Trabalho 1 PAA

Arthur Koichi Nakao, Cristian Eidi Yoshimura

Outubro 2024

Sumário

1 Introdução

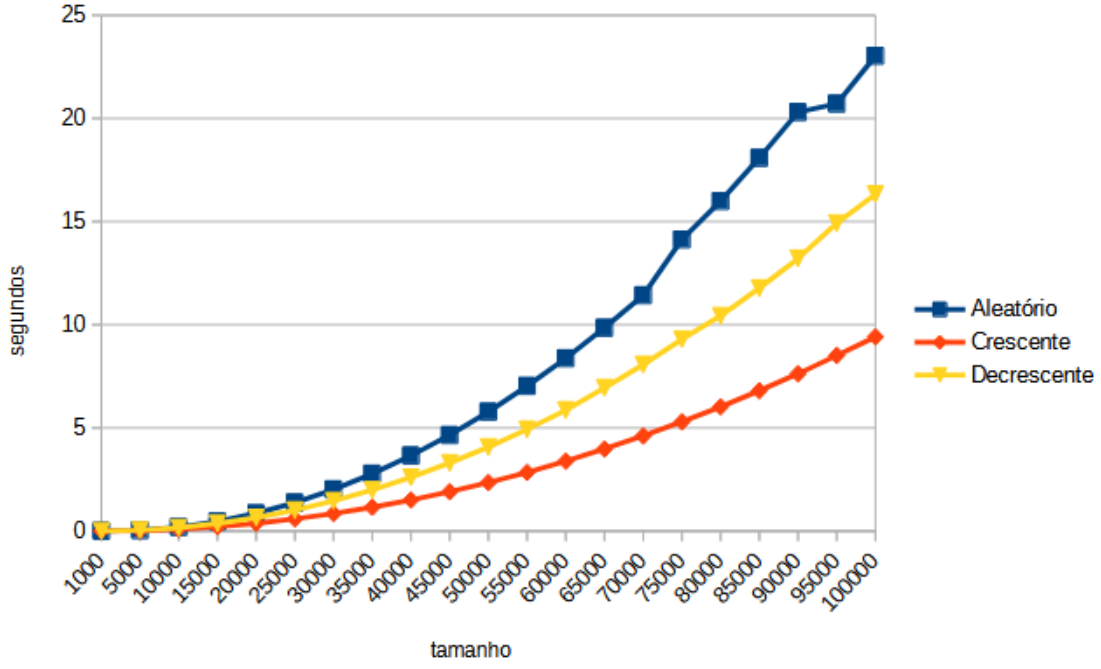
Esse trabalho visa realizar a análise de diferentes tipos de algoritmos de ordenação. Esses são o Bubble sort(sem melhoramento e com melhoramento), Quick sort(pivô no primeiro elemente e no primeiro elemento), Insertion sort, Shell sort, Selection sort, Heap sort e Merge sort. Os algoritmos foram feitos em linguagem C e foram testados 100 vezes para cada tamanho de vetor.

2 Análise dos algoritmos

2.1 Bubble sort

Conforme feito em aula, a complexidade do bubble sort em seu melhor, pior e médio caso é $\Theta(n^2)$.

imagem do gráfico:



2.2 Bubble sort melhorado

A melhoria do bubble sort diminui a complexidade de maneira significante somente se o vetor já estiver semi ordenado(no caso os testes de melhor caso foram feitos com vetores já ordenados), pois se um vetor estiver realmente aleatorizado, é mais provavel que o mesmo só ficará ordenado nos últimos laços de repetição do código.

Concluindo-se que o melhor caso é $\Theta(n)$ e seu pior caso, como também o caso médio é $\Theta(n^2)$

Imagem do gráfico:

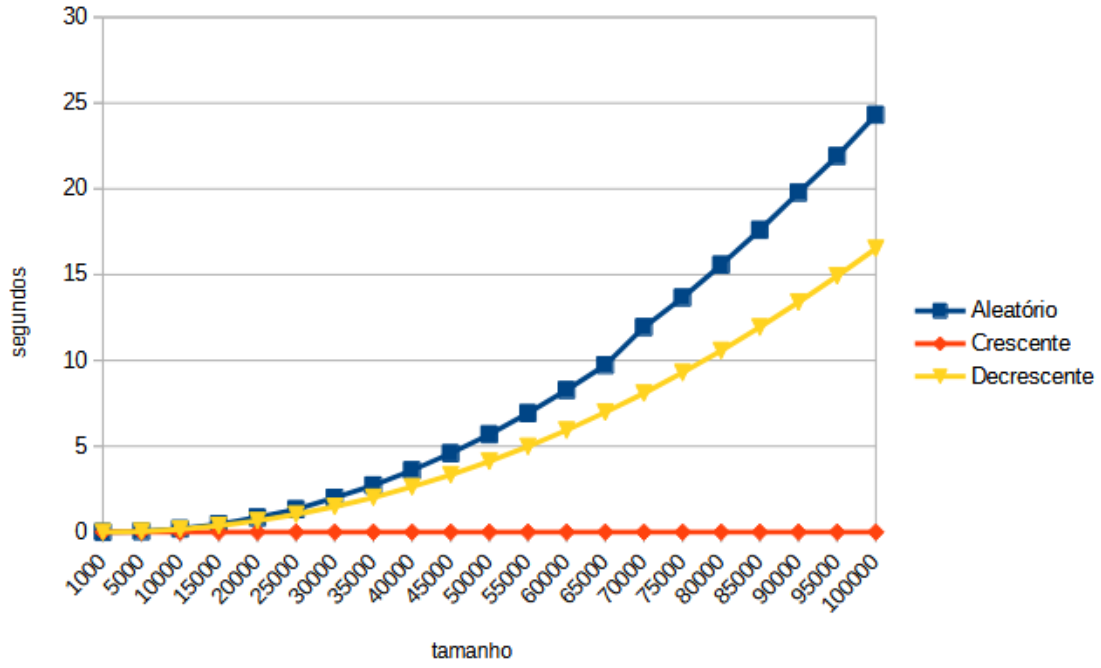
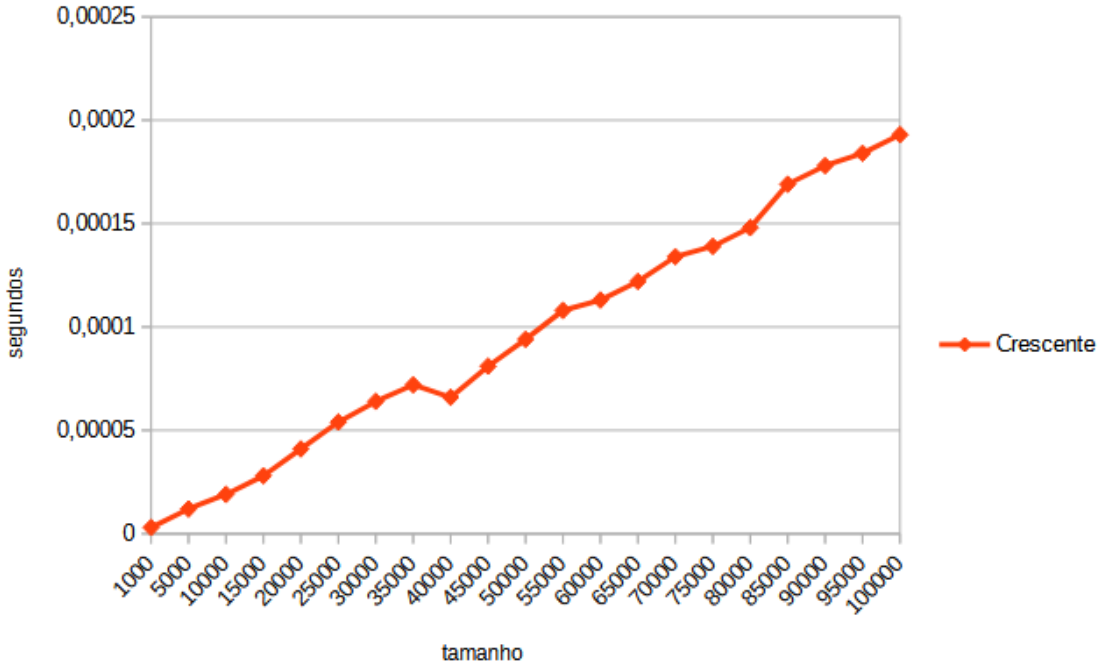


imagem sem as ordenações de complexidade $\Theta(n^2)$:



2.3 Quick sort com pivô no primeiro elemento

Análise também feita em aula, onde no somente o pior caso possui complexidade $\Theta(n^2)$, pois uma das partições possui um tamnho muito pequeno e os outros casos possuem complexidade $\Theta(n \log n)$.

imagem do gráfico:

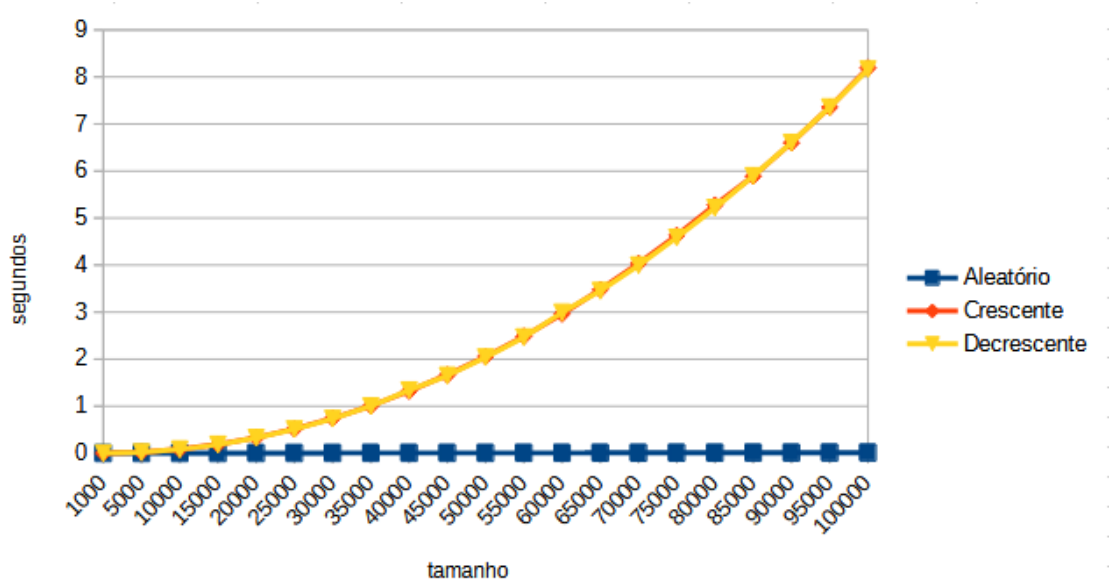
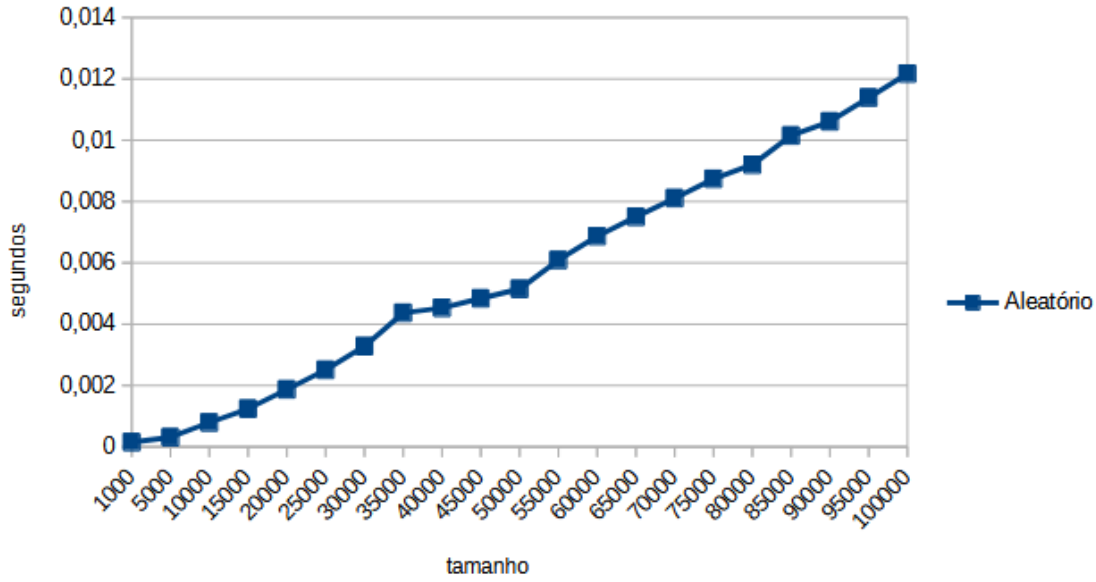


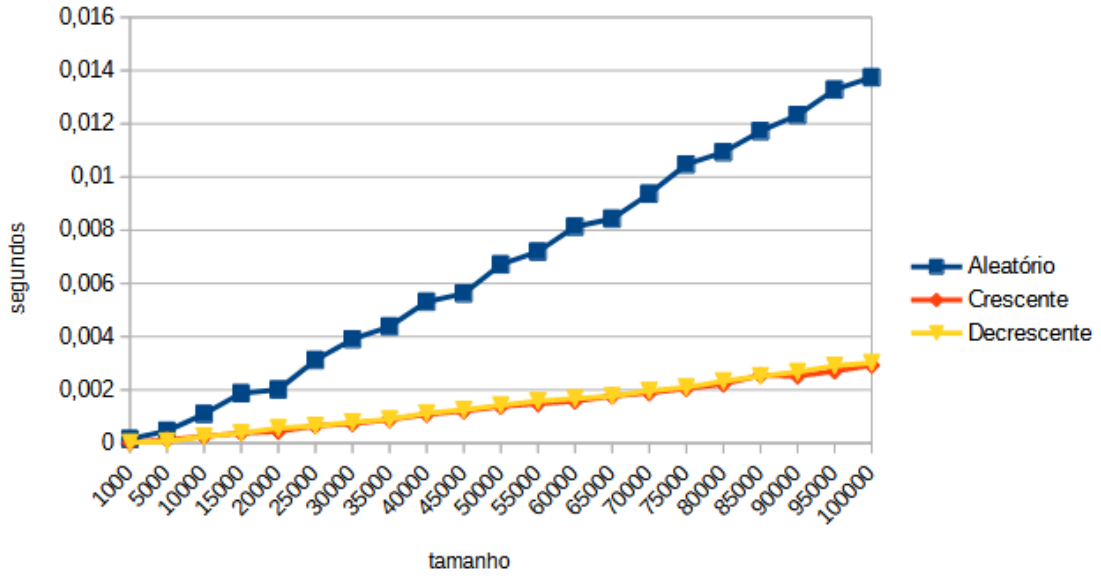
imagem sem os casos de complexidade $\Theta(n^2)$:



2.4 Quick sort com pivô no elemento central

A análise do quick sort com pivô no elemento central é semelhante ao quick sort com pivô no primeiro elemento, por isso seu pior caso é $\Theta(n^2)$ e seu melhor e médio é $\Theta(n \log n)$, porém seu pior caso não ocorre mais, em um vetor já ordenado.

imagem do gráfico:



2.5 Insertion sort

Em aula o algoritmo insertion sort foi analisado onde seu melhor caso é um vetor já ordenado, pois não há necessidade de trocar o valor dos elementos do vetor, assim possuindo complexidade $\Theta(n)$, e nos casos restantes a complexidade é $\Theta(n^2)$.

imagem do gráfico:

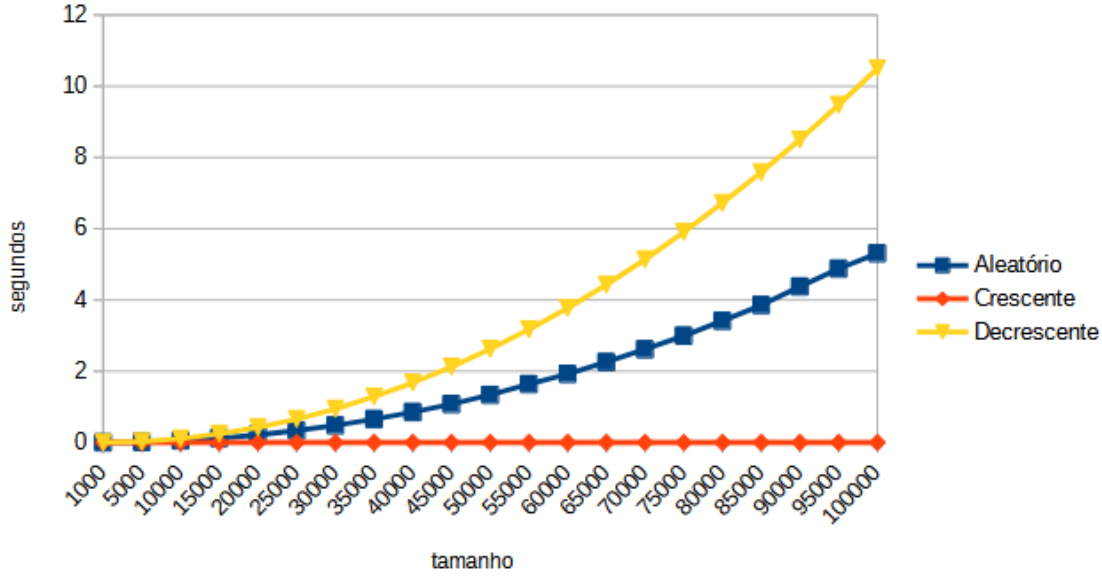
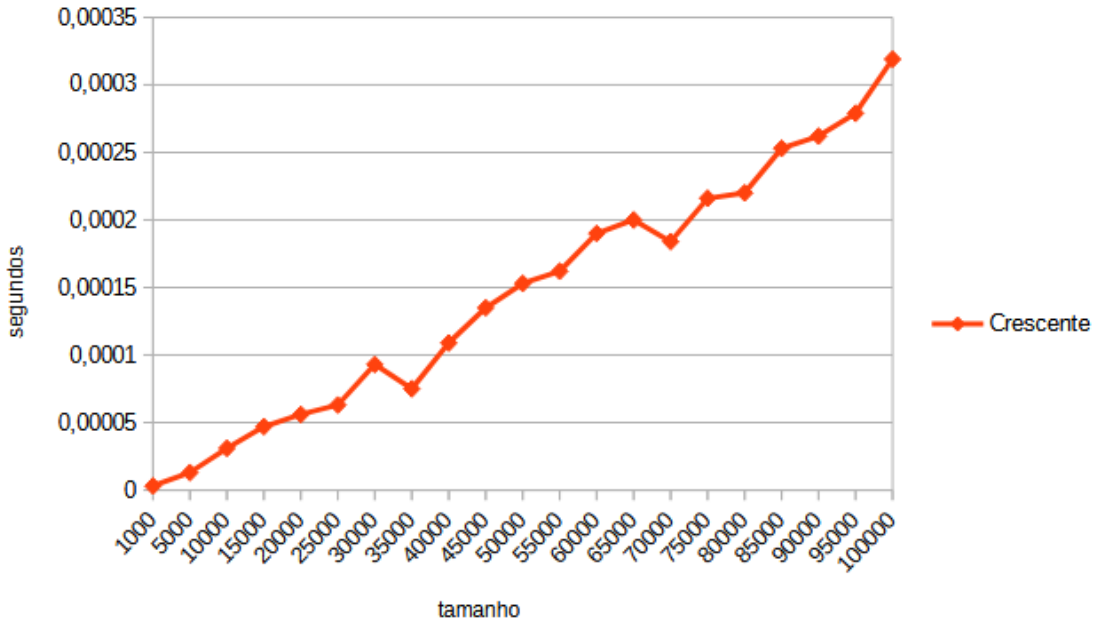


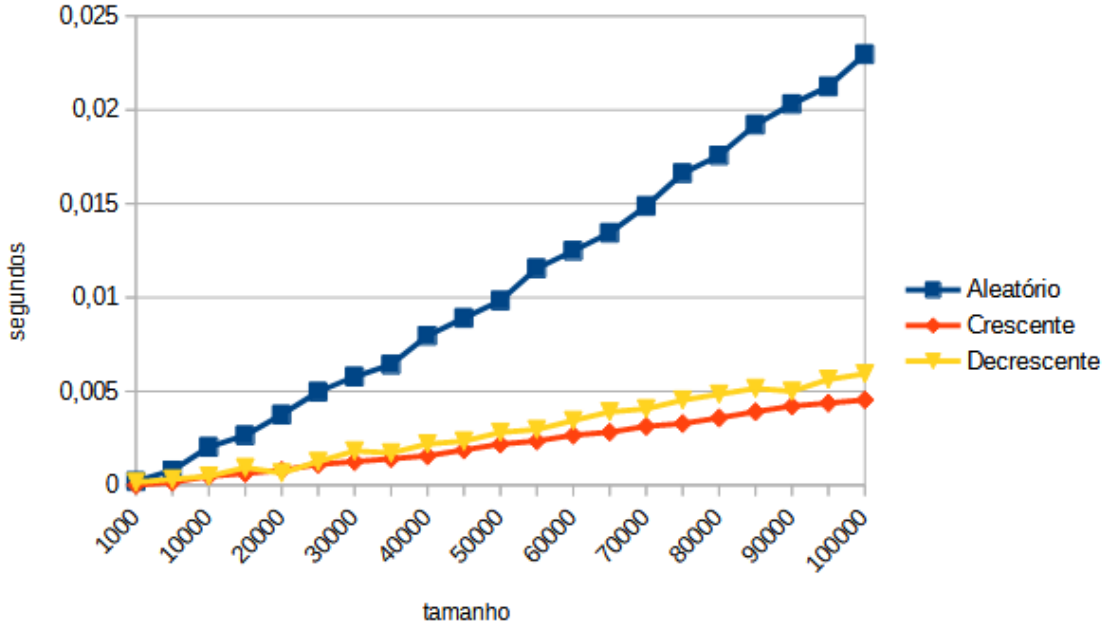
imagem sem as ordenações de complexidade $\Theta(n^2)$:



2.6 Shell sort

Foi visto em aula que a complexidade do shell sort usando a razão $2^k - 1$ é $\Theta(n^{\frac{3}{2}})$.

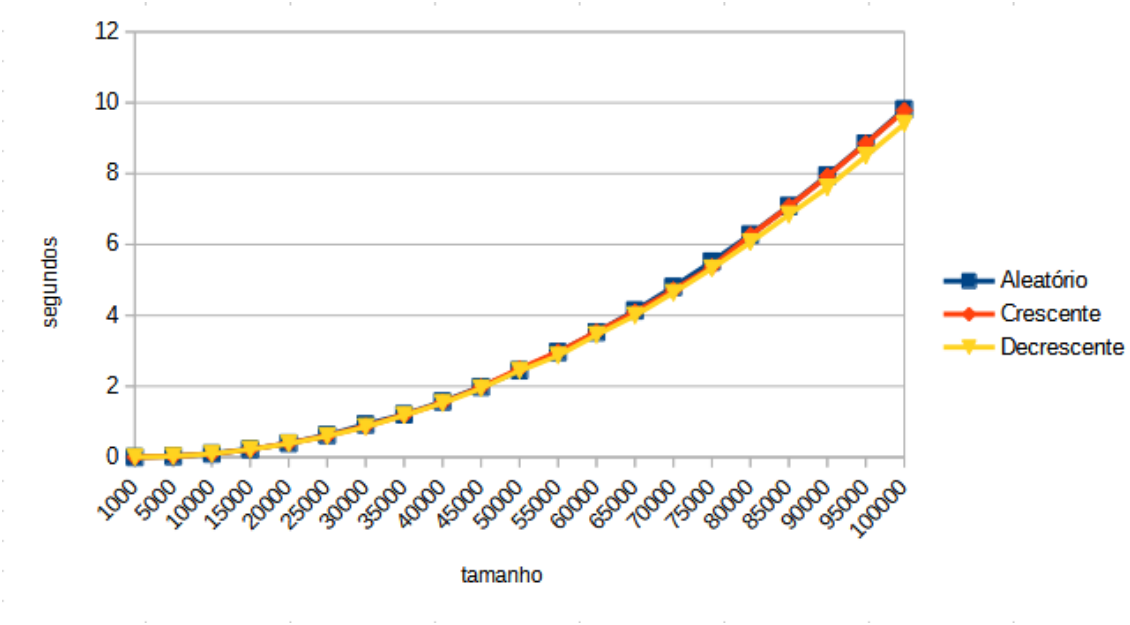
imagem do gráfico:



2.7 Selection sort

Conforme estudado em aula o insertion sort possui complexidade $\Theta(n^2)$ em todos os casos, porque o número de processos segue uma progressão aritmética.

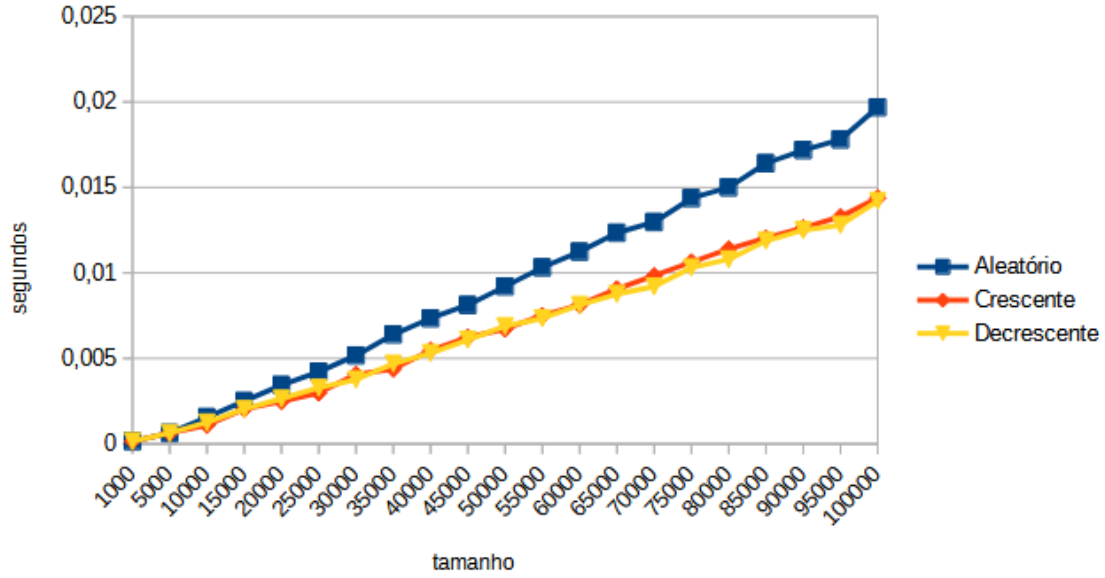
imagem do gráfico:



2.8 Heap sort

Em aula foi visto que a função buildMaxHeap possui complexidade $\Theta(n)$ e a função maxHeapfy tem complexidade $\Theta(\log n)$, porém é repetida $n-1$ vezes. Podendo-se concluir que a complexidade é $\Theta(n \log n)$.

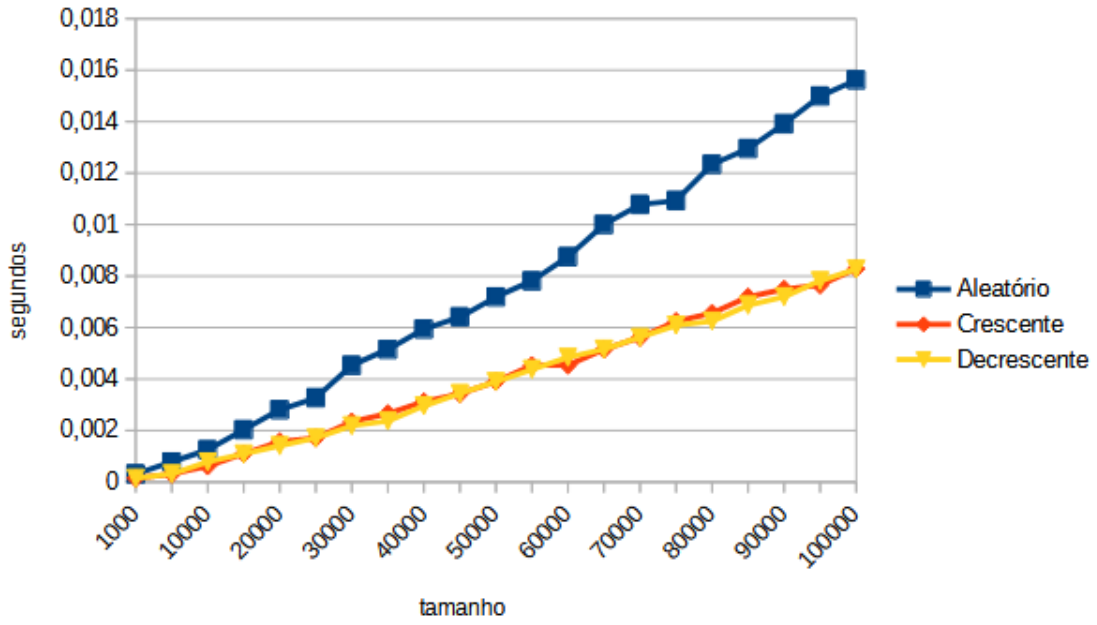
imagem do gráfico:



2.9 Merge sort

Em aula foi utilizado o teorema mestre para calcular a complexidade do merge sort e o resultado foi $\Theta(n \log n)$.

imagem do gráfico:



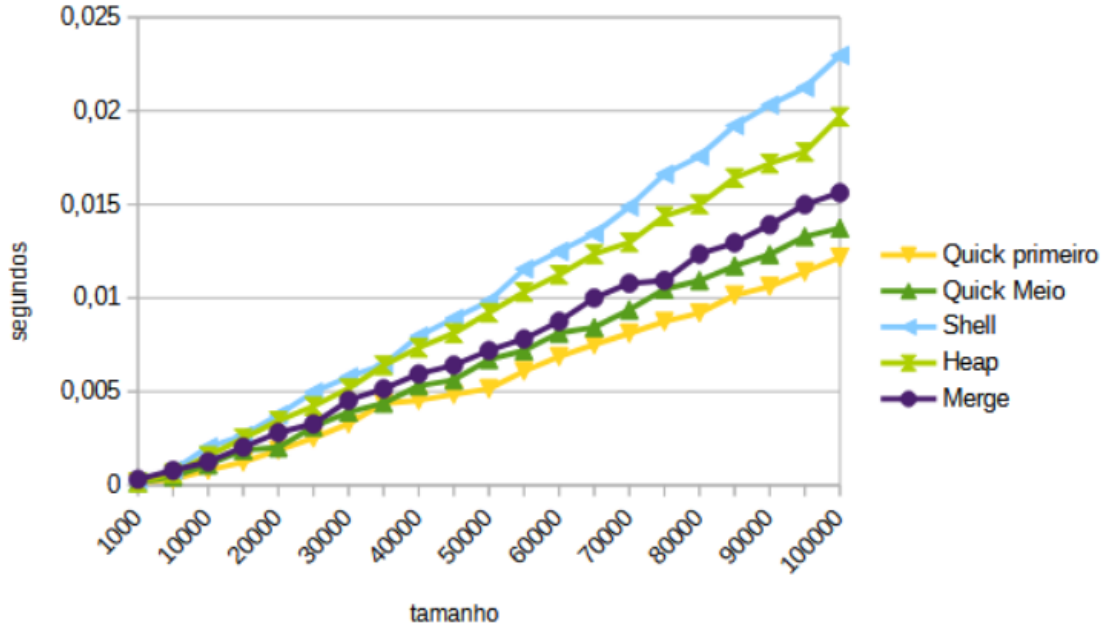
3 Conclusão

3.1 Melhor desempenho

3.1.1 Aleatório

Comparando os algoritmos em ordem aleatória, foi observado que o algoritmo de maior desempenho foi o quick sort com pivô no primeiro elemento.

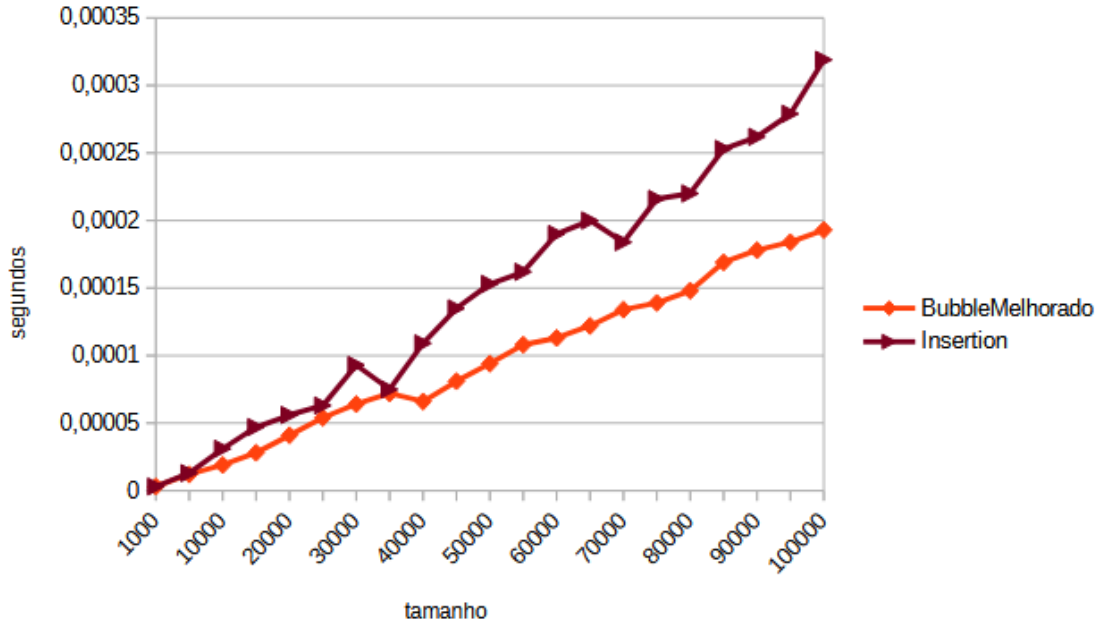
imagem do gráfico dos algoritmos em ordem aleatória(somente os que possuem O($n^{\frac{3}{2}}$) para a escala permitir a visualização):



3.1.2 Crescente

Comparando os algoritmos em ordem crescente, foi observado que o algoritmo de maior desempenho foi o bubble sort melhorado.

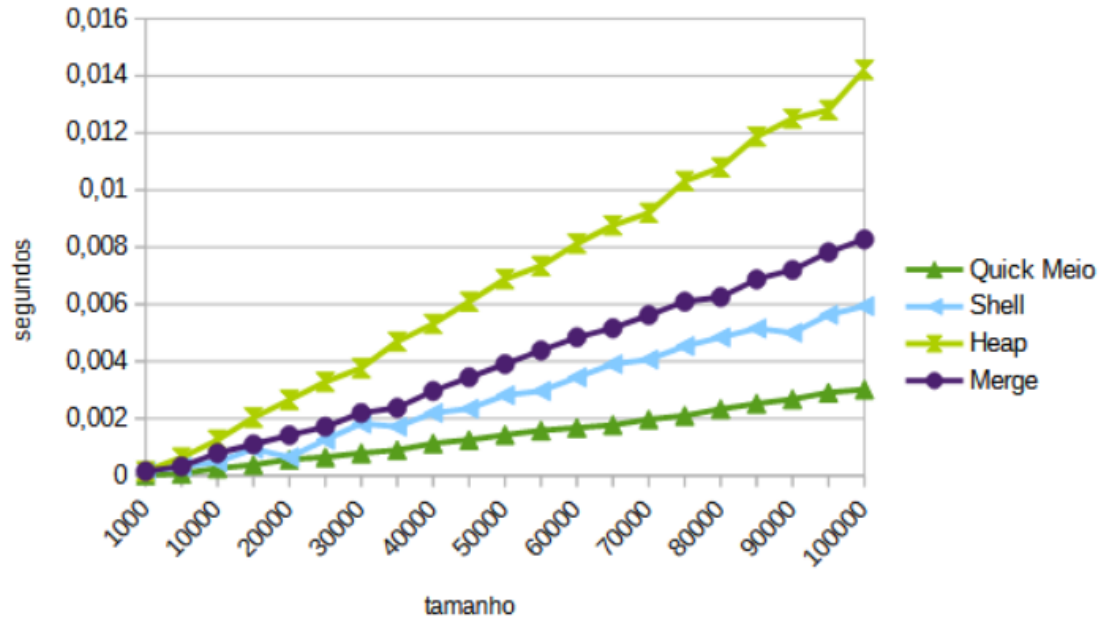
imagem do gráfico dos algoritmos em ordem crescente(somente os que possuem $\Theta(n)$:



3.1.3 Decrescente

Comparando os algoritmos em ordem decrescente, foi observado que o algoritmo de maior desempenho foi o quick sort com pivô no elemento do meio.

imagem do gráfico dos algoritmos em ordem decrescente(somente os que possuem $O(n^{\frac{3}{2}})$:



3.2 Geral

Separação do gráfico dos algoritmos em cada ordem em relação a sua complexidade.

3.2.1 Aleatório

imagem do gráfico com O(n^2):

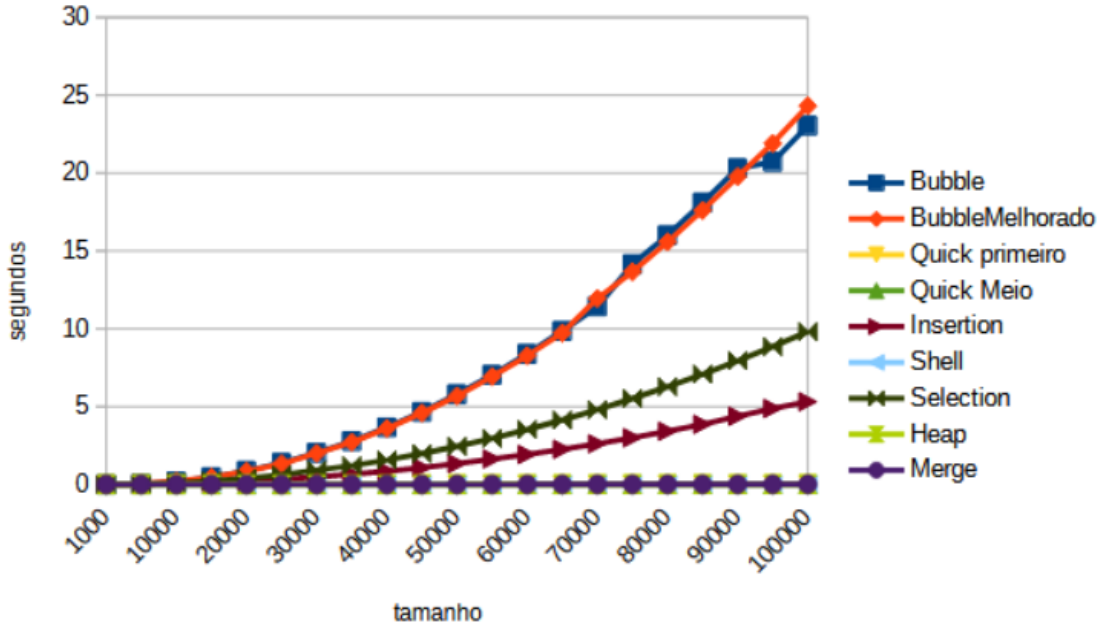
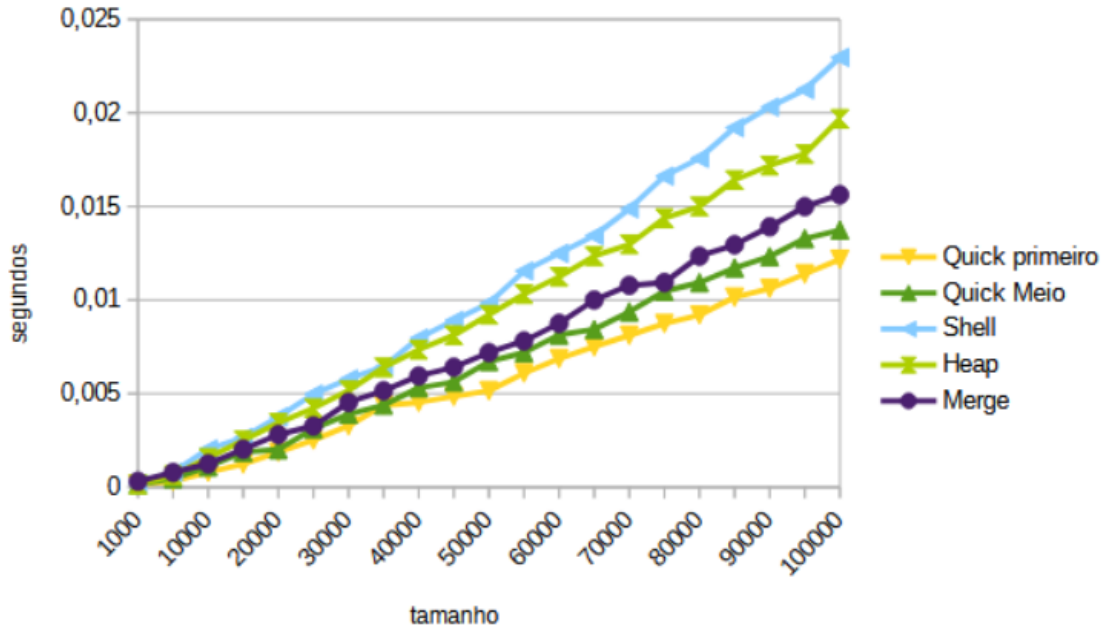


imagem do gráfico com O($n^{\frac{3}{2}}$):



3.2.2 Crescente

imagem do gráfico com O(n^2):

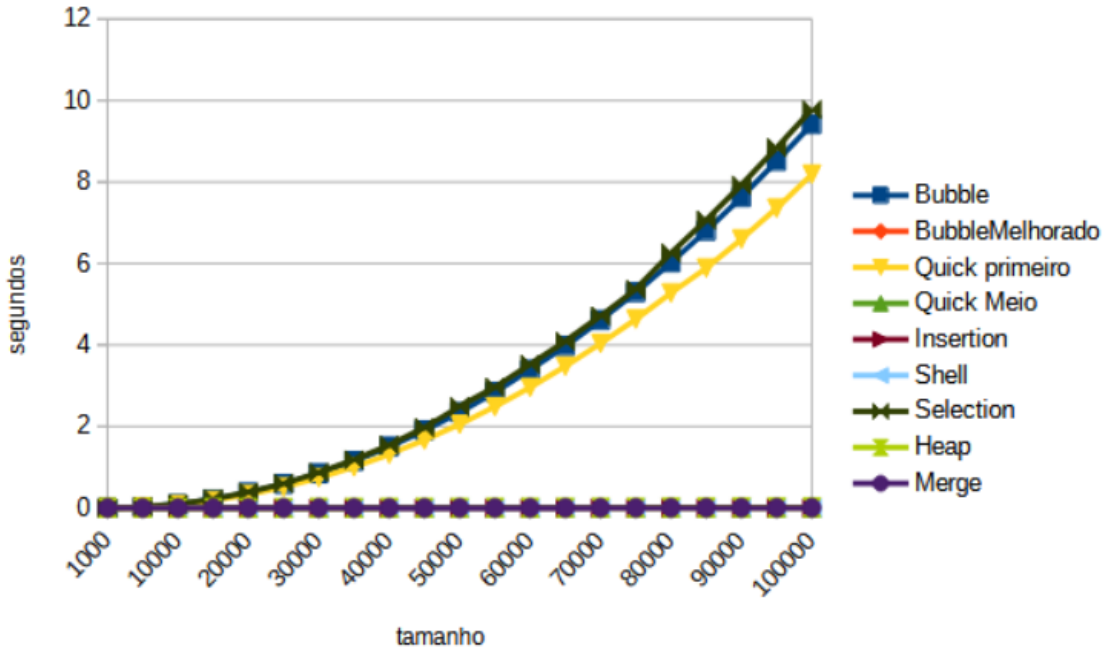


imagem do gráfico com O($n^{\frac{3}{2}}$):

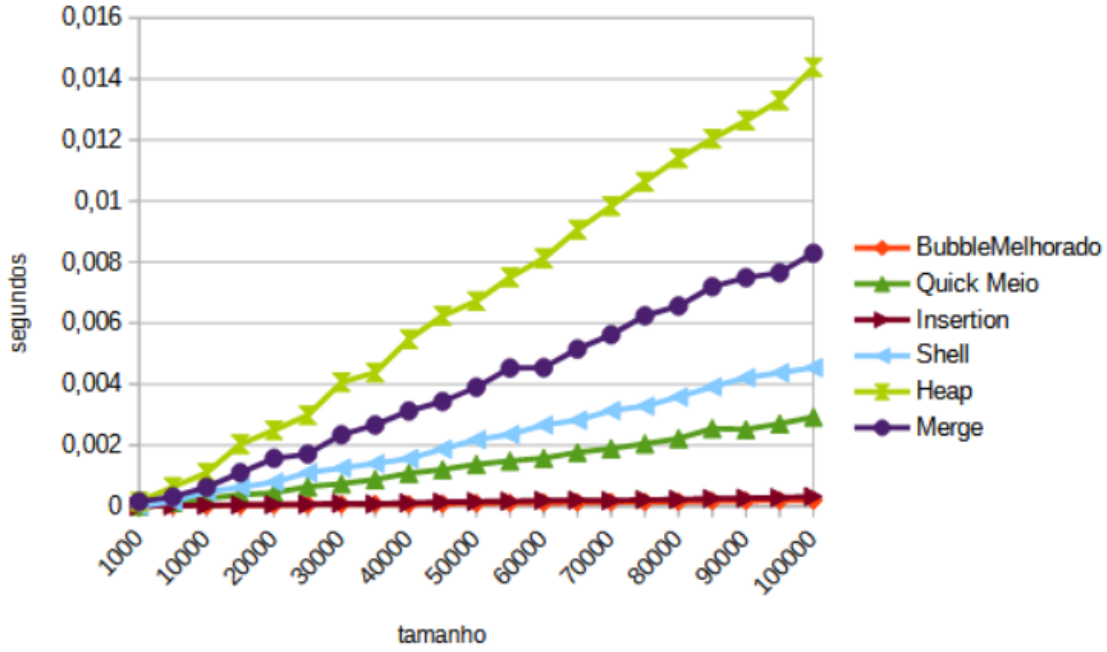
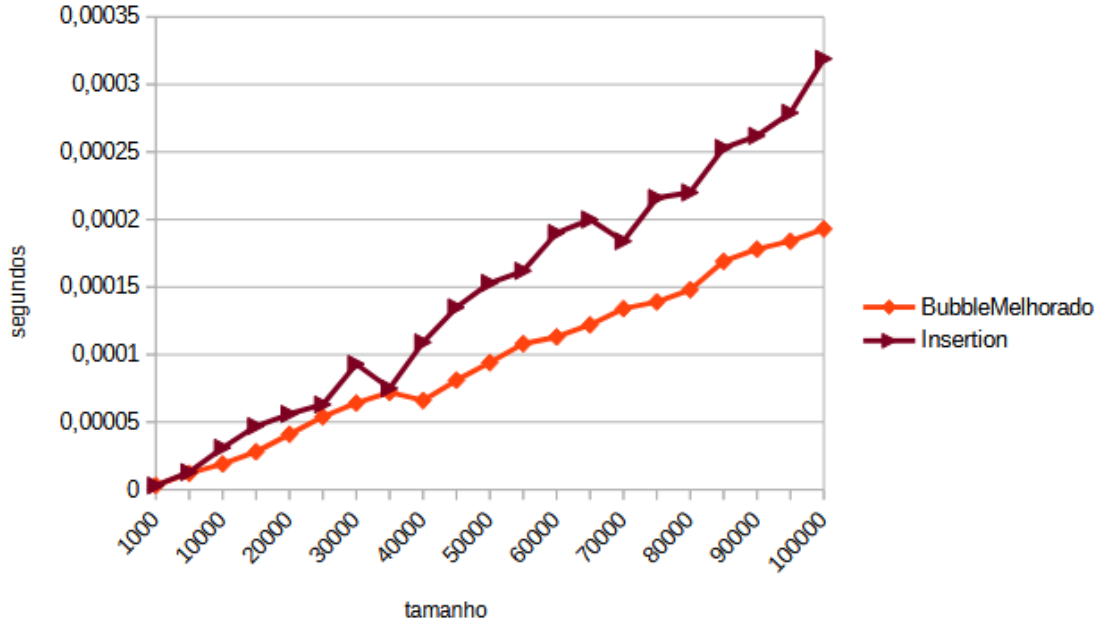


imagem do gráfico com O(n):



3.2.3 Decrescente

imagem do gráfico com O(n^2):

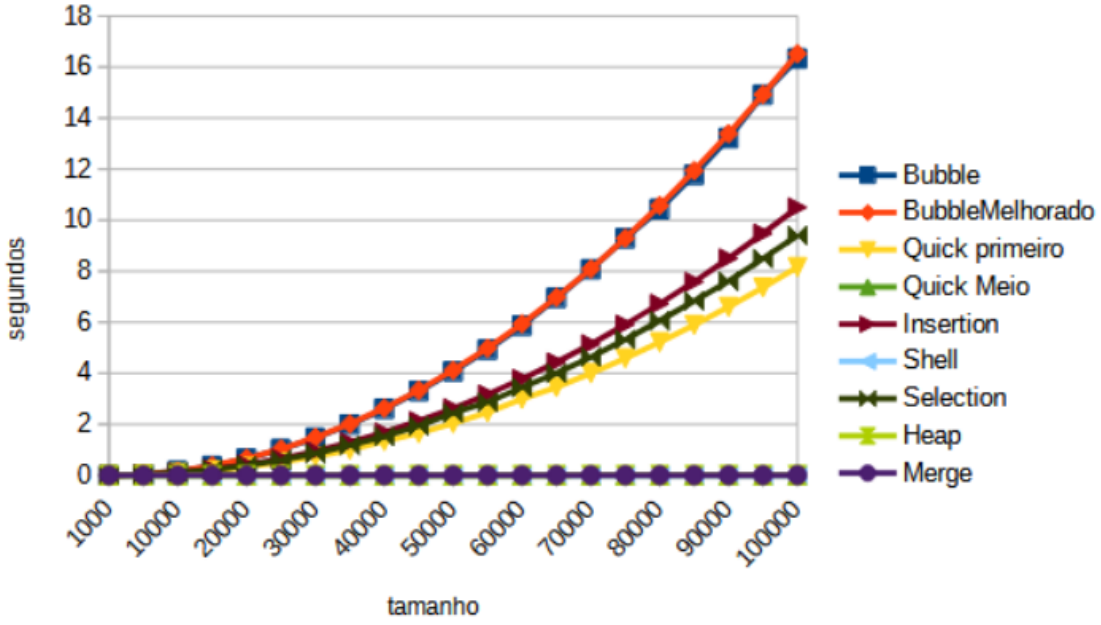
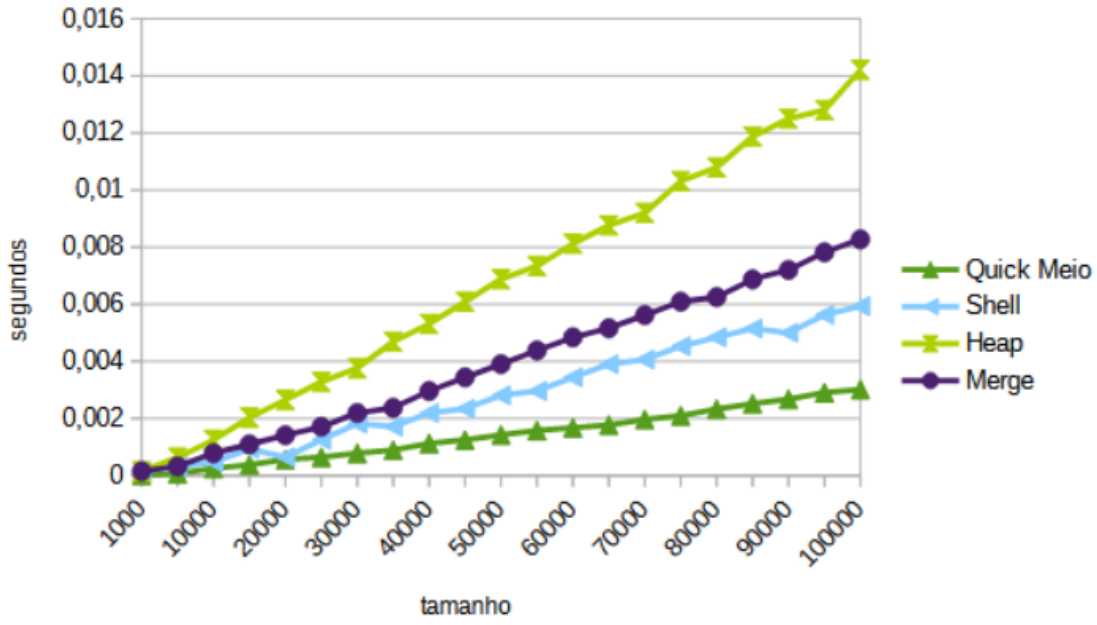


imagem do gráfico com O($n^{\frac{3}{2}}$):


0,016
0,014
0,012
0,01
0,008
0,006
0,004
0,002
0
segundos
1000
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
tamanho
Quick Meio
Shell
Heap
Merge