Instituto Tecnológico Autónomo de México

Estructuras de Datos Avanzadas

Análisis de algoritmos de ordenamiento

José Luis Gutiérrez Espinosa 179888

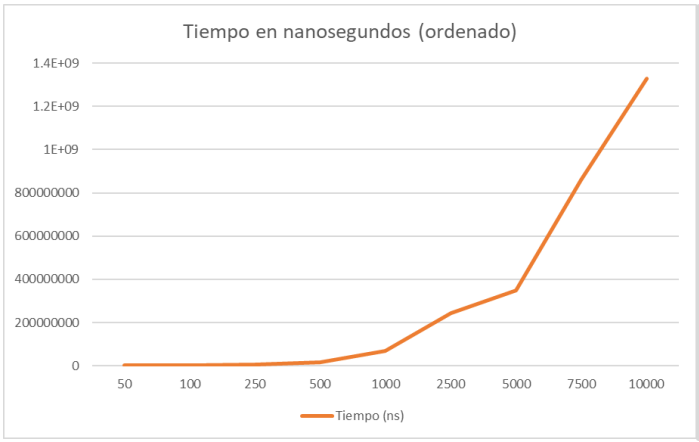
Fecha de entrega: 18-Septiembre-2019

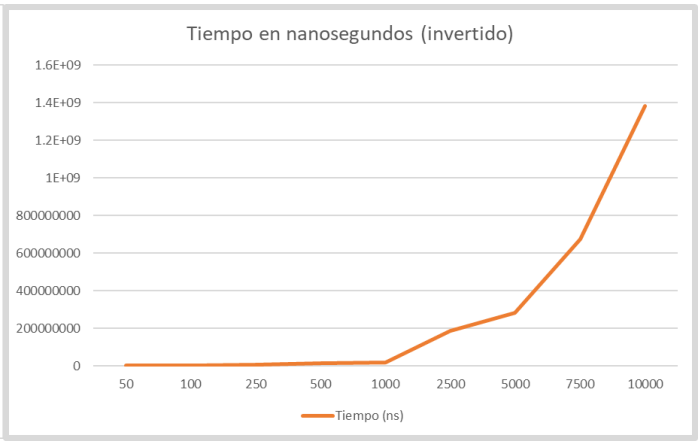
En el trabajo presente se mostrarán los resultados de probar los distintos algoritmos de ordenamiento vistos en clase (selection sort, insertion sort, bubble sort, quick sort y merge sort). Para todos los algoritmos se realizaron pruebas con 50, 100, 250, 500, 1000, 2500, 5000, 7500 y 10000 datos para comparar el desempeño de cada uno según el tamaño de la entrada a ordenar.

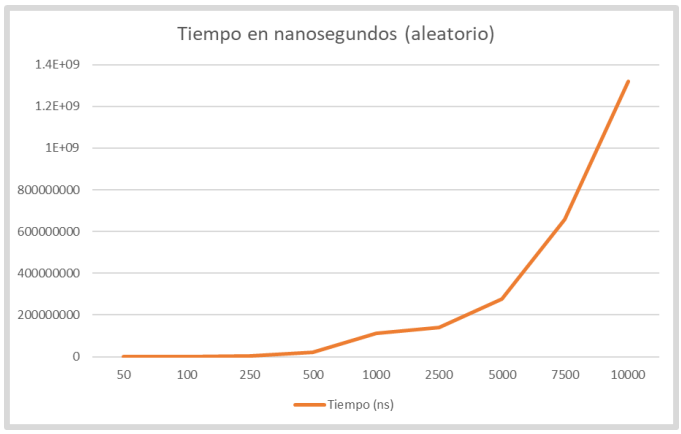
Resultados:

a) Tiempos de ejecución

- **Selection Sort**



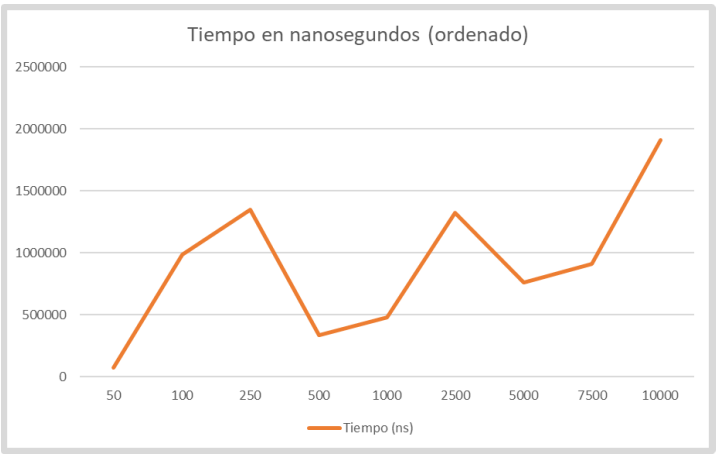


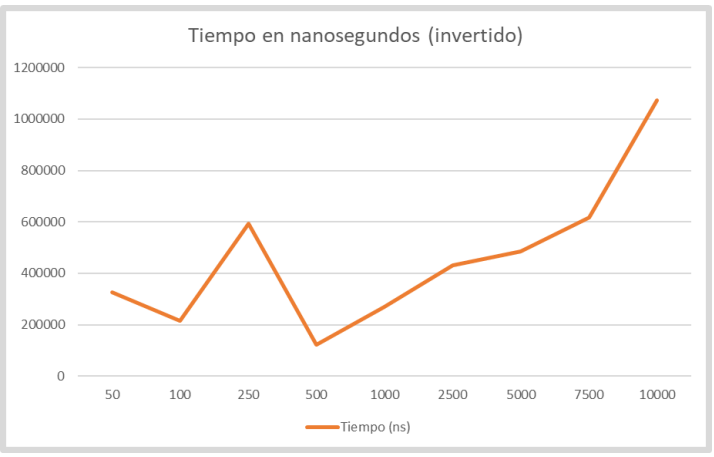


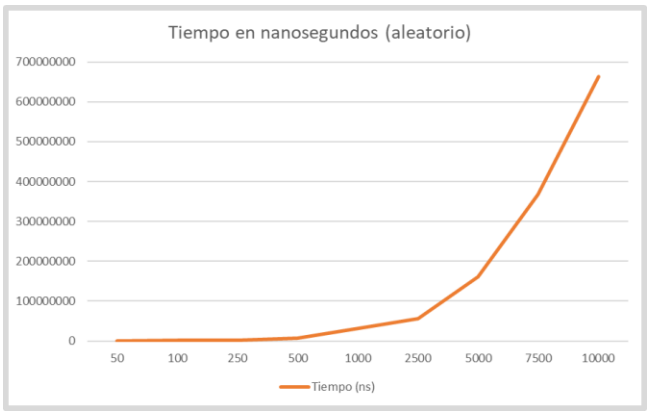
En el tiempo de ejecución para selection sort podemos observar que todas las gráficas son parecidas, y que crecen con una tendencia prácticamente exponencial, siendo el caso con el arreglo invertido el más evidente, ya que se aproxima mucho a la gráfica $y = 603264e^{0.8598x}$.

Este crecimiento tan acelerado nos muestra que es un algoritmo muy demandante para el CPU, por lo que no es una buena opción para ordenar arreglos de gran tamaño.

- **Insertion Sort**

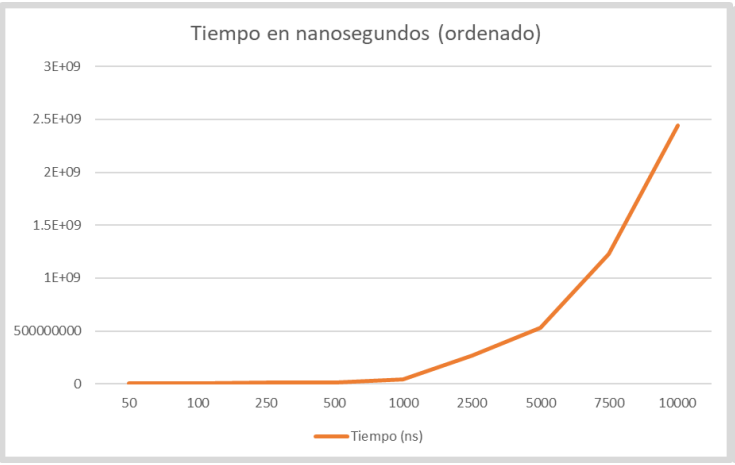


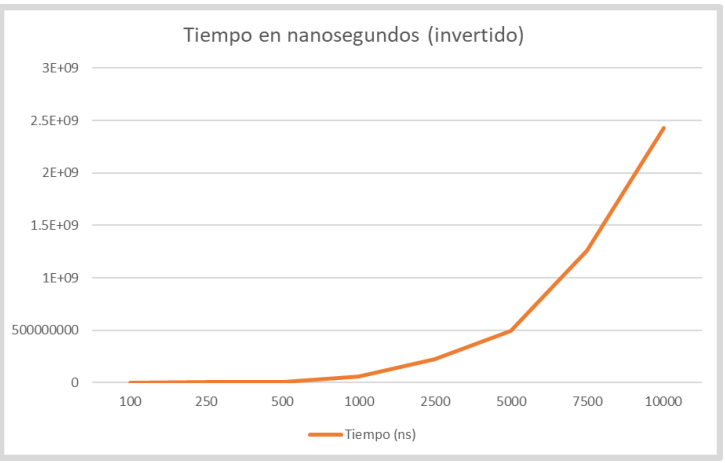


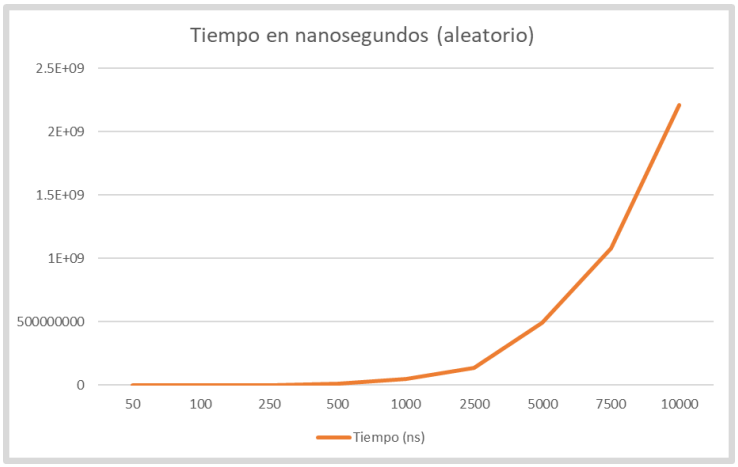


Las gráficas obtenidas para el tiempo de ejecución del algoritmo insertion sort son muy interesantes, ya que las gráficas para el arreglo ordenado y el arreglo invertido no tienen una tendencia clara, además del hecho que tienden a crecer a medida que el tamaño de la entrada también crece, sin embargo, con el arreglo aleatorio podemos observar un comportamiento más estable. Incluso, es posible tener una aproximación confiable con la gráfica $y = 4E+06x^3 - 3E+07x^2 + 1E+08x - 7E+07$, lo que nos muestra que, al parecer, tiene un crecimiento de x^3. Este nivel de demanda al CPU, aunque no es tan alto como el que muestra el algoritmo de selection sort, sigue siendo alto.

- **Bubble Sort**

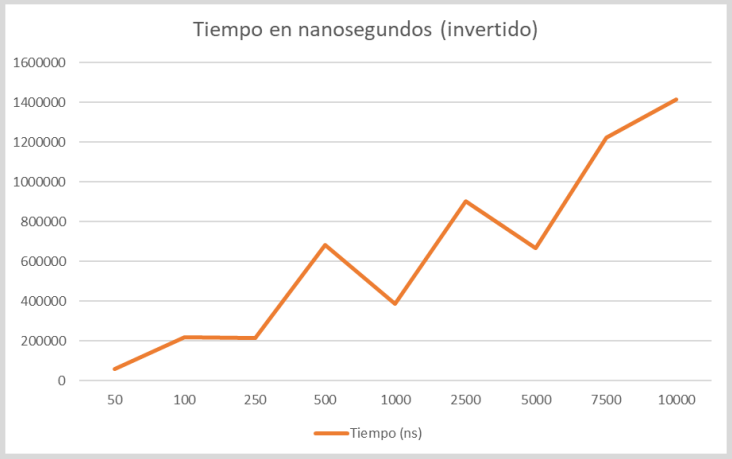






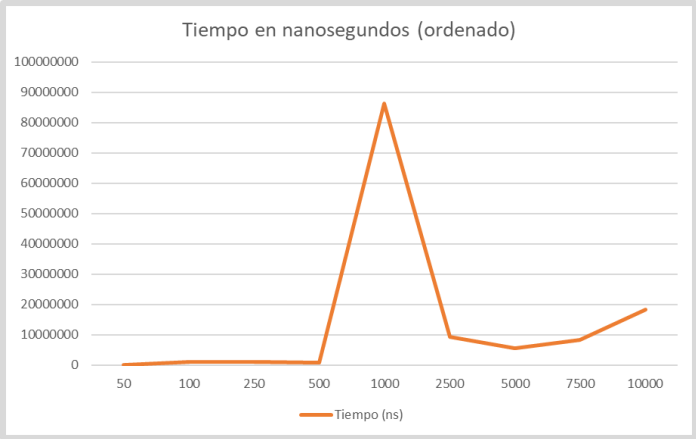
Para el algoritmo bubble sort podemos observar un comportamiento parecido en los tres casos, mostrando que es un algoritmo estable. Sin embargo, los valores son tan elevados para entradas de tamaños superiores que la demanda que tiene al CPU es demasiado elevada como para considerarlo viable.

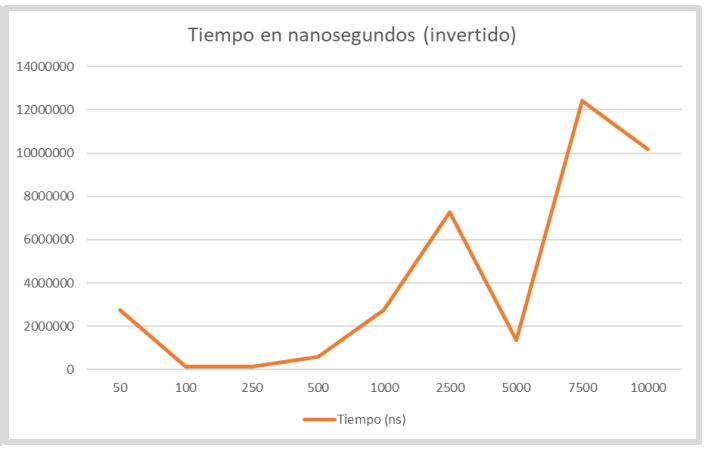
- **Quick Sort**



El algoritmo quick sort no muestra un comportamiento estable, ya que las gráficas para los distintos casos difieren mucho entre ellas. Además, se pueden observar algunos picos en el tiempo de ejecución con el arreglo invertido y en orden. Es posible que esto se deba a la "mala suerte", ya que, por la naturaleza aleatoria del algoritmo, es posible que haya tocado el peor escenario posible para esa prueba en específico. No obstante, en las pruebas con el arreglo ordenado aleatoriamente, la gráfica muestra un comportamiento regular que puede ser aproximado con la gráfica $y = 8E+06x^3 - 7E+07x^2 + 2E+08x - 2E+08$, lo que nos muestra un crecimiento muy rápido y muy parecido al que resultó de l algoritmo insertion sort. Por lo mismo, puede ser viable entre los algoritmos analizados hasta ahora.

- **Merge Sort**



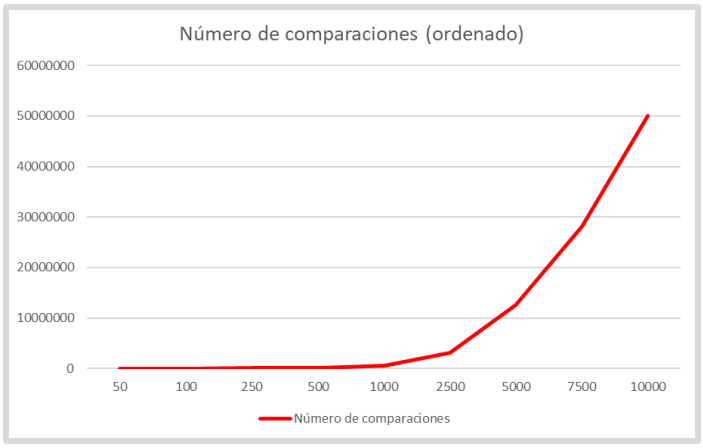


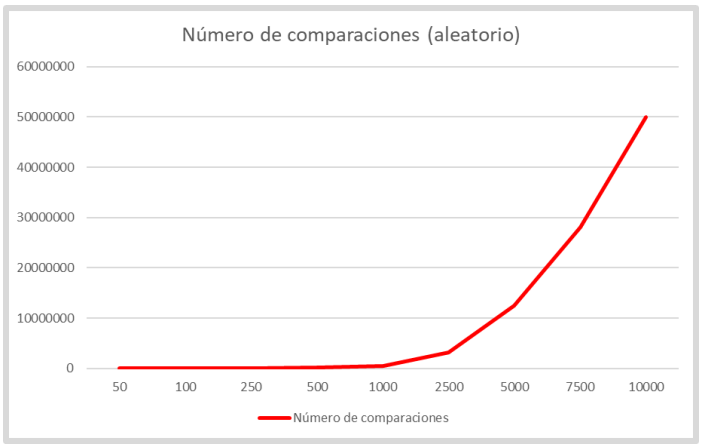
Para el algoritmo merge sort no podemos observar un comportamiento similar en ninguna de las gráficas, sin embargo, el crecimiento que presenta a medida que aumenta la entrada de datos a ordenar es el menor de todos los algoritmos analizados. Además, el tiempo de ejecución y por ende la demanda al CPU observados en este algoritmo es el menor de todos los analizados.

Basándonos en los tiempos de ejecución de los algoritmos analizados, podemos concluir que la mejor opción para que el CPU no tenga una alta demanda es el algoritmo “Merge Sort”.

b) Número de comparaciones realizadas

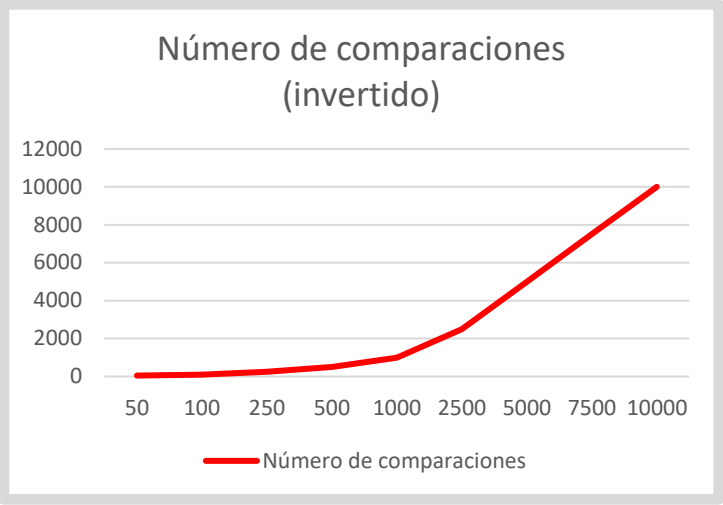
- **Selection Sort**





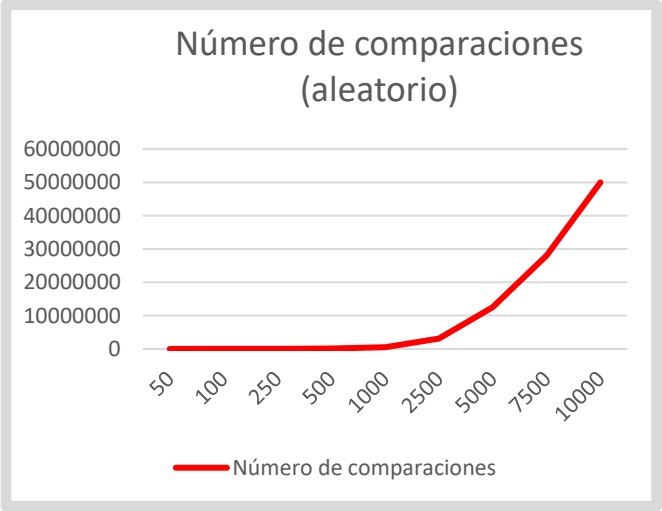
Podemos observar que el algoritmo selction sort hace el mismo número de comparaciones sin importar el orden del arreglo.

- **Insertion Sort**



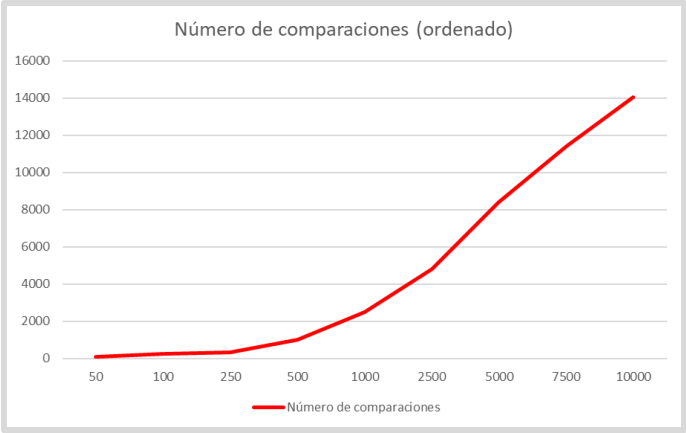
Podemos ver que insertion sort tiene el mismo número de comparaciones cuando el arreglo está ordenado que cuando está invertido, sin embargo, cuando éste se encuentra en orden aleatorio, el número de comparaciones realizadas crece a cantidades mucho más elevadas, por lo que puede que no sea la mejor opción.

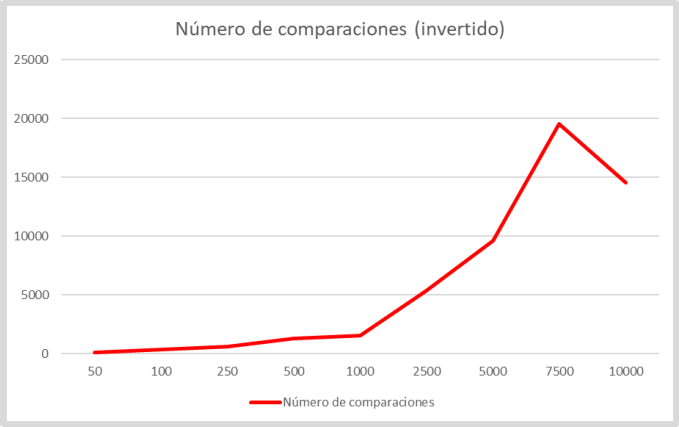
- **Bubble Sort**



Para el algoritmo bubble sort podemos ver que hace aproximadamente las mismas comparaciones en los tres casos, por lo que podemos ver que la eficiencia no depende del orden de los datos, sin embargo, la cantidad de comparaciones es más del doble que el peor caso que tiene el selection sort, por lo que bubble sort no parece ser una buena opción considerando los algoritmos analizados.

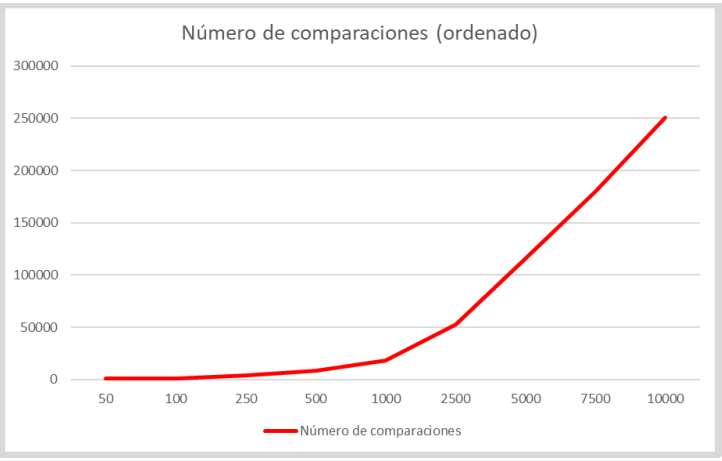
- **Quick Sort**

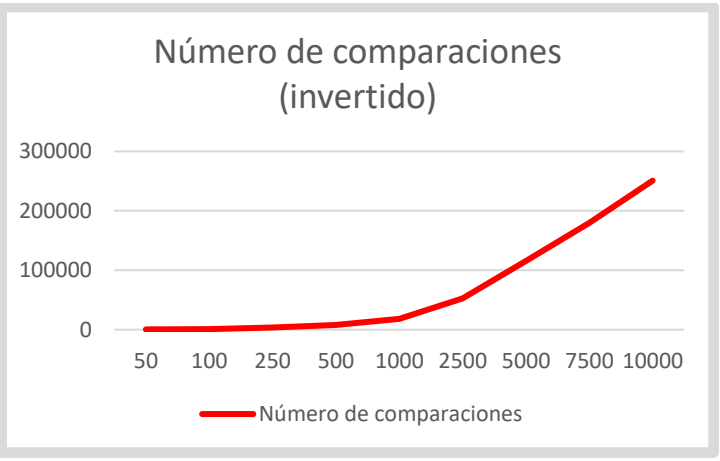


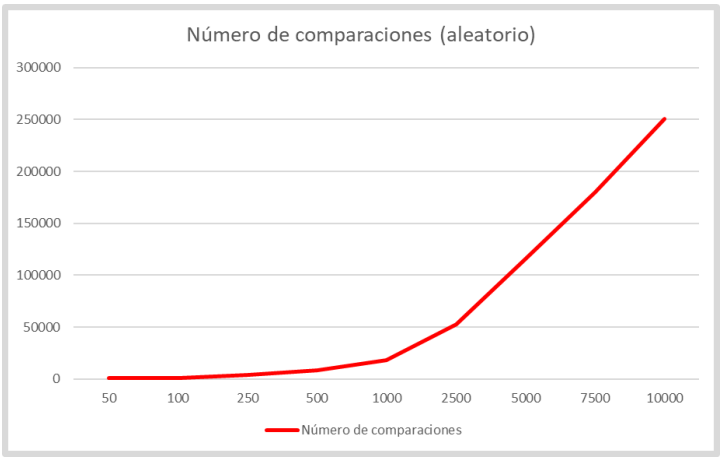


El algoritmo quick sort nos muestra un comportamiento parecido en los tres casos, sin embargo, al estar los datos en orden aleatorio, se puede observar que el número de comparaciones realizadas llega a ser más de 1000 veces mayor que en el caso del arreglo invertido, por lo que la eficiencia del algoritmo no es igual para todos los casos posibles. Además, debido al factor aleatorio del algoritmo, se puede observar en el caso del arreglo invertido que hay un pico hacia abajo en 10000 datos, lo que nos da otro argumento para decir que este algoritmo, aunque puede ser muy bueno en el mejor de los casos, depende de la suerte, por lo que puede que no sea la mejor opción.

- **Merge Sort**







El algoritmo merge sort muestra el mismo número de comparaciones para los tres casos evaluados, esto nos muestra que es un algoritmo cuya eficiencia no varía de caso en caso. Además, el número de comparaciones totales que hace el método no es tan alto como el que tienen los demás.

Analizando el número de comparaciones que hace cada algoritmo, podemos observar que claramente, el algoritmo más confiable y al mismo tiempo, el que hace menos comparaciones en general es el algoritmo “Merge Sort”.

Finalmente, tomando en cuenta que merge sort fue declarada la mejor opción tanto en el rubro de número de comparaciones como en el rubro de tiempo de ejecución, podemos concluir que el algoritmo que resulta ser la mejor opción para ordenar arreglos es el “Merge Sort”.