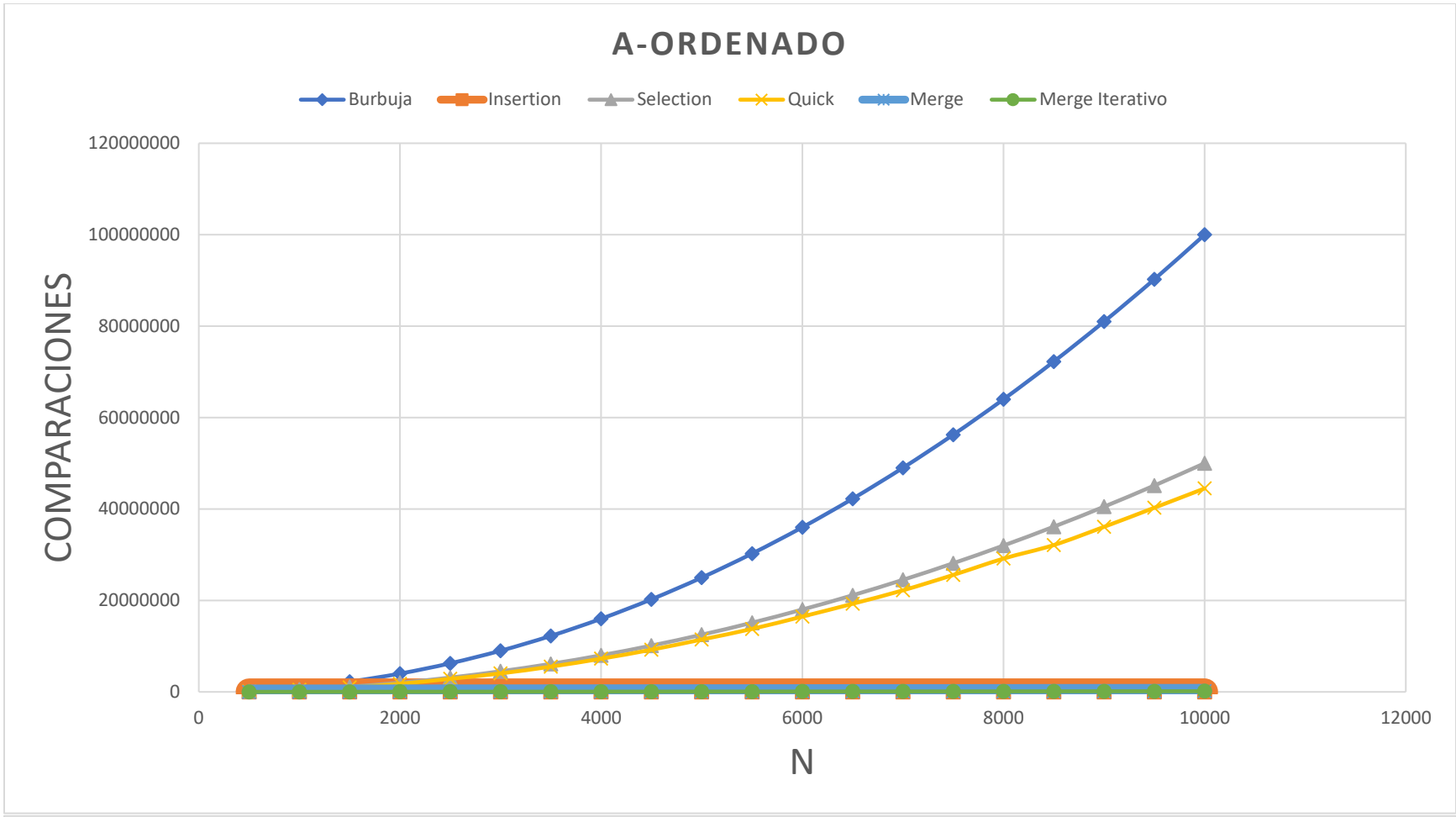
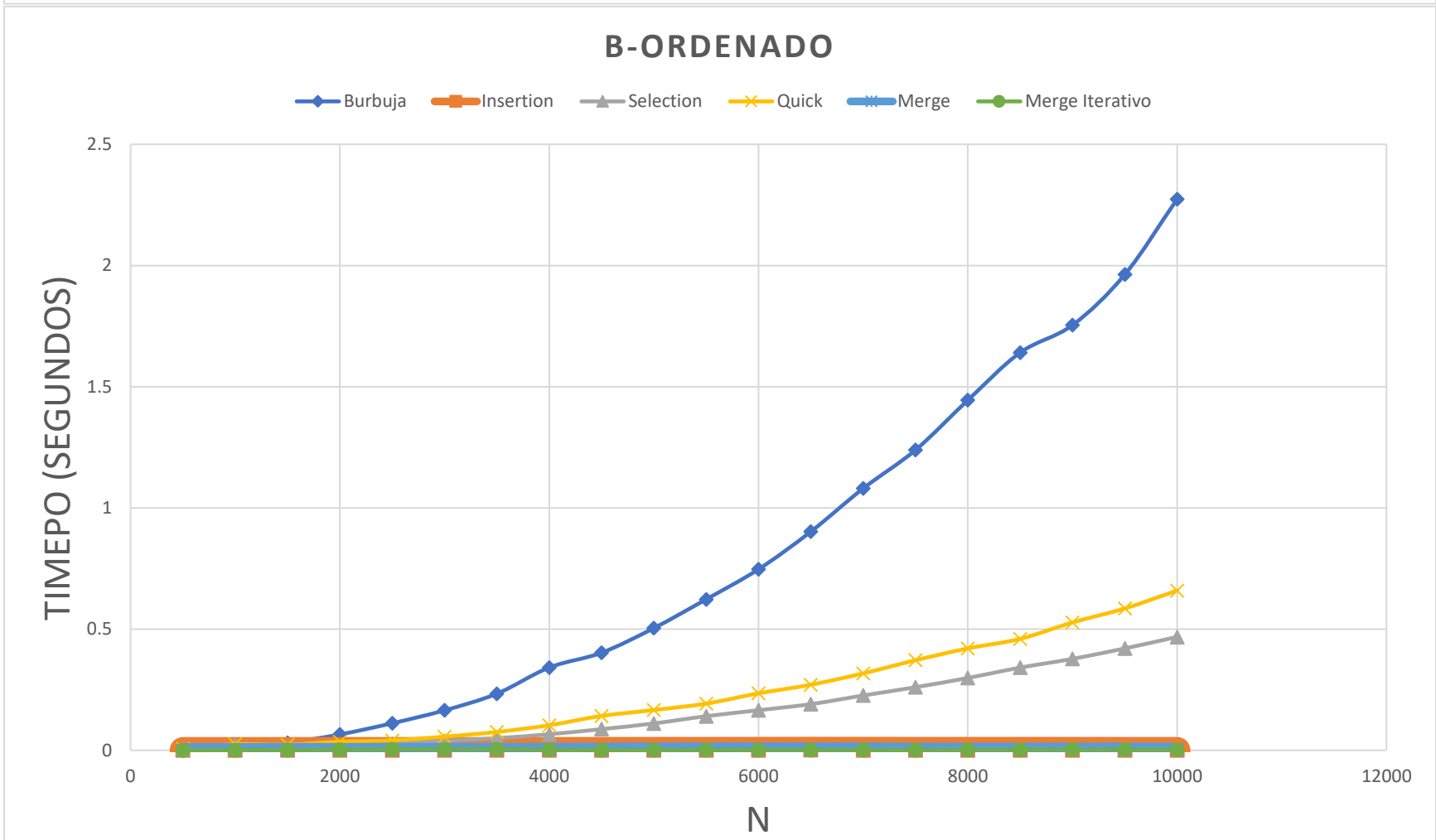
Nombre: Javier Prieto CU: 179563

Materia: Estructuras de Datos Avanzadas

Tarea 1

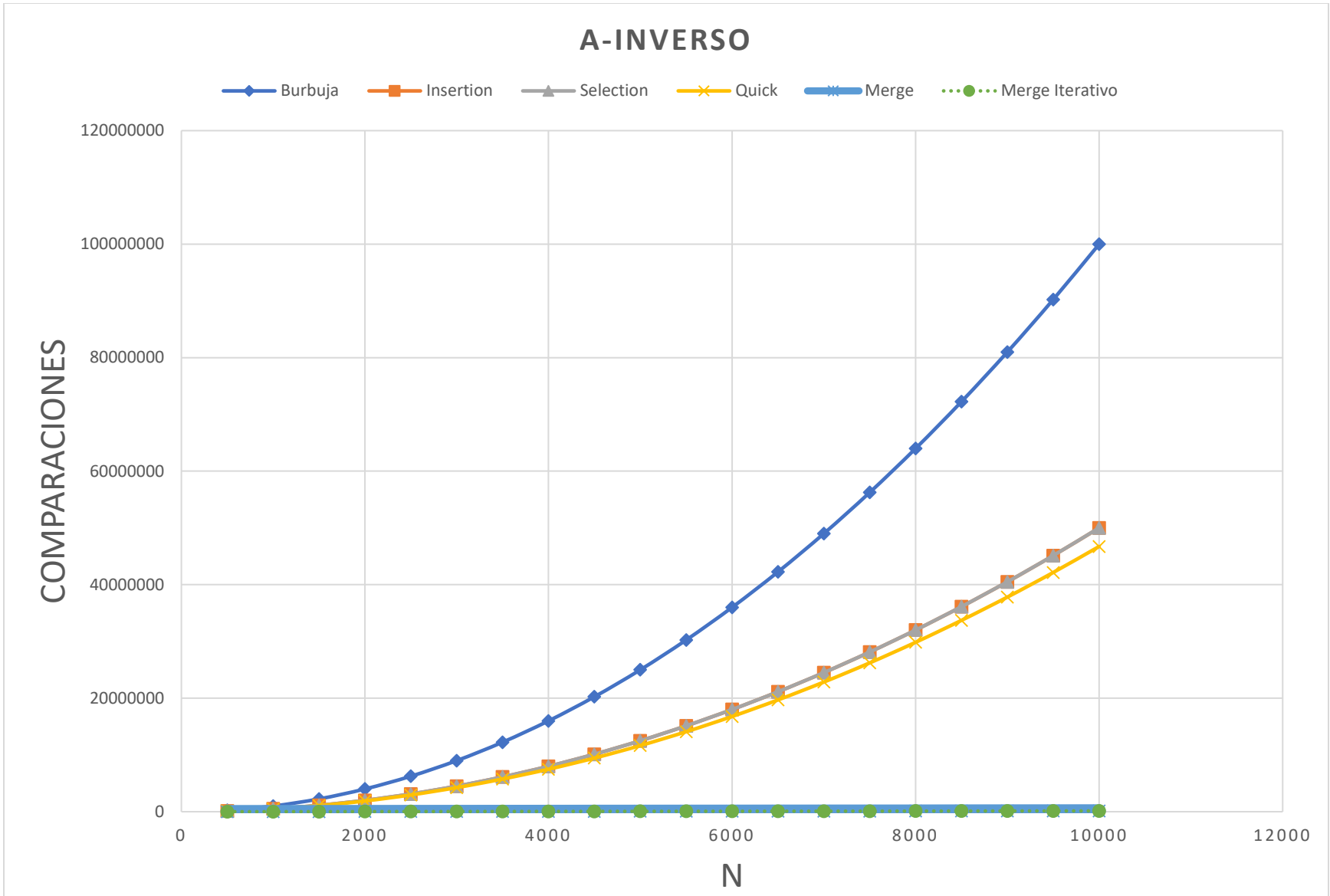
Análisis de Algoritmos de Ordenamiento

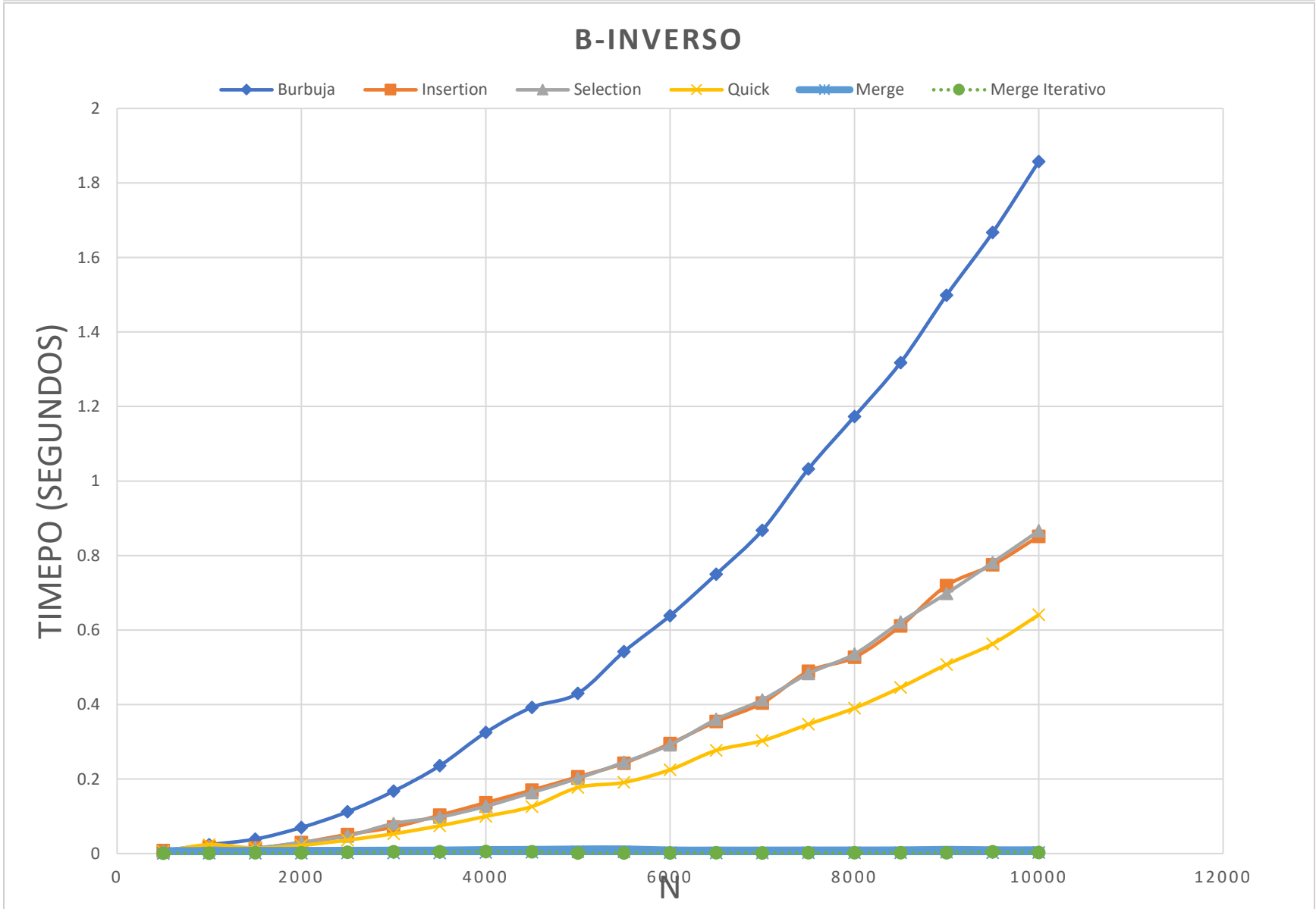




Las 2 gráficas anteriores muestran el desempeño de los 6 algoritmos de ordenamiento vistos en clase al dar como entrada un arreglo ordenado. Es bastante claro a partir de las 2 gráficas que el tiempo que toma a un algoritmo de ordenamiento en ejecutarse es proporcional al número de comparaciones que este realiza. Se puede observar también que la curva de tendencia de todos los algoritmos para el número de comparaciones coincide con la de su complejidad ($O(n^2)$).

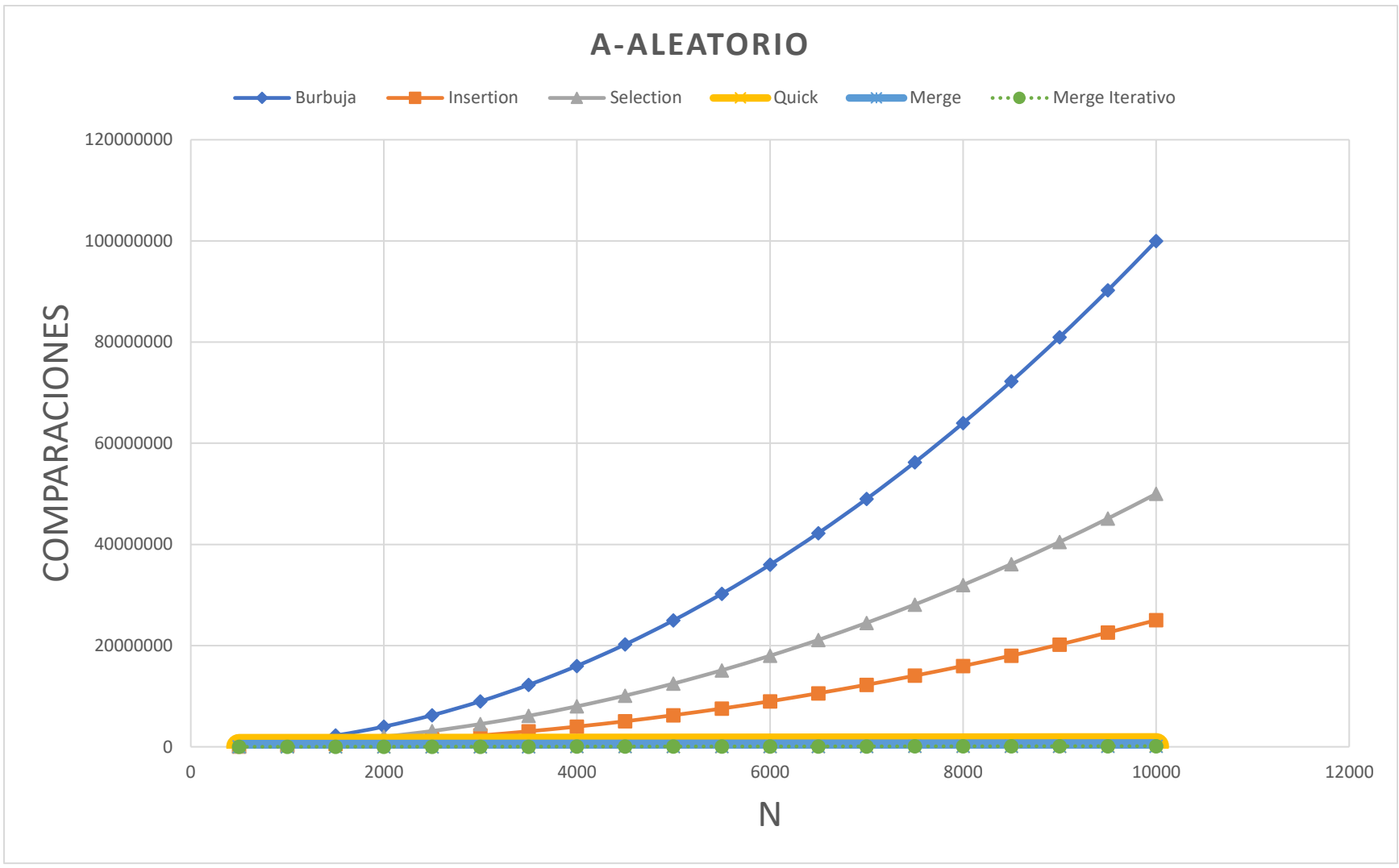
Los 3 algoritmos que menos tiempo tomaron en ejecutarse fueron Insertion Sort, Mergse Sort y la versión iterativa de Merge Sort; es lógico que Merge Soert y su versión iterativa tengan el mismo desempeño en todas las pruebas. Como Insertion Sort interrumpe su ciclo de comparaciones tan pronto encuentre el orden correcto entre cada par de elementos, se corre en tiempo lineal para este caso.

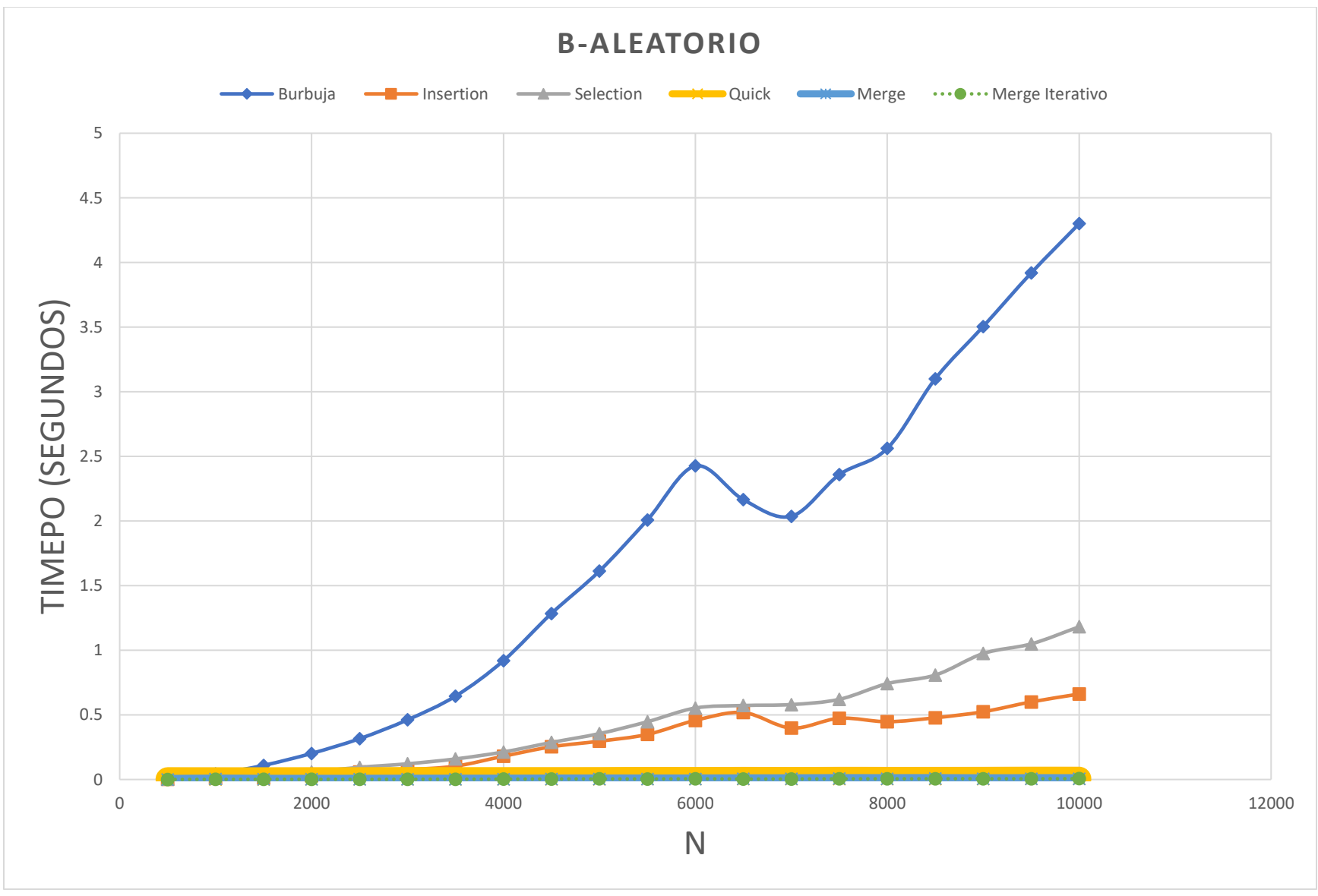




Estas gráficas muestran el desempeño de los algoritmos con un arreglo en orden inverso.

Esta vez, Insertion Sort no tuvo tan buen desempeño como en el caso anterior. En cambio, el resto de los algoritmos tuvieron el mismo comportamiento que en el caso pasado.


A-ALEATORIO
Burbuja
Insertion
Selection
Quick
Merge
Merge Iterativo
COMPARACIONES
120000000
100000000
80000000
60000000
40000000
20000000
0
0
2000
4000
6000
8000
10000
12000
N


B-ALEATORIO
Burbuja
Insertion
Selection
Quick
Merge
Merge Iterativo
TIMEPO (SEGUNDOS)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
2000
4000
6000
8000
10000
12000
N

Por último, estas 2 gráficas muestran el desempeño promedio de los algoritmos con un conjunto de datos aleatorios como entrada.

Burbuja se comportó exactamente igual que en los 2 casos anteriores.

Insertion Sort tiene un mejor desempeño en promedio que en el caso inverso, pero sigue una tendencia cuadrática.

Las gráficas muestran algo interesante, Quicksort se desempeña mucho mejor que en los casos anteriores; muy parecido a los algoritmos de Merge Sort. Por esta razón, la función de sort por defecto para datos primitivos en Java es un Quicksort, pues, aunque los Merge Sort sigan siendo más eficientes en cuanto a tiempo de ejecución, estos requieren de memoria lineal.