Análise da Comparação

Considerações:

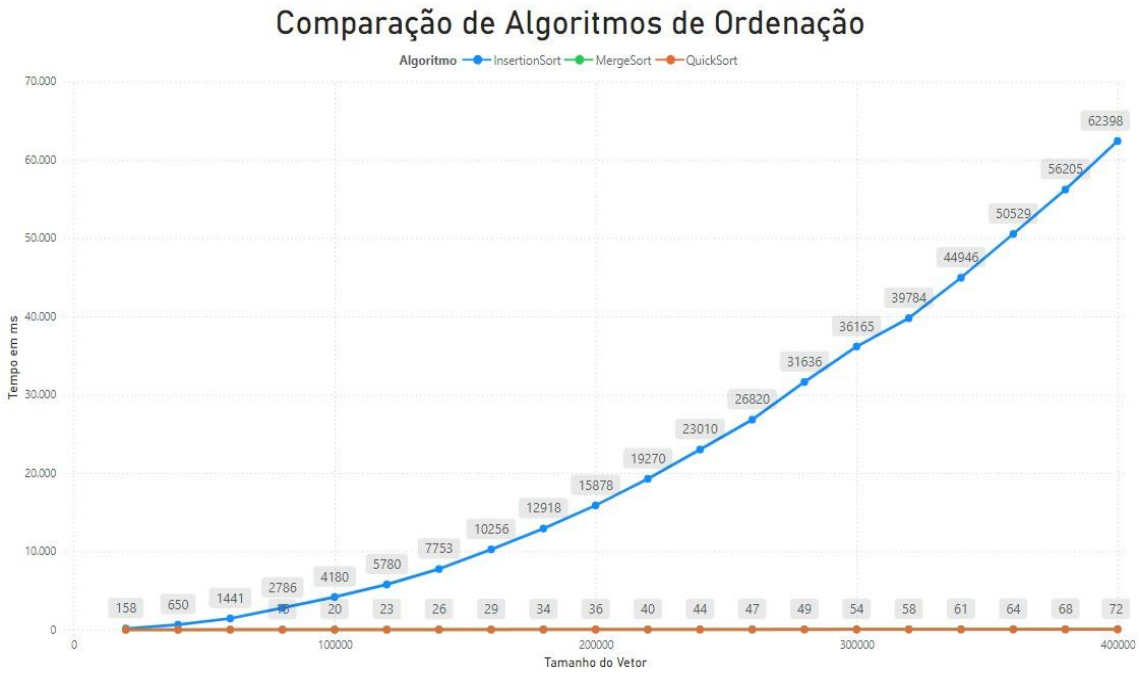
No presente estudo, foram utilizados três algoritmos famosos de ordenação (*InsertionSort*, *MergeSort* e *QuickSort*) implementados em C, para realizar uma comparação entre os tempos de execução na ordenação de vetores em cada um destes algoritmos. Como forma de equiparação, os três algoritmos rodaram para as mesmas magnitudes vetoriais de dimensão 1, com tamanho inicial de 20000 até 400000 num passo de 20000. Para o preenchimento do vetor foi utilizada a biblioteca <stdlib.h> para uso do método rand() que gera valores aleatórios entre 0 e 32767.

Algo importante a se ressaltar nesse último ponto é que vista a limitação do método na geração dos valores, o grau de aleatoriedade dos números gerados manteve-se o mesmo a partir do momento em que a magnitude do vetor ultrapassou o RAND_MAX (32767, como citado anteriormente), ou seja, já no segundo passo. Dessa forma a probabilidade da geração de números repetidos aumentou cada vez mais a partir dessa magnitude do vetor.

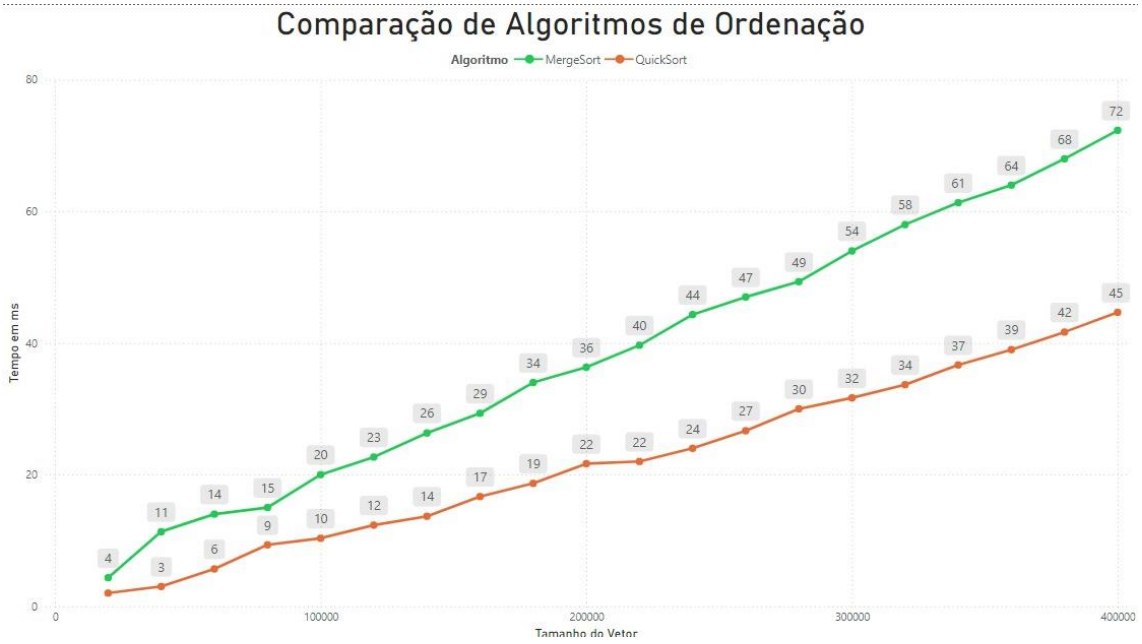
Para registro desses valores e posterior análise exploratória dos dados, foi utilizada a classe FILE advinda da biblioteca <stdio.h> para definir parâmetros de entrada em um arquivo de texto .csv de forma que os valores do tempo de execução atribuídos a cada execução do algoritmo de ordenação fossem salvos junto da respectiva magnitude do vetor no passo do loop e o nome de seu algoritmo. De tal maneira, foi possível utilizar essa base de dados para criar um dashboard com o auxílio da ferramenta PowerBI Desktop, disponibilizada gratuitamente pela Microsoft.

O registro do tempo de execução a cada chamada do respectivo algoritmo de ordenação foi feito por meio da biblioteca <time.h> através do método clock() que inicia uma contagem em milissegundos a partir do momento que é executado. A fim de que houvesse uma confiabilidade maior no tempo de execução da ordenação para cada algoritmo foi feita uma média do tempo de execução de 3 ordenações para cada algoritmo de acordo com cada faixa de magnitude do vetor.

O Dashboard com filtros e análise individual tempo a tempo dos algoritmos encontra-se no arquivo *Dashboard Comparação.pbix* localizado neste repositório.



Na imagem acima, é possível ver a comparação entre os tempos de execução dos três algoritmos (*InsertionSort*, *MergeSort* e *QuickSort*) marcados pelas magnitudes vetoriais que variaram de 20000 ao passo dos mesmos 20000 até 400000 de tamanho. Como pode-se observar, o algoritmo de Inserção destoa como o mais custoso em termos de período em execução. Devido a essa distorção, os algoritmos de *Merge* e *Quick* acabam se sobrepondo de maneira a impossibilitar a percepção de suas diferenças. Essa grande diferença entre o primeiro e os dois últimos se dá pelo fato do tempo de execução do algoritmo de Inserção ser proporcional ao tamanho do vetor. Dessa forma o tempo máximo da execução a depender do nível de ordenação do mesmo pode ser no pior dos casos n^2 (sendo “n” a quantidade de números do vetor).



Já nessa imagem, é possível ver como se dá a diferença entre os algoritmos *Merge* e *Quick.* Embora a diferença para o algoritmo de Inserção seja gigante, a diferença ponto a ponto destes dois últimos também se mostra significativa a medida que o tamanho do vetor aumenta, sendo em grandes parte dos recortes uma diferença de 2:1.

Apesar de ambos algoritmos possuírem premissas parecidas de dividir o vetor em partes de forma que possa ser feita essa ordenação, o fato de o algoritmo de *Merge* utilizar de alocação de memória acaba pesando mais em termos de processamento. Aliado a esse fator, o *Quick* tem duas vantagens interessantes:

1. Na definição do pivô (valor do vetor que vai conter o número que dividirá o vetor – inicialmente – em duas partes: de um lado maiores que ele, do outro menores ou iguais, por exemplo), esse valor não será mais alterado de posição a partir do momento em que os subvetores forem definidos, o que já é um gasto a menos;
2. Quanto mais organizado o vetor, ou seja, quanto mais próximo da ordenação um vetor estiver maior será o tempo de execução. Isso pode parecer ruim, mas é uma condição que na prática em aplicações reais tem probabilidade ínfima de acontecer;

A escolha do pivô pode impactar e muito no tempo de execução do *QuickSort*. Caso o pivô seja um dos maiores ou um dos menores valores do vetor, a divisão em subvetores pode ser extremamente desigual o que aproximará o tempo de execução em n^2. Como já dito, caso extremamente raro, pois na maioria dos casos, o tempo de execução do *Quick* se aproximará de $n \log n$ na base 2.