**Tecnologia em Sistemas para Internet**

**Estrutura de Dados – ARQESDD**

**Prof. Ednilson Geraldo Rossi**

**Relatório Técnico**

**Testes de Métodos de Classificação**

**Lucas Eduardo Parila**

**Matheus Correia**

Instituto Federal – Campus Araraquara

**1. INTRODUÇÃO**

Neste estudo iremos abordar diferentes algoritmos de ordenação para testá-los em situações diversas e analisar os resultados de cada um deles, comparando assim a eficiência dos métodos de acordo com o tempo de execução e o número de comparações e de trocas realizadas.

Serão adotados para os testes vetores de números inteiros, a fim de facilitar a análise dos resultados e seu entendimento, testado com três quantidades de elementos (um mil, cem mil e um milhão) e com três disposições diferentes (vetor ordenado crescente, vetor ordenado decrescente e vetor aleatório).

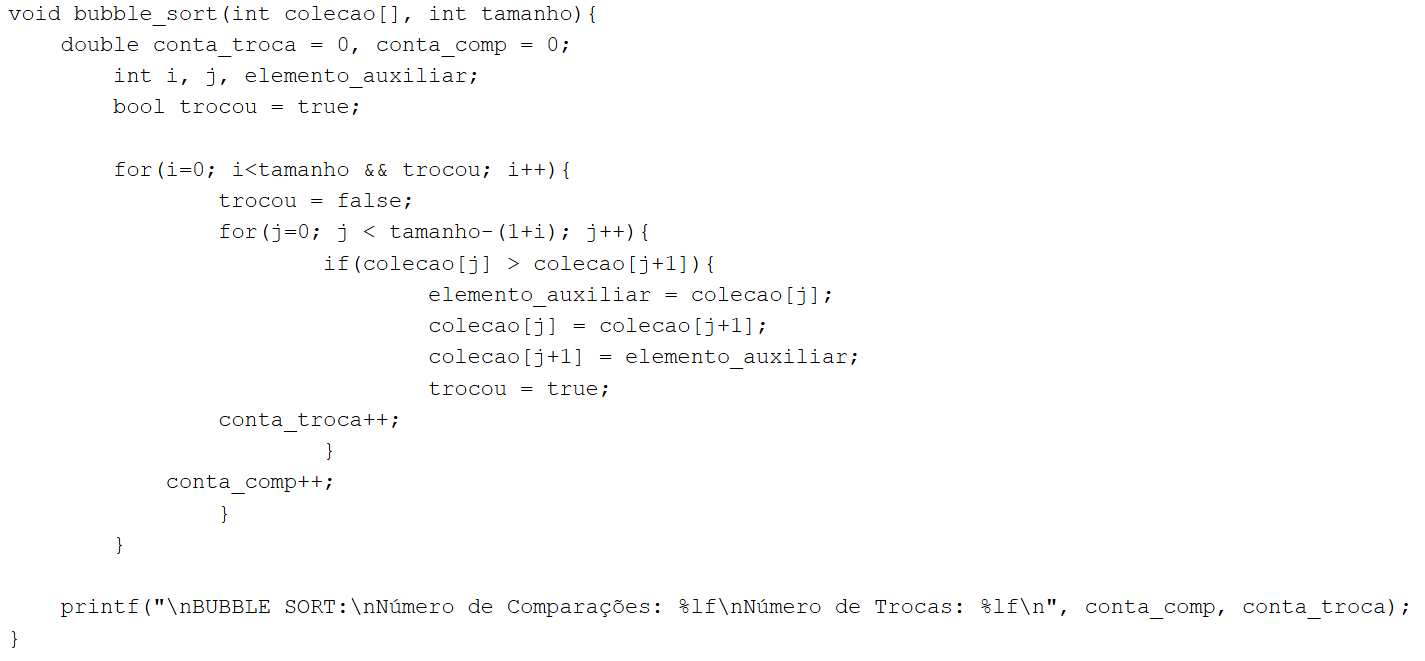
**2. MÉTODOS UTILIZADOS**

Para ordenar os vetores em cada situação citada, serão utilizados os seguintes métodos de ordenação:

**2.1 Bubble Sort**

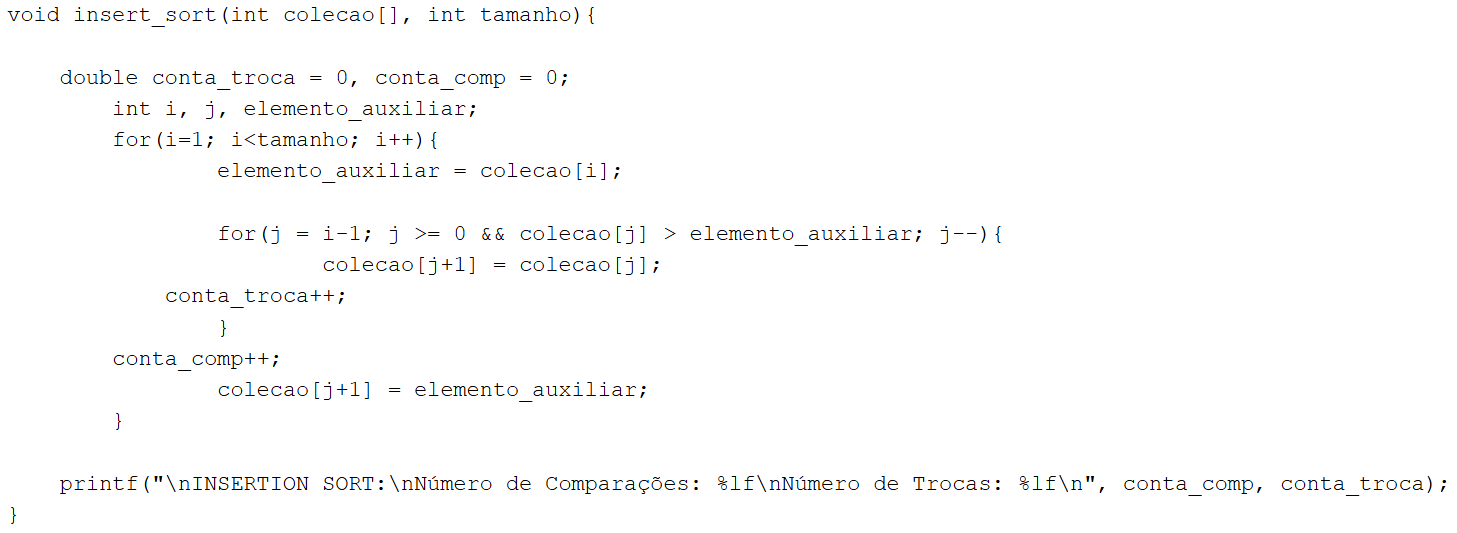
Funciona comparando pares de elementos vizinhos e realizando as trocas quando necessárias. Assumindo que queremos o vetor em ordem crescente, sempre na primeira iteração do algoritmo com o vetor, o maior elemento ficará na última posição. Assim se repete o processo até a total ordenação.

Uma melhoria que vale ser feita, é implementar uma condição de parada caso não seja realizada nenhuma troca em uma iteração, apontando que o vetor já está ordenado.



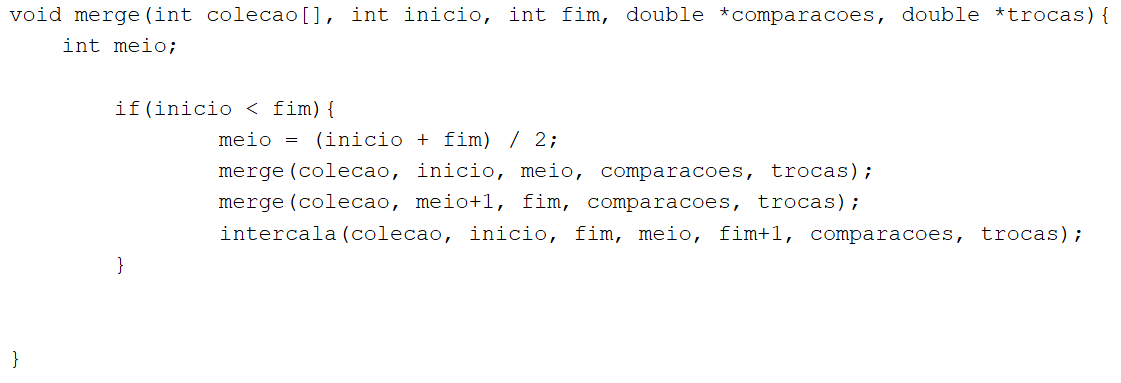
**2.2 Inserction Sort**

Funciona inserindo cada elemento em sua posição correta, criando um “sub-arquivo” já ordenado nas duas primeiras posições. Após, cada elemento do vetor é comparado com os elementos do “sub-arquivo” e é inserido na devida posição, utilizando uma variável auxiliar.



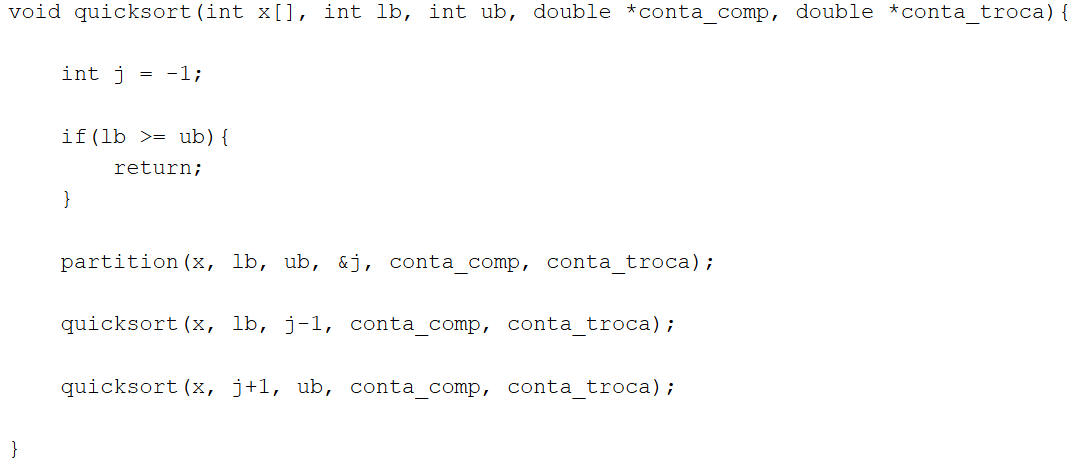
**2.3 Merge Sort**

Este método usa do conceito de Dividir para Conquistar; dividindo o problema em subproblemas menores, de mais fácil resolução. O algoritmo divide o vetor pela metade, recursivamente, até que cada elemento fique sozinho. Após, os elementos são comparados e reagrupados novamente, intercalando-os nas posições corretas, até que o vetor volte ao seu tamanho original totalmente ordenado.



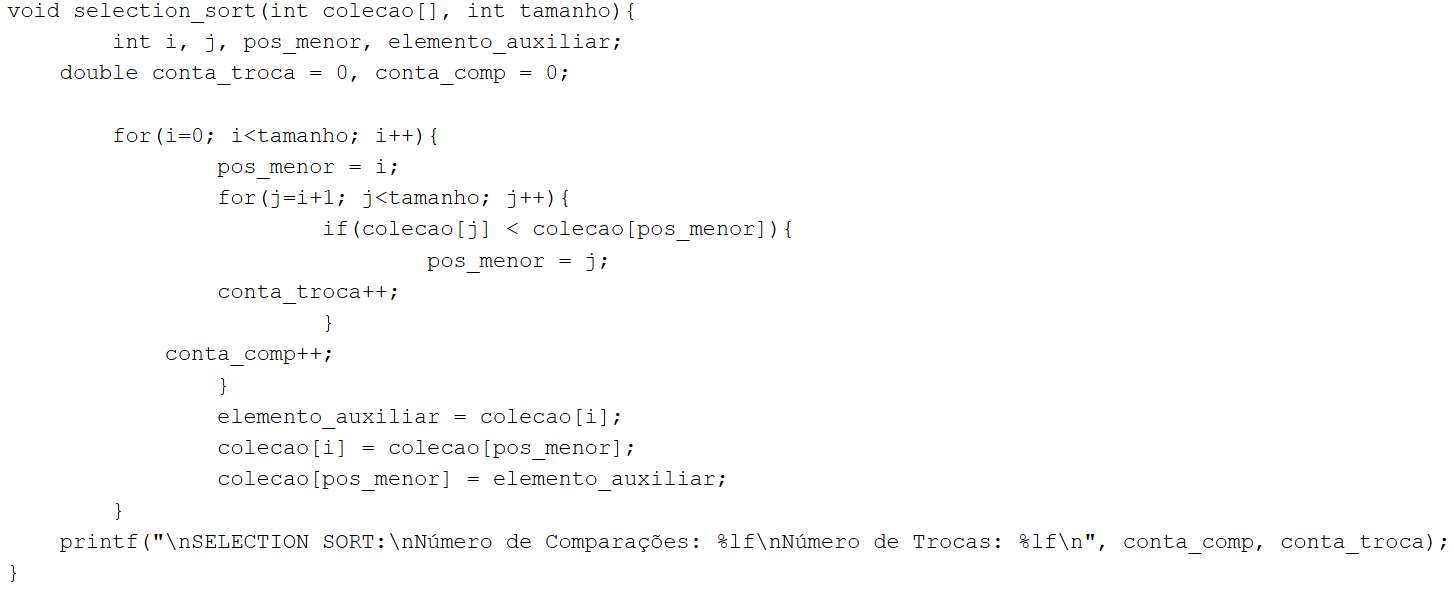
**2.4 Quick Sort**

Este também utiliza do conceito de Dividir para Conquistar. Primeiramente um elemento do vetor é escolhido como “pivô”. A escolha de um pivô ruim pode levar a utilização deste método a um desempenho ruim. Após a escolha do elemento, são criados dois “sub-arquivos”, um com os elementos menores que o pivô e outro com os maiores. Esse processo se repete recursivamente até que restem “sub-arquivos” com apenas um elemento, onde estes serão reagrupados já na ordem correta.



**2.5 Selection Sort**

Funciona selecionando o menor elemento do vetor (quando se quiser um vetor em ordem crescente) e permutando este com o elemento da primeira posição. Após a primeira troca, o primeiro elemento, que está na posição correta, é ignorado e o processo se repete até a ordenação total.



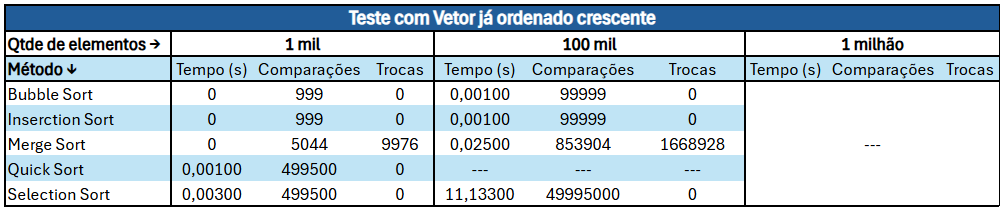
**3. ANÁLISE DE RESULTADOS**

O programa foi desenvolvido em linguagem C no Visual Studio Code. As configurações básicas da máquina na qual o programa foi executado para os testes são as seguintes:

* Processador: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz.
* RAM instalada: 8,00 GB (utilizável: 7,87 GB).
* Tipo de sistema: Sistema operacional de 64 bits, processador baseado em x64.
* Sistema operacional: Windows 11 Pro.

**3.1 Teste com Vetor Ordenado (em ordem crescente)**

A tabela a seguir demonstra os resultados obtidos após os testes com vetores ordenados, com diferentes quantidades de elementos:



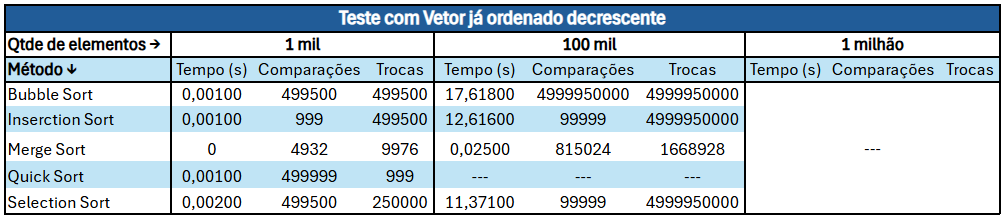
Podemos observar que os métodos Bubble Sort e Inserction Sort apresentaram o melhor desempenho com os vetores ordenados, pois fazem apenas uma iteração com o vetor, e, claro, não realizam nenhuma troca. **Importante lembrar que o Bubble Sort utilizado possui a melhoria citada no item 2.1.**

Já o método menos eficiente neste teste foi o Merge Sort, que, além de realizar um número maior de comparações, realizou diversas trocas, mesmo com o vetor já na ordem correta.

O método Quick Sort rodou apenas no teste com 1 mil elementos.

**3.2 Teste com Vetor Invertido (em ordem decrescente)**

A seguir, temos a tabela que mostra os resultados obtidos após os testes com vetores invertidos, com diferentes quantidades de elementos:

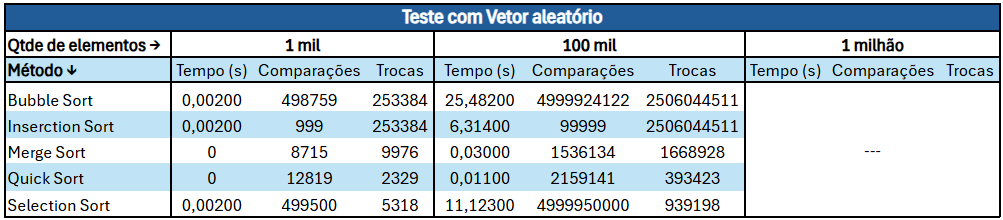


Com os resultados em mãos, vemos que o método com melhor desempenho no teste foi o Merge Sort, considerando comparações e trocas realizadas.

O pior método do caso foi o Bubble Sort, o qual troca os elementos em todas as comparações feitas, além do tempo elevado de execução.

**3.3 Teste com Vetor Aleatório**

Na última variação do teste, a tabela apresenta os resultados dos testes com vetores gerados aleatoriamente, nas diferentes quantidades de elementos:



Em nosso teste com vetores aleatórios, o Inserction Sort apresenta o menor número de comparações realizadas, fazendo apenas uma iteração com o vetor, porém, apresentou o maior número de trocas, juntamente com o Bubble Sort. O método com resultados mais satisfatórios foi o Quick Sort, que realizou o menor número de trocas, no menor tempo entre os algoritmos.

**4. CONCLUSÃO**

Com os testes, concluímos que o método Bubble Sort não apresenta resultados satisfatórios em nenhuma situação, desconsiderando a melhoria citada no item 2.1, devendo ser utilizado apenas para demonstração de lógica em ambientes educacionais.

O Inserction Sort e o Selection Sort são boas opções para vetores pequenos e quase ordenados, devendo ser descartados para maiores problemas.

O Quick Sort é o método mais rápido de classificação, mas depende de uma boa escolha no pivô, tornando-o instável em seus resultados. Mas o alto número de comparações pode tornar sua implementação mais cara. É uma ótima opção quando o tempo de execução é essencial.

O método mais estável é o Merge Sort, apresentando resultados parecidos em todos os casos testados. Mas sua recursividade pode fazê-lo apresentar um tempo maior de execução.

Infelizmente, a máquina utilizada não apresentou resultados com vetores de 1 milhão de elementos, não apontando nenhum erro de compilação ou execução. O método Quick Sort também não apresentou resultados além dos vetores de 1 mil elementos, exceto no teste com vetores aleatórios.