Implementação algorítmica – Trabalho 1

Aluno: Héber Pereira Garcia

O trabalho a ser apresentado consistiu em implementar e realizar experimentos com alguns algoritmos de ordenação pré-definidos: Selection sort, insertion sort, merge sort, heap sort, quick sort e counting sort.

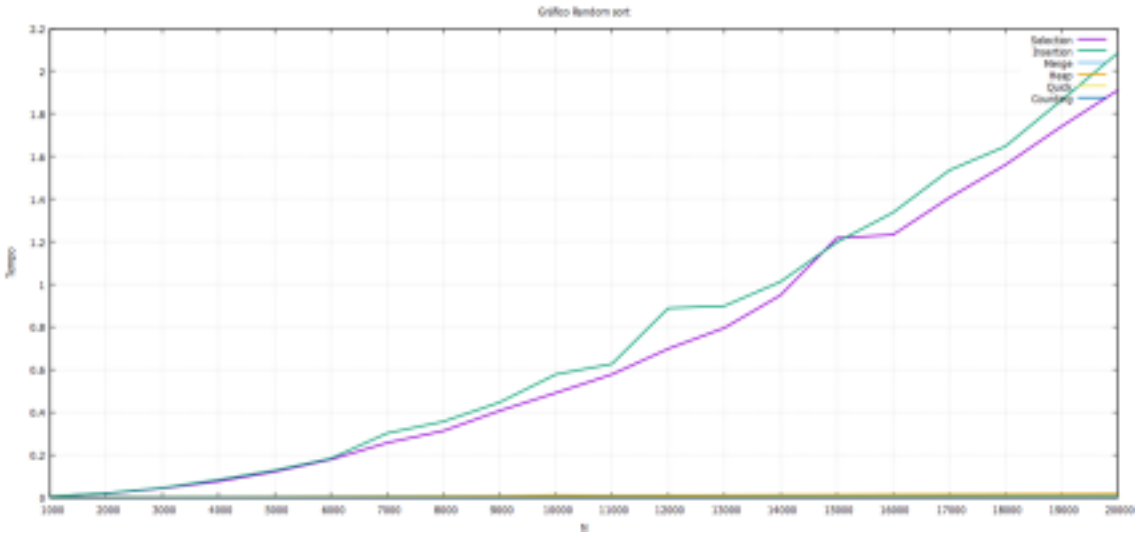
Iniciarei minhas conclusões sobre o trabalho realizado inicialmente destacando que, a linguagem de programação escolhida para a realização dessa atividade foi a linguagem python.

Os experimentos foram realizados em 4 vetores diferentes, um vetor aleatório, um vetor decrescente, um vetor crescente e um vetor semi-ordenado.

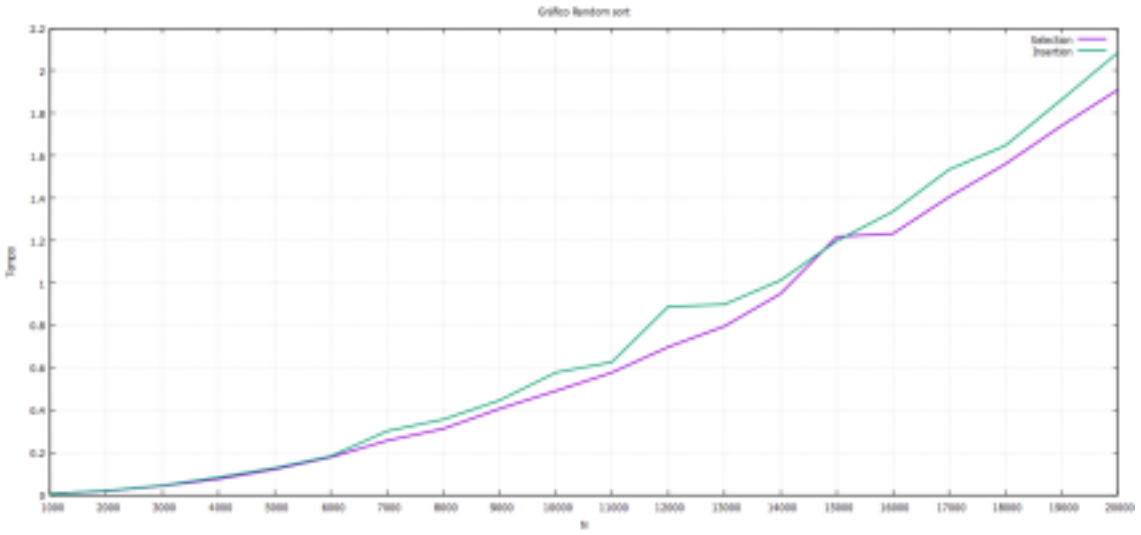
Observou-se no experimento que os algoritmos mais eficientes possuem tempo esperado de Θ (n log n), como o quick sort, merge sort e heap sort. Porém, o algoritmo mais eficiente dos analisados é o counting sort, pois é um algoritmo linear onde seu tempo esperado é de Θ (n)

Os algoritmos selection sort e insertion sort apresentam tempo de Θ (n²) no pior caso

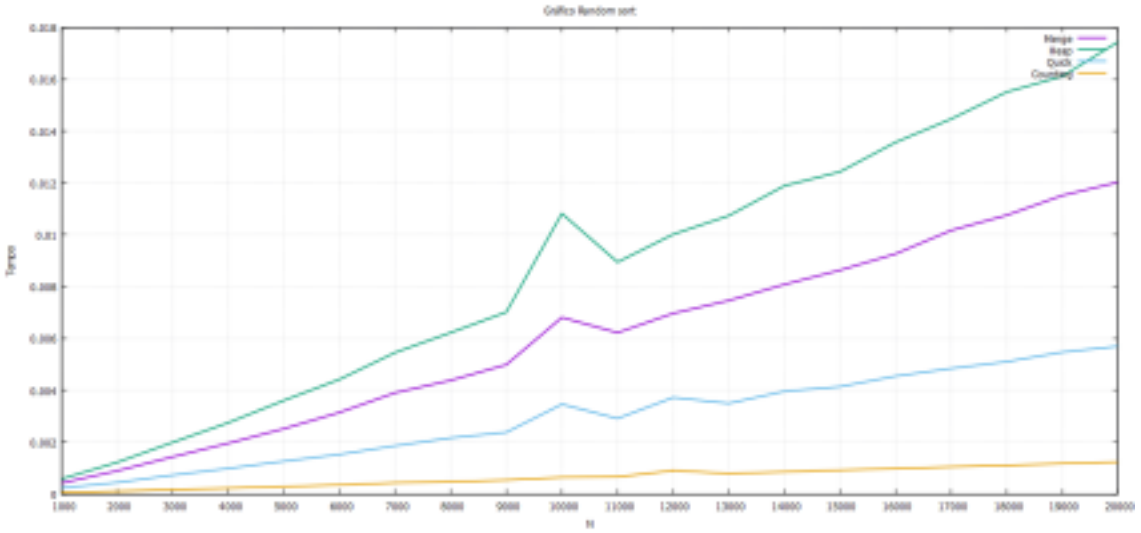
Iniciaremos falando sobre o vetor aleatório, apresentando gráficos com o tempo de execução dos algoritmos em relação ao número de entradas no vetor:



a) Todos os algoritmos para o vetor aleatório



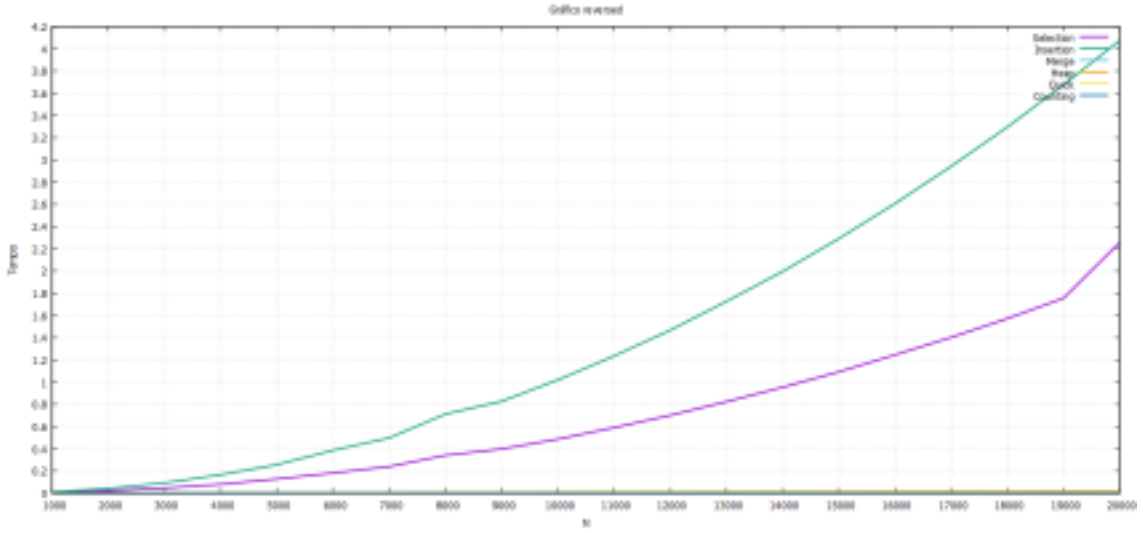
b) algoritmos elementares no vetor aleatório



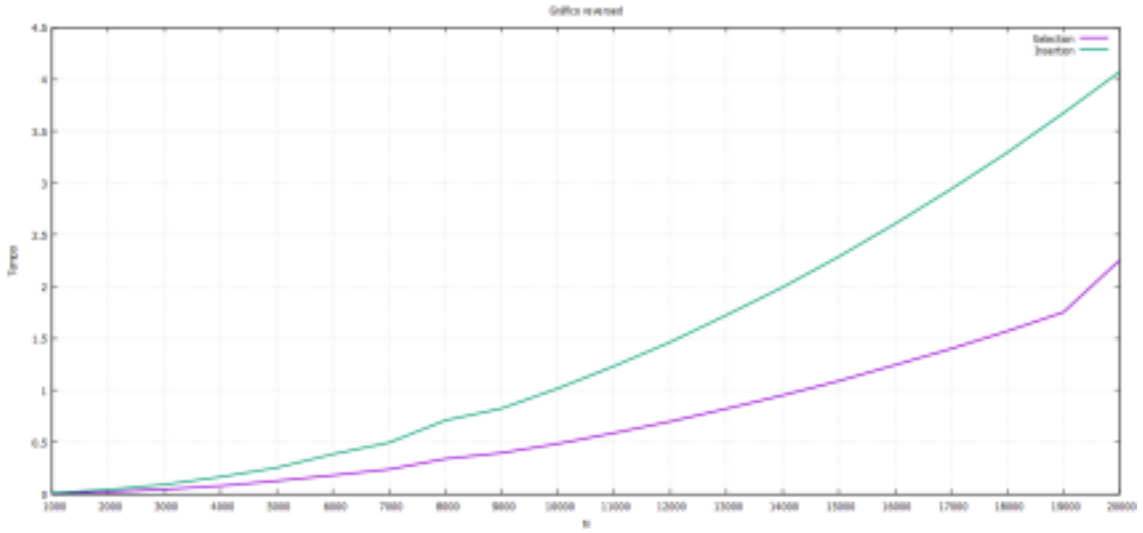
c) algoritmos eficientes no vetor aleatório

Observa-se que os algoritmos elementares, acabam tendo um desempenho inferior aos algoritmos eficientes. Dentre os algoritmos eficientes destaca-se o counting sort (linha amarela) por sua velocidade e eficiência superior aos demais. Também podemos destacar que, apesar de todos os algoritmos eficientes - com exceção do counting sort – possuírem a mesma complexidade, observa-se que há uma certa diferença de tempo entre eles. Há no gráfico alguns momentos em que ocorre certa variação, provavelmente devido ao fato de que o vetor a ser ordenado nesse período é um vetor que caiu mais próximo ao pior ou melhor caso do algoritmo

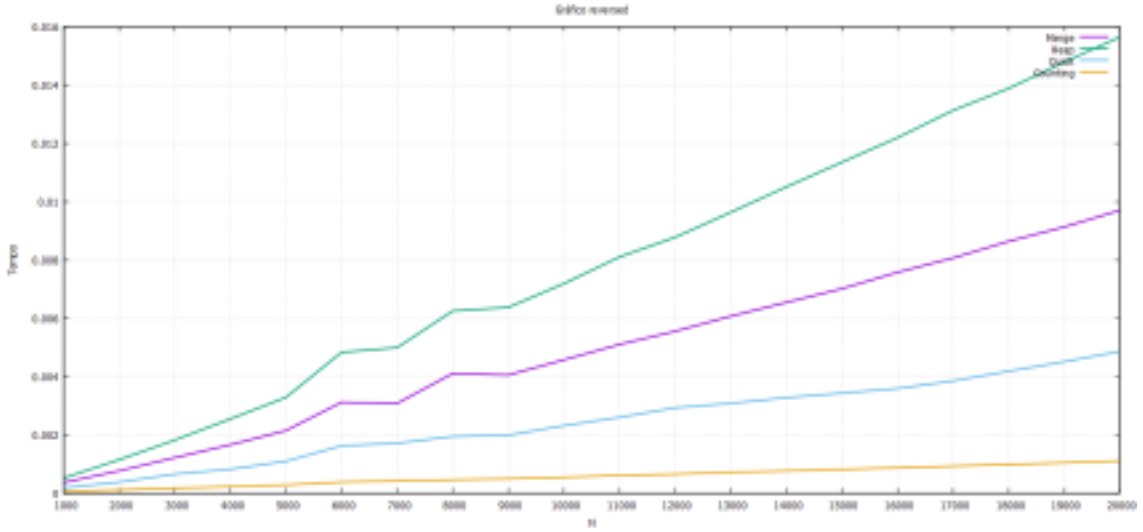
Vetor decrescente:



a) Todos os algoritmos de ordenação no vetor decrescente



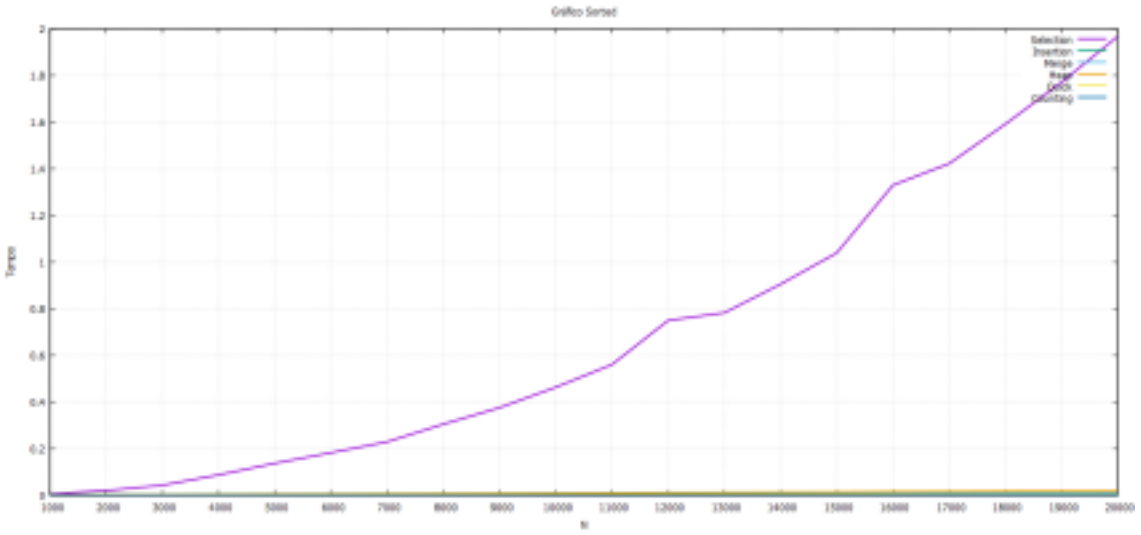
b) Algoritmos elementares no vetor decrescente



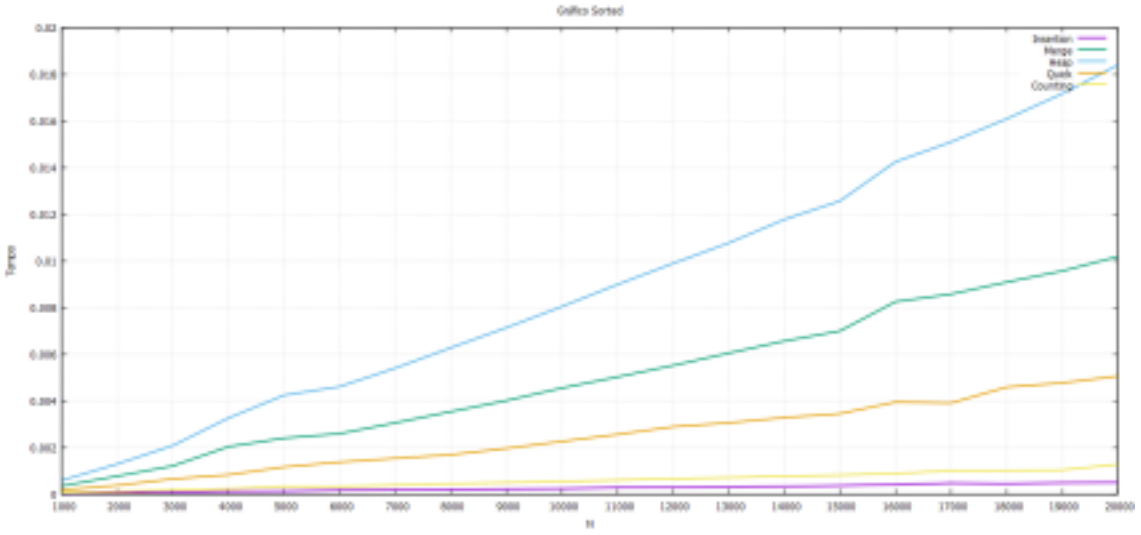
c) Algoritmos eficientes no vetor decrescente

No vetor decrescente observa-se os piores desempenhos dos algoritmos elementares, pois no vetor decrescente ocorre o pior caso para esses algoritmos. Observa-se novamente que com exceção do counting sort, todos os demais algoritmos eficientes possuem a mesma complexidade, mas há entre eles uma perceptível diferença de tempo, que se mantém em todos os vetores e casos de execução do trabalho

Vetor crescente:



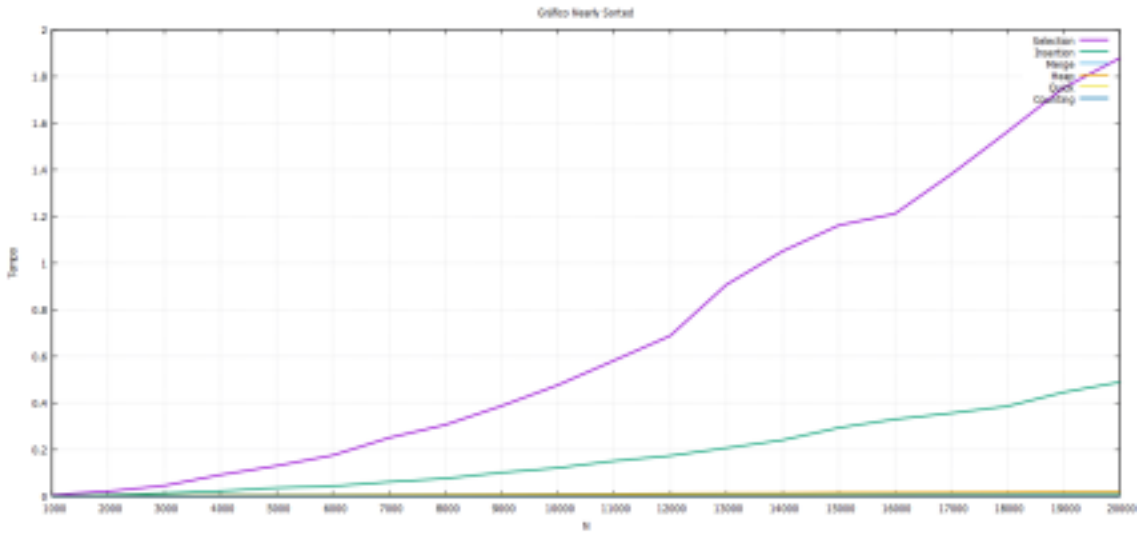
a) Todos os tempos de execução do vetor crescente



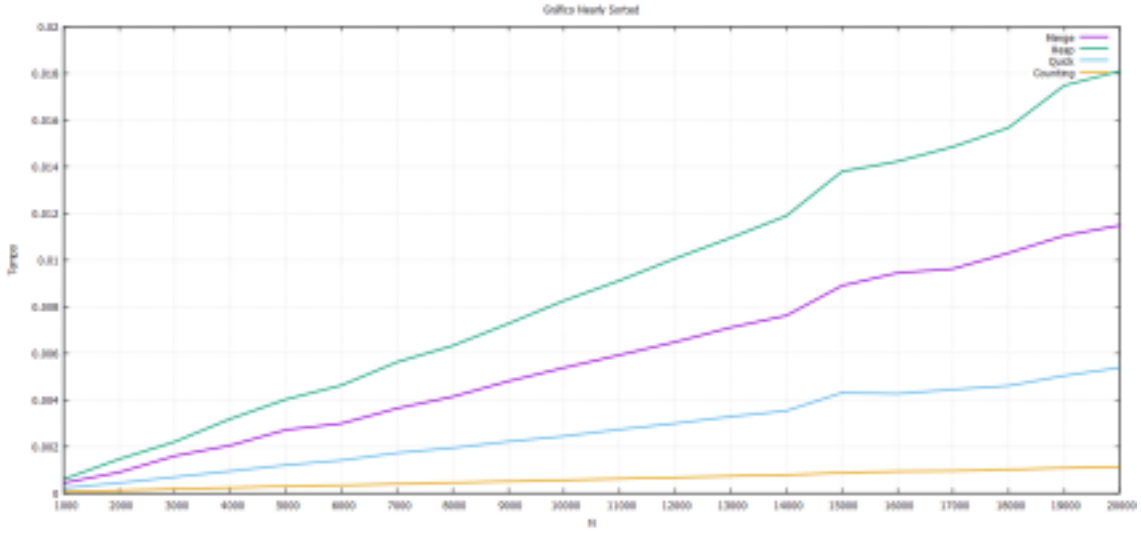
b) Algoritmos eficientes no vetor crescente. Observa-se a presença do algoritmo insertion sort aqui

Ao analisar o vetor crescente temos algumas diferenças. O insertion sort é o algoritmo mais eficiente, pois cai em seu melhor caso em que o mesmo se torna linear, ficando bem próximo ao counting sort.

Vetor semi-ordenado:



a) Todos os algoritmos no vetor semi-ordenado



b) Algoritmos eficientes no vetor semi-ordenado

Observamos que na figura “a”, há uma grande diferença entre o selection e o insertion sort, os algoritmos da figura “b” continuaram muito eficientes e mantiveram suas médias, com pouca variância onde caem em casos mais complexos para realizar ordenação

Conclusões: Os algoritmos selection e insertion sort de forma geral foram os mais lentos, porém destaca-se uma certa superioridade do insertion, principalmente nos vetores ordenados e semi-ordenados.

O algoritmo heap sort se mostra muito eficiente, mas como já esperado, entre os algoritmos de tempo (n log n), ele é o mais lento, perdendo para o merge e o

quick sort.

Os algoritmos heap sort e merge sort, possuem como limitante superior (seu pior caso) o tempo de execução O (n log n), sendo assim sempre muito eficientes e estáveis em todos os casos

O quick sort pode chegar em seu pior caso a um tempo de execução $\Theta(n^2)$, porém isso depende do pivô que está sendo selecionado. Esse caso geralmente ocorre quando o vetor já está ordenado. O caso médio e esperado do quick sort é de (n log n), e com os dados fornecidos e a execução do código, o algoritmo se manteve sempre no seu tempo médio, destacando a diferença que um bom pivô faz para tal algoritmo de ordenação.

O algoritmo counting sort é o mais eficiente de todos, onde em quase todos os casos foi superior a todos os demais algoritmos, perdendo apenas para o insertion em vetores que já estão ordenados. O counting sort possui seu tempo esperado em $\Theta(n)$.

Também se nota uma diferença de eficiência mesmo entre algoritmos de mesmo tempo de execução.