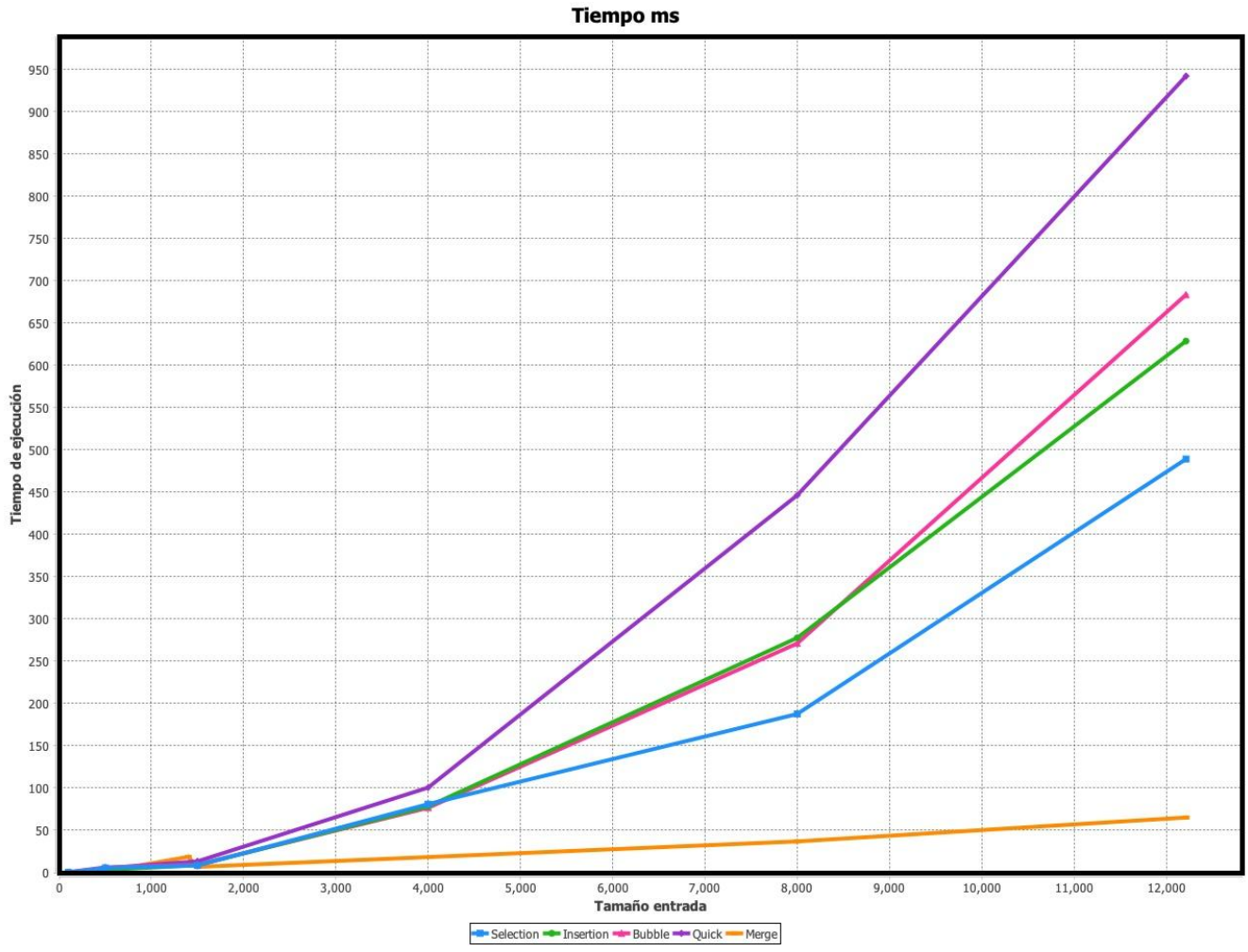
Interpretación.

En esta gráfica tenemos algo interesante: la curva que corresponde tanto a Quick, como a Bubble y como a Insertion es la misma curva que la de Selection. ¿A qué se debe esto? Como estamos considerando el *peor de los casos*, debe decirse que estos cuatro algoritmos tienen complejidad de $O(n^2)$. ¿Qué tiene de implicaciones? Como dijimos anteriormente, el resultado de tener una complejidad de este tipo en el conteo de comparaciones es que el resultado siempre será de $(n^2+n)/2$.

Esto nos indica una y solo una cosa segura: para el *peor de los casos*, Merge Sort es el algoritmo que realiza menor cantidad de comparaciones. Esto se debe al hecho de que, en este caso, el algoritmo es de complejidad $O(n \log n)$. Aquí tenemos algo interesante que ver: el *mejor caso* del Quick Sort tiene tiene la misma complejidad del *peor de los casos* del Merge.

Además, es interesante ponerse a pensar que en este caso, 4 de 5 algoritmos resultan ser los de mayor número de comparaciones, llegando siempre a su tope máximo: $(n^2+n)/2$.

IV. *Figura B*: Tiempo de ejecución para datos en un orden inverso.



Interpretación.

Por otro lado, podemos observar cosas interesantes en esta gráfica. Para empezar, Quick Sort resulta ser el algoritmo que más se tarda en ejecutarse, ¿a qué se puede deber esto? Puede deberse a dos factores: a su complejidad [$O(n^2)$] y a su estructura algorítmica. Esto último lo digo porque para este algoritmo el todos los elementos de la partición siempre serán mayores que el último elemento.

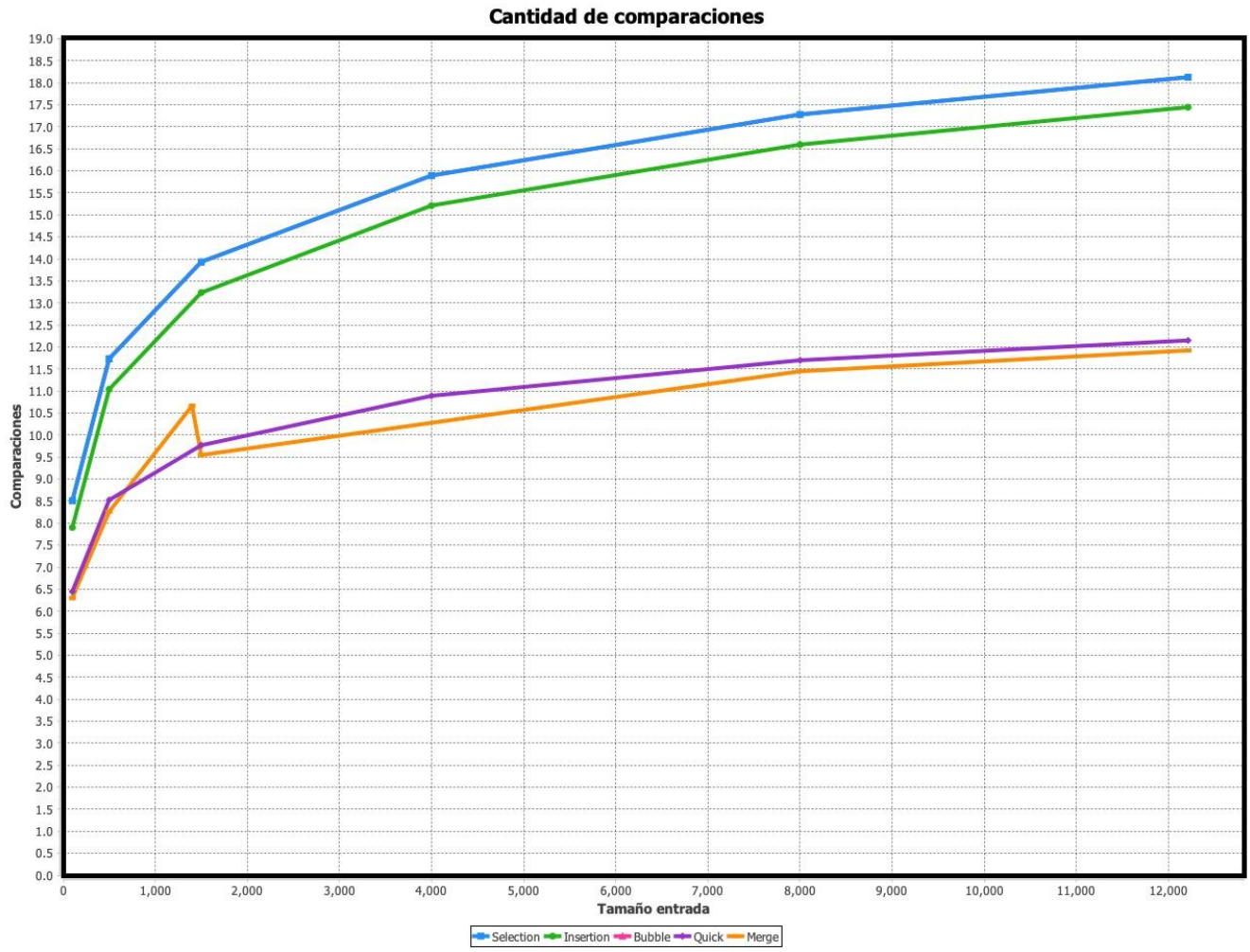
Resulta ser que Merge Sort es el algoritmo que menos se tarda en efectuar el ordenamiento y nuevamente esto puede ser por su complejidad en el peor de los casos. Es aquí donde entra la importancia de que este algoritmo divide en mitades para poder ordenar, pues como se dice en clase: "¿cuánto se va a tardar?" $n \log n$.

En cuanto a los otros algoritmos solo podemos mencionar el hecho de que parecen cruzarse las curvas de Bubble e Insertion, pero esto puede deberse a una cuestión de margen de error, pues parece ser que al final Bubble suele ser más tardado que Selection e Insertion. Una cosa que puede influir en la diferencia de tiempos entre Selection y Bubble es que mientras que el primero guarda la posición del menor elemento para luego hacer el **swap** correspondiente al posicionamiento de este menor elemento, en el segundo algoritmo se van haciendo los intercambios en la medida en la que avanza por el arreglo. También esto puede ser motivo por el cual Bubble suele ser el más tardado [exceptuando este caso, en el cual Quick resulta ser el menos eficiente].

Por lo tanto, el orden de menor a mayor en cuanto a tiempo de ejecución en *el peor de los casos* queda como el siguiente: Merge, Selection, Insertion, Bubble y Quick.

Datos ordenados aleatoriamente. [El caso promedio]

V. *Figura A*: Cantidad de comparaciones para datos ordenados aleatoriamente.



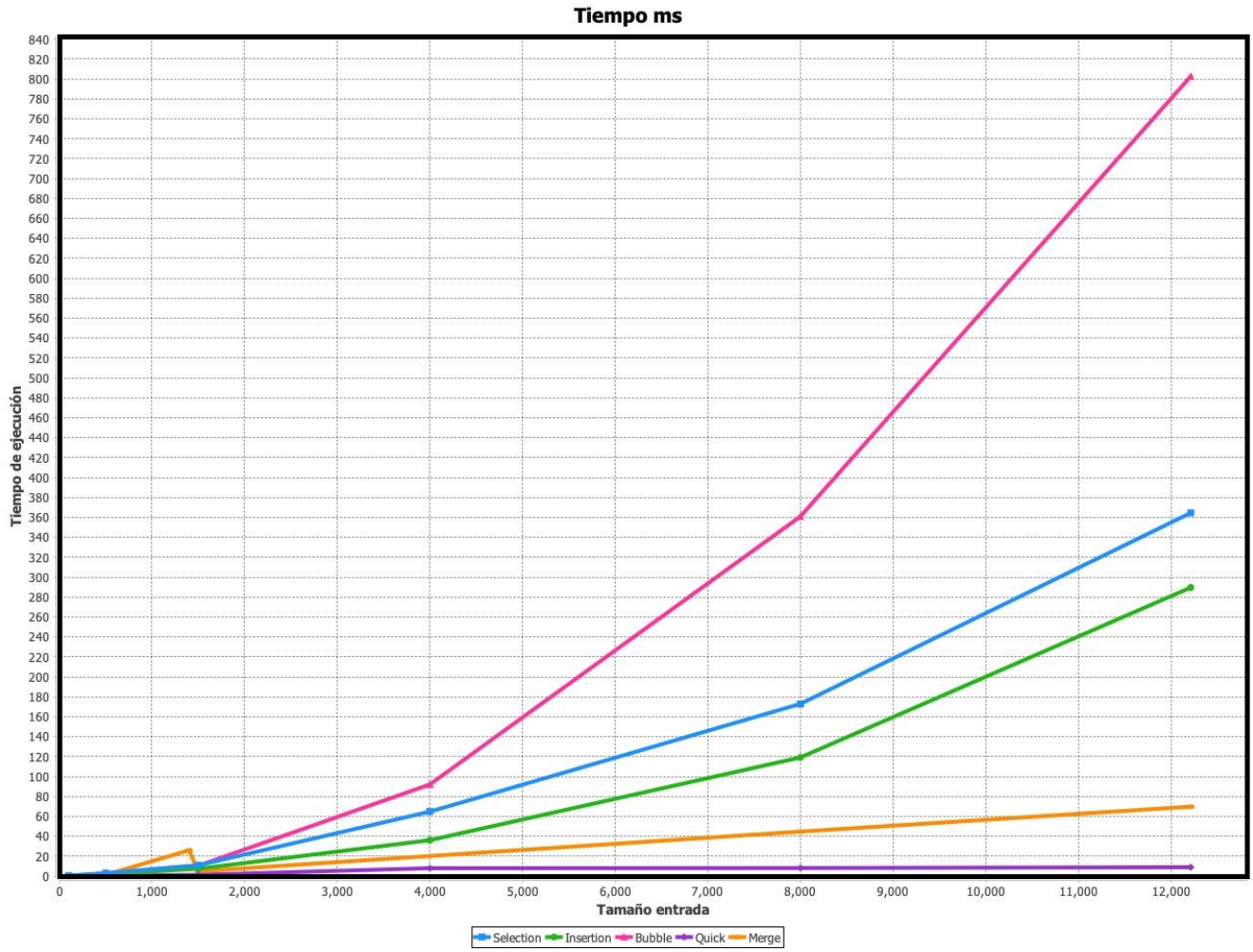
Interpretación.

Por último, hay que volver a recordar una cosa importante: tanto en el *peor caso* como en el *mejor caso* o en el *caso promedio*, Selection y Bubble tienen complejidad de $O(n^2)$, es por ello que son los que más comparaciones hacen [recordemos: $(n^2+n)/2$]. También podemos ver y confirmar algo curioso que desde un principio mencionamos: Merge Sort tiene una menor cantidad de resultado por tamaño de entrada que Quick Sort y esto puede radicar en la complejidad de sus *casos promedios* [ambos de O(n log n)]. A pesar de tener la misma complejidad, la misma estructura algorítmica puede generar esa diferencia que parece ser mínima en comparaciones.

Ahora bien, la diferencia entre la cantidad de comparaciones de Bubble y Selection con Insertion puede deberse a que este último algoritmo tiene un **while** en su interior el cual puede o no entrar y ejecutarse, mientras que en los otros dos entran en un **for** obligatorio.

Por lo tanto, el orden de menor a mayor en cuanto a número de comparaciones en *el caso promedio* queda como el siguiente: Merge, Quick, Insertion, Bubble y Selection.

VI. *Figura B*: Tiempo de ejecución para datos ordenados aleatoriamente.



Interpretación.

Por último podemos ver en su clara "aleatoriedad" de datos el tiempo de ejecución de cada uno de los algoritmos. Bubble sort resulta ser el más tardado, y esto podría decir que es por su estructura en la que con cada vuelta tiene hacer los **swap** dentro de una estructura algorítmica repetitiva interior. Aquí podemos volver a recordar que Selection, en su contraparte no hace los **swap** necesarios dentro de la estructura repetitiva interior, sino que lo realiza en la exterior y esto claramente influye en el tiempo. Por otro lado, podemos recalcar la importancia de que Insertion ocupe de un **while** para hacer las

comparaciones, porque en el momento en que se encuentre con una serie de datos ordenados, no tendrá que seguir continuando con su ciclo repetitivo. Pero, ¿entonces por qué quick Sort y merge Sort tienen los mejores tiempos de ejecución? Siempre hay que tener en mente la complejidad de los algoritmos para sus diferentes casos, ya que estos nos dan cierto "rango" en el cual estos trabajan. A pesar de que son métodos recursivos (y esto podría pensarse que genera un mayor tiempo), estos dos trabajan con partes del arreglo original de tal manera que sea lo más eficiente su manejo para el ordenamiento. La diferencia entre ambos puede radicar, como ya se ha dicho antes, en cómo es que ambos tratan con las particiones, con las mitades y cómo es que ordenan los elementos dentro de ellas.

Por lo tanto, el orden de menor a mayor en cuanto a tiempo de ejecución en *el caso promedio* queda como el siguiente: Quick, Merge, Insertion, Selection y Bubble.

Conclusión.

Es interesante conocer los tiempos de ejecución y la cantidad de comparaciones que estos algoritmos de ordenamiento realizan debido a que nos dan una idea de cuál es el más eficiente para cierta cantidad de datos y cierto arreglo de los mismos. No podemos concluir con que uno sea el más eficiente, pues depende tanto del ordenamiento de los datos como del tamaño de entrada. Sobre esto último, es importante hacer notar que entre más chica sea la entrada, parece ser que las diferencias entre tiempo son casi nulas. Esto nos da un aspecto importante a tomar sobre cuál algoritmo de ordenamiento utilizar cuando se trata de grandes volúmenes de datos.