Análise 1 (Gráfico):

```
~/edbatv-unidade2/Exercicio 7$ ./Main
insertionSort para 100 elementos, tempo de execução: 2450 ns, passou? = Sim
mergeSort para 100 elementos, tempo de execução: 7150 ns, passou? = Sim
countingSort para 100 elementos, tempo de execução: 810 ns, passou? = Sim
insertionSort para 1000 elementos, tempo de execução: 228840 ns, passou? = Sim
mergeSort para 1000 elementos, tempo de execução: 94840 ns, passou? = Sim
countingSort para 1000 elementos, tempo de execução: 6710 ns, passou? = Sim
insertionSort para 10000 elementos, tempo de execução: 19529709 ns, passou? = Sim
mergeSort para 10000 elementos, tempo de execução: 1167889 ns, passou? = Sim
countingSort para 10000 elementos, tempo de execução: 50429 ns, passou? = Sim
insertionSort para 100000 elementos, tempo de execução: 1725818769 ns, passou? = Sim
mergeSort para 100000 elementos, tempo de execução: 11669049 ns, passou? = Sim
countingSort para 100000 elementos, tempo de execução: 784720 ns, passou? = Sim
insertionSort para 500000 elementos, tempo de execução: 44130542262 ns, passou? = Sim
mergeSort para 500000 elementos, tempo de execução: 68426005 ns, passou? = Sim
countingSort para 500000 elementos, tempo de execução: 4530800 ns, passou? = Sim
```

Figura 1 - Print da execução do exercício 7

Conforme visível na figura 1, o exercício 7 consta na realização de testes dos algoritmos implementados nos exercícios 3, 4 e 5 para diferentes quantidades de elementos no vetor. Os algoritmos implementados foram insertionSort, mergeSort e countingSort para os exercícios 3, 4 e 5 respectivamente. A seguir, o gráfico que relaciona o tempo de execução com a quantidade de elementos dos algoritmos implementados.

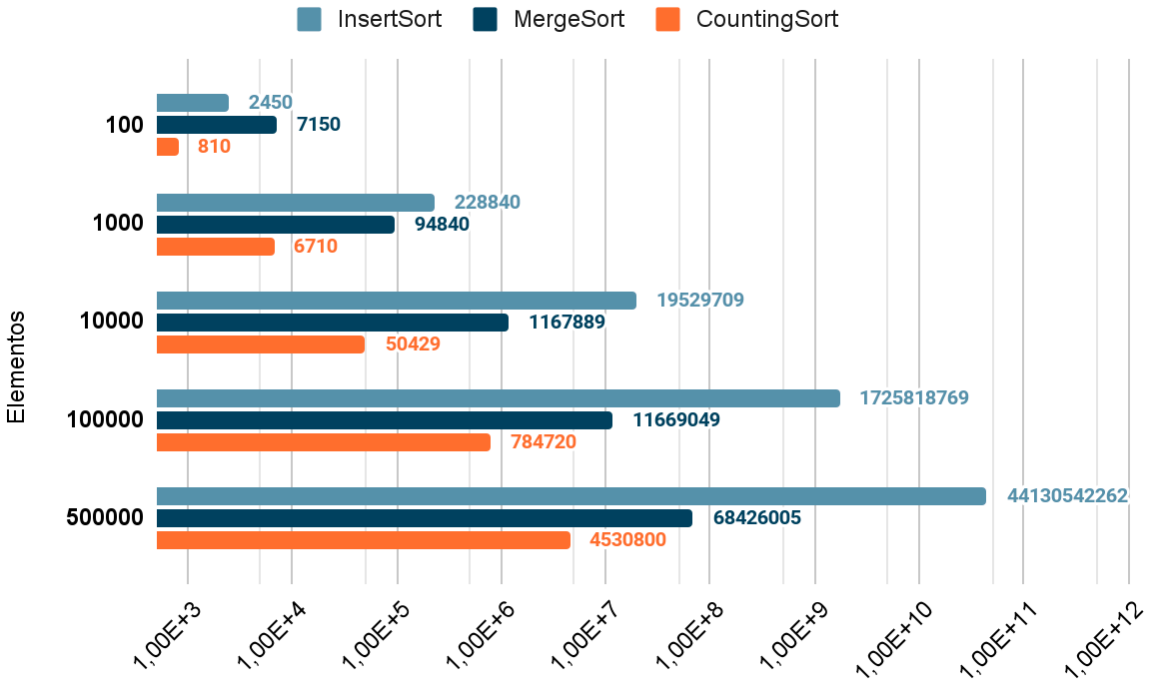

Figura 2 - Gráfico dos tempos de execução (tempo em nanosegundos)

Análise 2 (Tempo de execução x Complexidade):

1. **InsertionSort**

   Complexidade teórica: $O(n^2)$

   O tempo de execução do insertionSort aumenta rapidamente conforme aumenta o número de elementos de entrada, é possível notar isso comparando-o com o mergeSort para 100 e 1.000 elementos. Enquanto o insertionSort cresce de 2.450 ns para 228.840 ns (diferença de 226.390 ns), o mergeSort cresce de 7.150 ns para 94.840 ns (diferença de 87.690 ns). Isso ocorre pois a complexidade teorizada do mergeSort é melhor que a do insertionSort, então conforme o número de elementos de entrada aumenta, maiores são as chances do mergeSort ser mais eficiente.

2. **MergeSort**

   Complexidade teórica: $O(n \log n)$

   Observando o gráfico, é possível notar que o tempo de execução do mergeSort é mais eficiente que o do insertionSort, mas inferior ao do countingSort. Isso fica claro ao analisar os maiores valores, para entrada de 100.000 e 500.000 elementos, o crescimento do tempo de execução do mergeSort é muito inferior ao do insertionSort, mas ainda é consideravelmente superior ao do mergeSort.

3. **CountingSort**

   Complexidade teórica:$O(n + k)$

   Fica claro que nos testes realizados o countingSort foi o algoritmo mais eficiente em tempo de execução, não apenas apresentando o menor tempo de execução desde o início, como também apresentando a menor curva de crescimento. Isso se dá devido a sua complexidade teorizada, na qual apresenta uma ótima eficiência.

Conclusão:

Depois de analisar individualmente cada algoritmo e suas curvas de tempo de execução, é possível confirmar o que foi teorizado em sala de aula, as notações de complexidade são diretamente proporcionais ao tempo de execução prático observado nos testes que foram realizados.