Análisis de Algoritmos 2018/2019

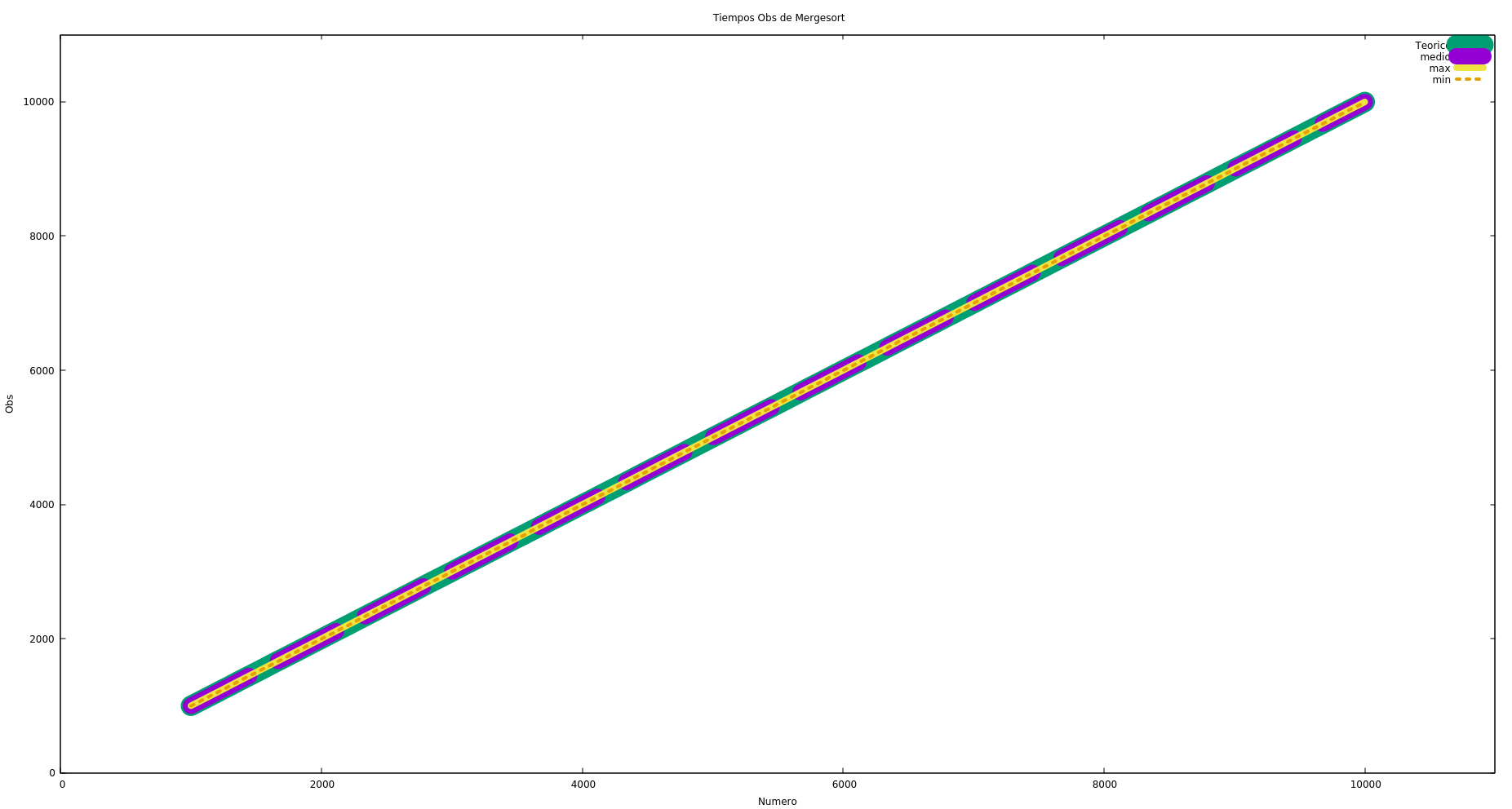
Práctica 2

Alba Ramos, Javier Lozano, 1212.

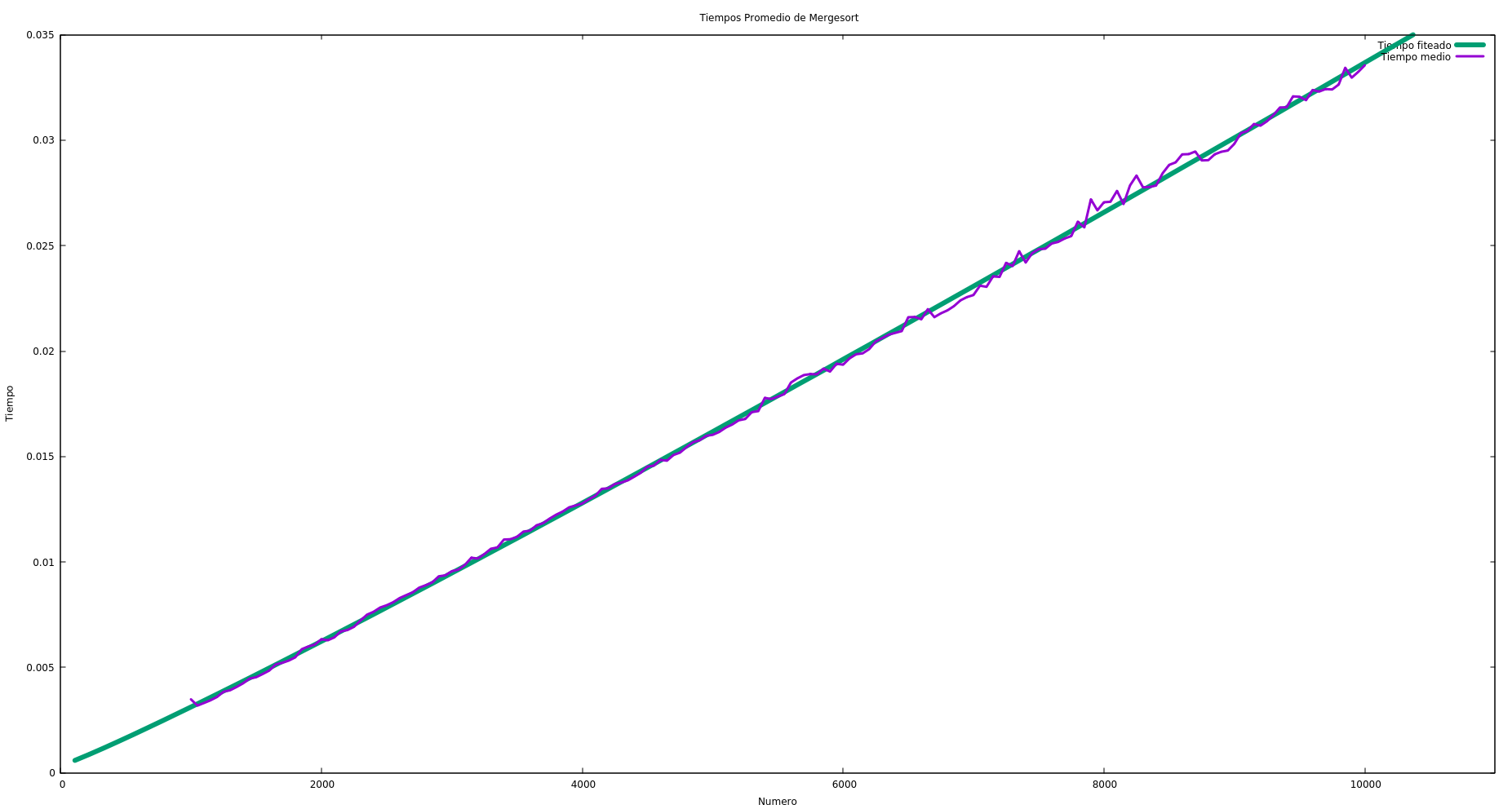


**5. Resultados, Gráficas**

5.2 Apartado 2

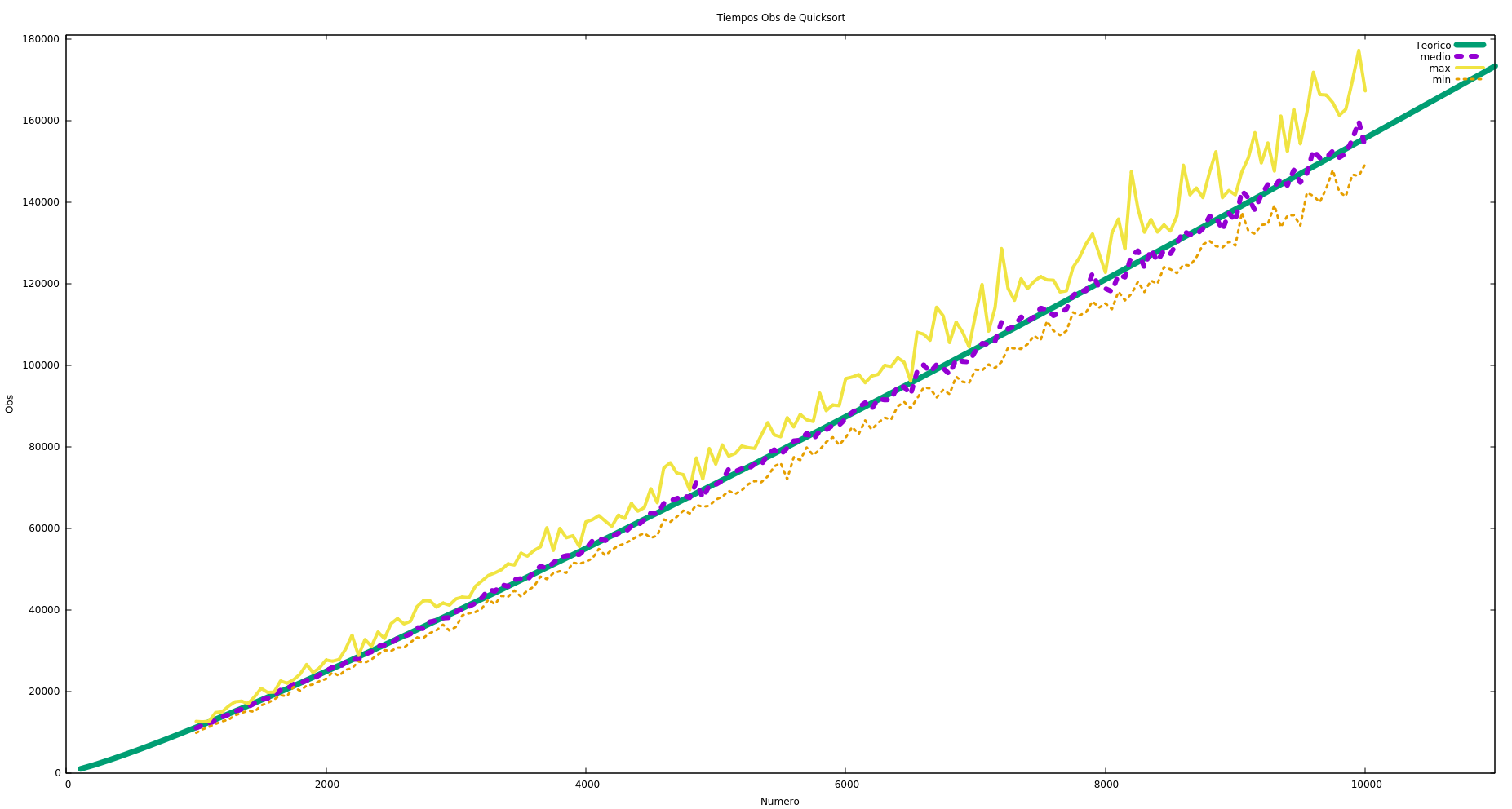


Podemos observar que el caso medio (morado), el caso peor (amarillo) y el caso mejor (naranja) se ajustan correctamente al valor teórico esperado de MergeSort: O(N\*log(N)), que aparece en verde.

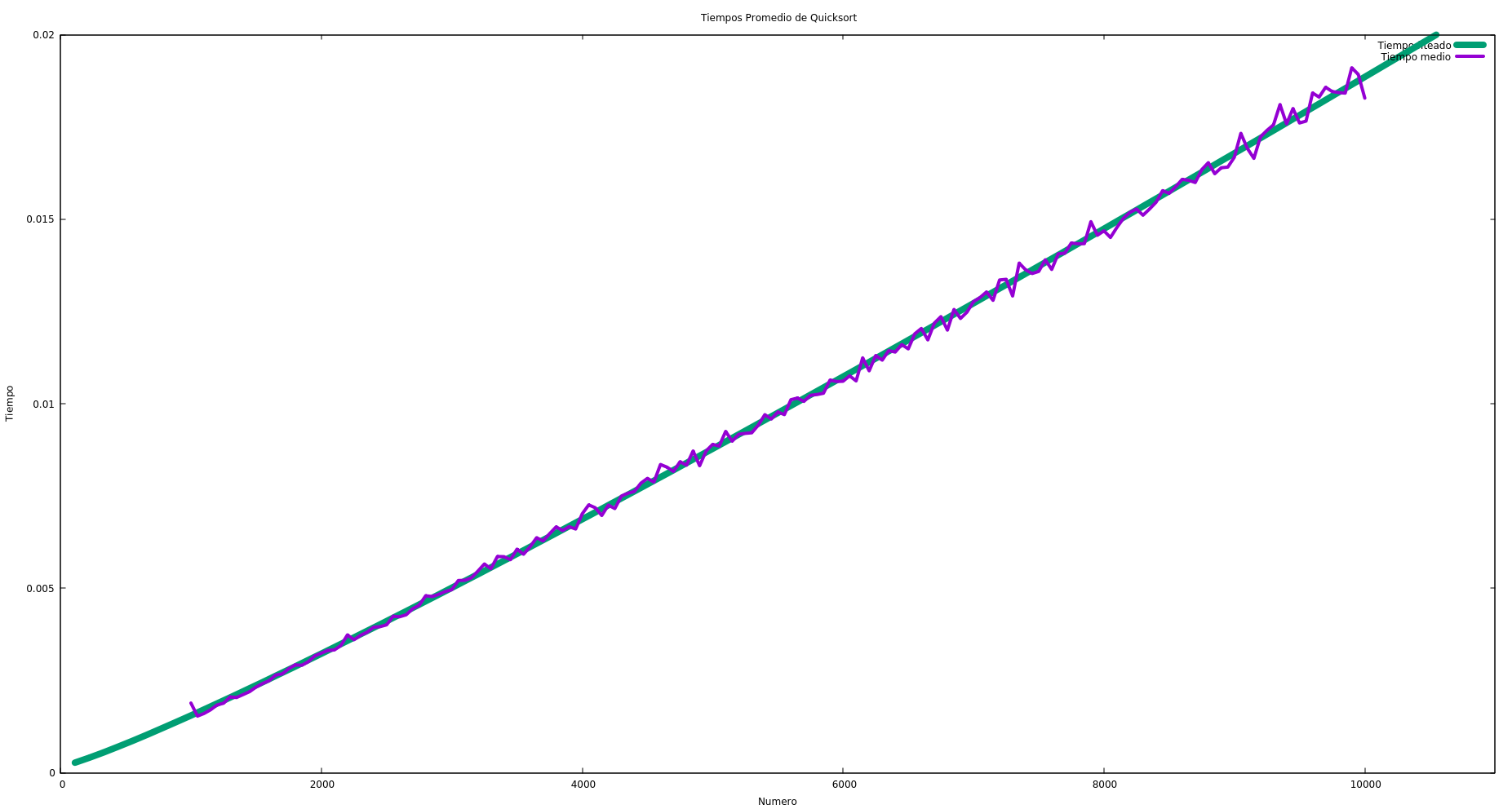
******

En este caso, también se observa que las curvas de tiempos se ajustan a O(N\*log(N)). Los picos existentes se deben al comportamiento interno del procesador en el momento de realizar la ejecución.

5.4 Apartado 4

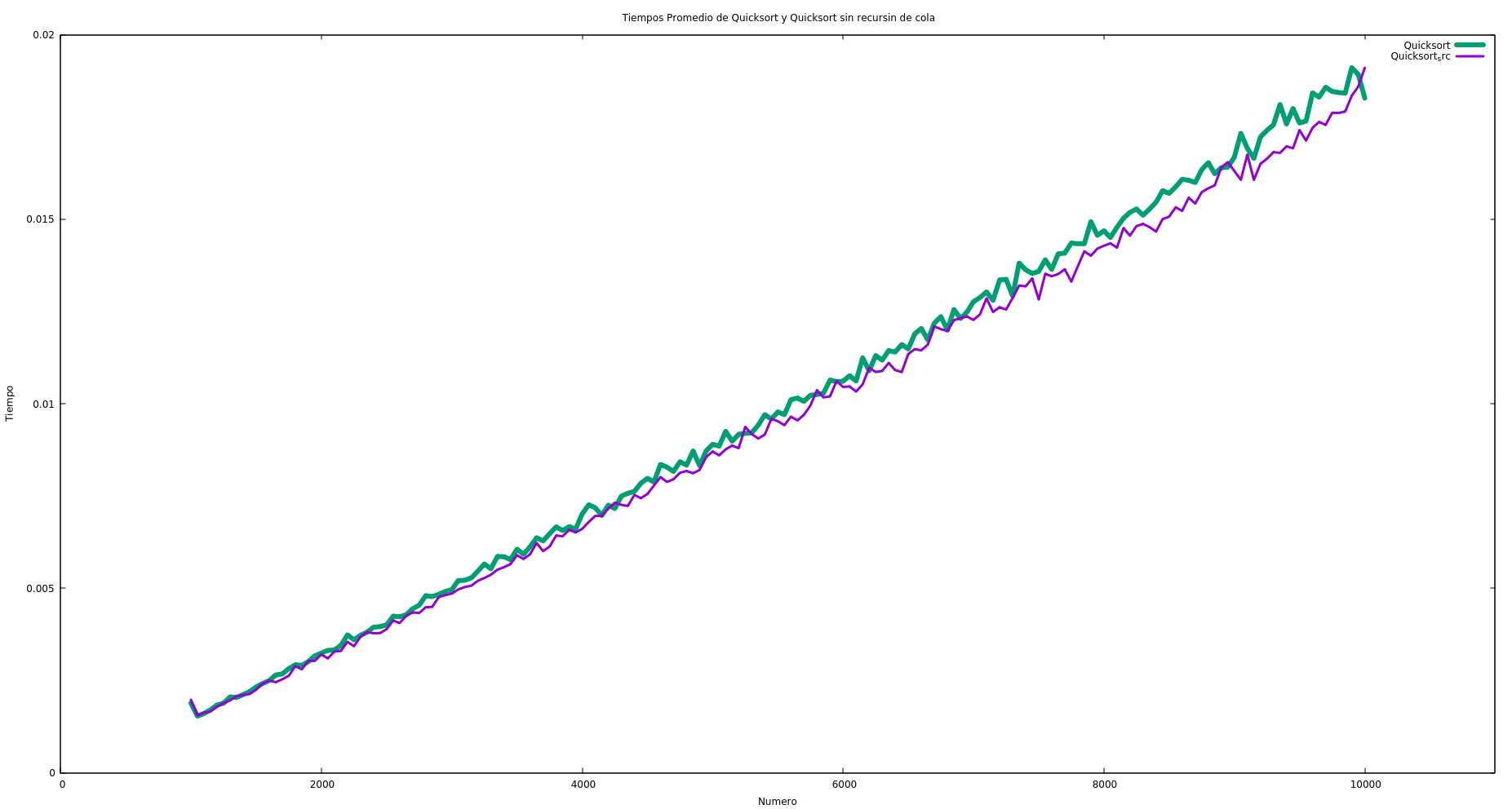
******

En este caso podemos observar diferencias entre las Obs del algoritmo, aunque vemos que se ajustan a los valores teóricos esperados. Estos picos se deben a los casos peores del algoritmo, donde se dispara el número de operaciones que realiza. También notamos picos a la hora de calcular el caso peor cuando recibimos un caso bueno. El tiempo medio se ajusta al valor teórico con el que está fiteado, mientras que los otros casos no, ya que no son O(N\*log(N)).



En este caso, también se observa que las curvas de tiempos se ajustan a O(N\*log(N)). Los picos existentes se deben al comportamiento interno del procesador en el momento de realizar la ejecución.

5.5 Apartado 5

******

**5. Respuesta a las preguntas teóricas.**

5.1 Pregunta 1

Como hemos explicado en anteriores apartados, el rendimiento de los algoritmos se ajusta al caso medio teórico. Los picos existentes en la curva de MergeSort se deben al estado del procesador en el momento de la ejecución, mientras que los de la curva de QuickSort se deben a los casos peores del algoritmo, donde se dispara el número de operaciones que realiza. Esto repercute en el uso de memoria y procesador, especialmente para tablas grandes.

5.2 Pregunta 2

En teoría, si el pivote queda en un extremo de la tabla estamos ante un coste O(N²), es decir, el caso peor. Sin embargo, este coste puede variar según tengamos implementada la elección del pivote. Por ejemplo, en tablas ordenadas, si elegimos como pivote el primer elemento (tal y como lo tenemos nosotros implementado), y nos pasan una ya tabla ordenada, la rutina Partir estaría dejando a la izquierda del pivote una tabla vacía, y a la derecha todo lo demás, lo cual es ineficiente.

Si el pivote queda en el medio de la tabla, estamos ante el mejor caso y la complejidad del algoritmo es O(N\*log(N)), ya que las tablas que quedan en ambos lados del pivote tienen los mismos elementos.

Si el pivote queda en el resto de posiciones, estamos ante el caso medio, de complejidad O(N\*log(N)).

En la práctica, se nos ha explicado que durante este curso no vamos a implementar una función que te devuelva un pivote que no sea el primer elemento de la tabla, por eso no existen diferencias apreciables en nuestras gráficas.

5.3 Pregunta 3

El caso peor de MergeSort es O(N\*log(N)). Para calcular su caso peor en la práctica, tendríamos que pasarle una lista cuyos elementos estén entremezclados de tal manera que en la extracción de las se tenga que extraer alternativamente un elemento de cada una de las las tablas hasta que solo quede un elemento en la tabla mayor.

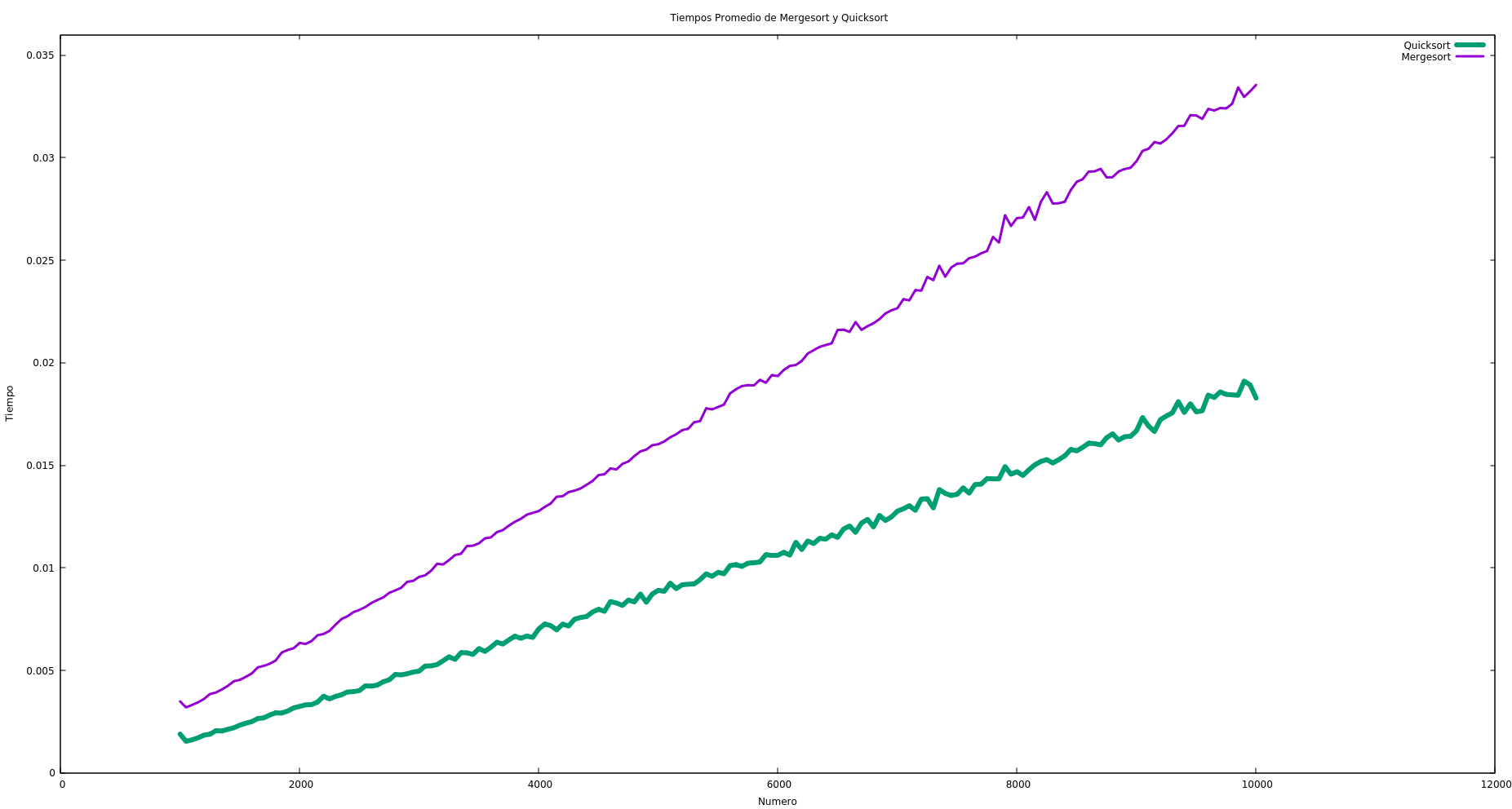
El caso mejor de MergeSort es O(N\*log(N)). Para calcular su caso mejor en la práctica, tendríamos que pasarle una lista ordenada de menor a mayor.

El caso peor de QuickSort es O(N²). Para calcular su caso peor en la práctica, tal como lo tenemos implementado tendríamos que pasarle una lista ordenada de menor a mayor (ya que elegimos como pivote el primer elemento, y esto hace que Partir sea ineficiente).

El caso mejor de QuickSort es O(N\*log(N)). Para calcular su caso mejor en la práctica, tendríamos que pasarle una tabla ordenada de menor a mayor, pero además el pivote tendría que ser el elemento del medio. De esta forma, partir es óptimo.

Para calcular el caso medio, deberíamos coger todas las posibles permutaciones y ver los tiempos que tardan, y eso dividirlo por el número de permutaciones.

5.4 Pregunta 4

******

Si observamos la gráfica de ambos algoritmos, podemos observar que QuickSort es más eficiente que MergeSort. Aunque el orden teórico de ambos algoritmos es O(N\*log(N)), existen ciertos multiplicadores ocultos por el operador O que no estamos teniendo en cuenta (para el caso peor de QuickSort, según como implementemos la elección del pivote, podremos conseguir que el coste de N² sea casi improbable, y se ajuste más a O(N\*log(N))). Estos multiplicadores son mayores para MergeSort que para QuickSort, por eso podemos observar que al final el tiempo de QuickSort es bastante más rápido que para el otro algoritmo.

Si hablamos de gestión de memoria, también es más eficiente QuickSort, ya que MergeSort utiliza memoria auxiliar para crear otra tabla, mientras que QuickSort es in place.

5.5 Pregunta 5

Como se puede observar en la gráfica del apartado 5, en la que se comparan los tiempos de quicksort y quicksort sin recursión de cola, el tiempo medio de ejecución mejora ligeramente. Esto se debe que al quitar la recursión de cola se reduce el número de llamadas a funciones, lo que hace que se reduzca el tiempo de ejecución.