INSTITUTO FEDERAL São Paulo Campus Araraquara	**Estrutura de Dados**
Métodos de Classificação – Relatório Final	
Prof.: Ednilson Rossi	
Alunos: Natan Mendes Araújo e Yara Leal Matheus	

1. Estrutura do Trabalho

O presente trabalho foi realizado com base nos métodos de classificação Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort, ensinados de acordo com o cronograma da disciplina.

A estrutura do programa consiste em três arquivos do tipo c (.c), mais um arquivo do tipo biblioteca (.h) no qual estão presentes os códigos dos métodos de classificação, das funcionalidades de imprimir o vetor, gerar vetor aleatório, em ordem crescente e decrescente e devolver os valores das comparações e trocas. Há ainda um arquivo tipo c destinado a teste.

Os três arquivos principais são referentes à quantidade de elementos do vetor trabalhado, sendo o primeiro denominado “mil.c”, o segundo “cem_mil.c” e o terceiro “um_milhao.c. Cada um desses foi dividido em blocos referentes aos métodos de classificação utilizados. Primeiro a função do tipo de ordenação é chamada, depois a função de imprimir o vetor, seguida da função de classificação e novamente a função de impressão, fazendo com que cada bloco tenha mais duas subdivisões: dos vetores em ordem solicitada nesta avaliação (crescente, aleatória e decrescente) e dos vetores ordenados.

O tempo gasto por cada tipo de classificação foi calculado através da função “clock()”. Foi feito um *struct* contendo cinco elementos que irão receber o cálculo do tempo gasto por cada método. A função *clock* foi iniciada antes da escrita da função do método e finalizada depois. Após isso, cada elemento da *struct* recebeu a diferença entre o fim e início que foi convertida em segundos. Foi criada também uma variável auxiliar para cada método que soma o tempo individual de cada bloco e exibe o total no final do código.

As trocas e comparações foram calculadas utilizando uma variável de contagem que foi inserida no meio das funções do Bubble Sort, Selection Sort e Insertion Sort. Nos casos do Merge Sort e Quick Sort foram utilizados ponteiros para as variáveis contadoras.

As configurações de uma das máquinas utilizadas são:

- Fabricante: LENOVO;
- Modelo: 20X2006LBO;
- Processador: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2419 Mhz, 4 Núcleo(s), 8 Processador(es) Lógico(s)
- Memória RAM: 8,00 GB;
- Sistema Operacional: Microsoft Windows 11 Home Single Language

Constatou-se que em nenhuma das máquinas utilizadas, o código com o vetor de um milhão de elementos funcionou. Alguns compiladores online foram utilizados (GDB e Replit) para essa finalidade, mas também não chegaram ao resultado.

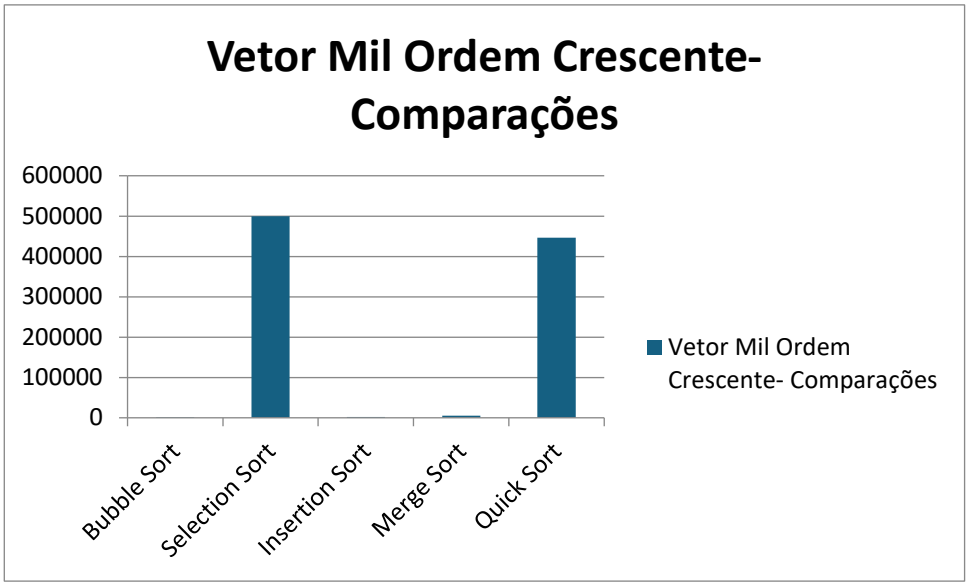
Os resultados dos outros dois vetores serão demonstrados a seguir

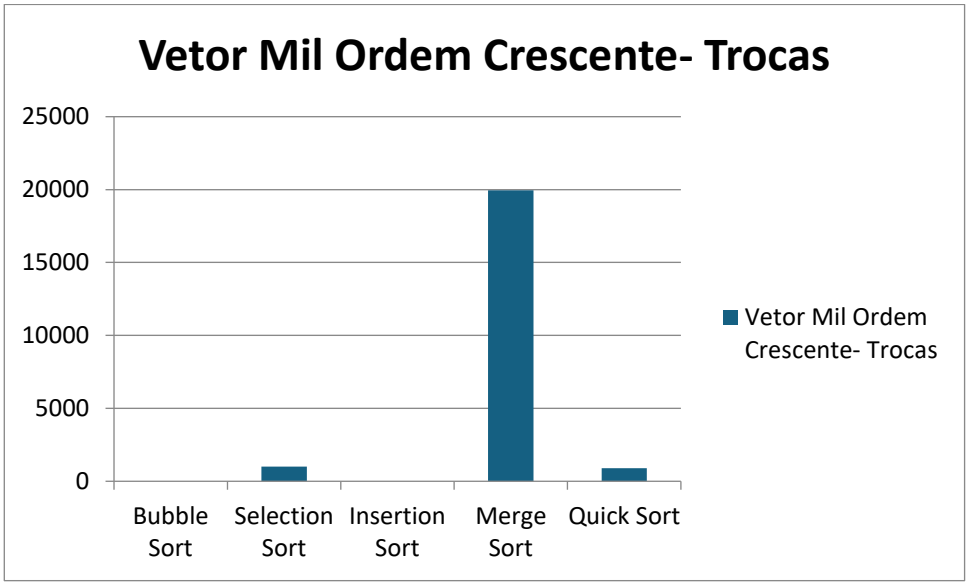
2. Testes e Resultados

2.1 Vetor Com Mil Elementos

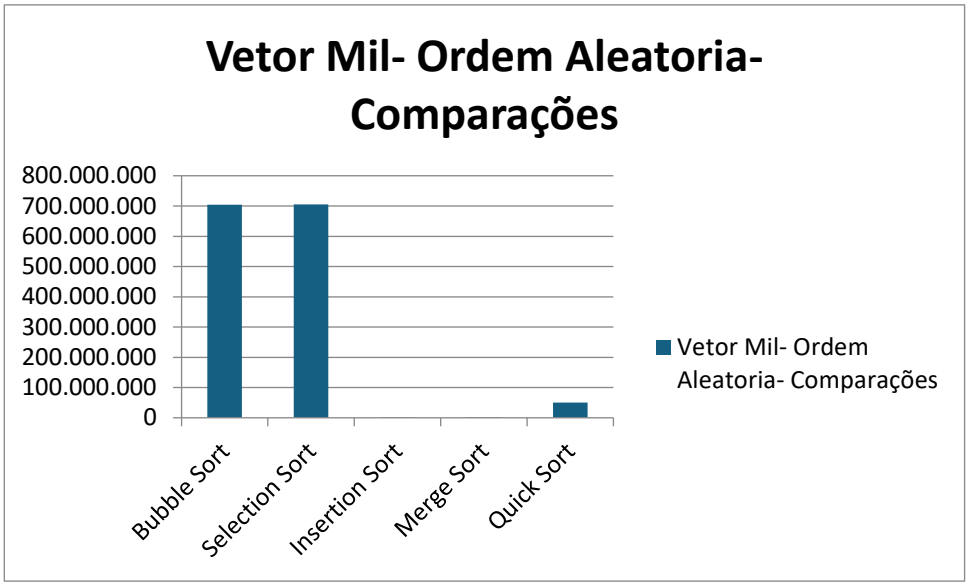
Vetor [1000] - Mil									
Lista	**Ordem Crescente**			**Ordem Aleatoria**			**Ordem Decrescente**		
Algoritmos	**Tempo(s)**	**Comp**	**Trocas**	**Tempo(s)**	**Comp**	**Trocas**	**Tempo(s)**	**Comp**	**Trocas**
Bubble Sort	0.000000	999	0	0.000000	499.380	247.827	0.000000	499.500	499.367
Selection Sort	0.000000	499.500	1000	0.000000	499.500	1000	0.000000	499.500	1000
Insertion Sort	0.000000	1998	0	0.000000	1998	243.288	0.000000	1998	499.364
Merge Sort	0.000000	5044	19.952	0.000000	8721	19.952	0.000000	4997	19.952
Quick Sort	0.000000	446.282	899	0.000000	12.956	2390	0.000000	440.949	961

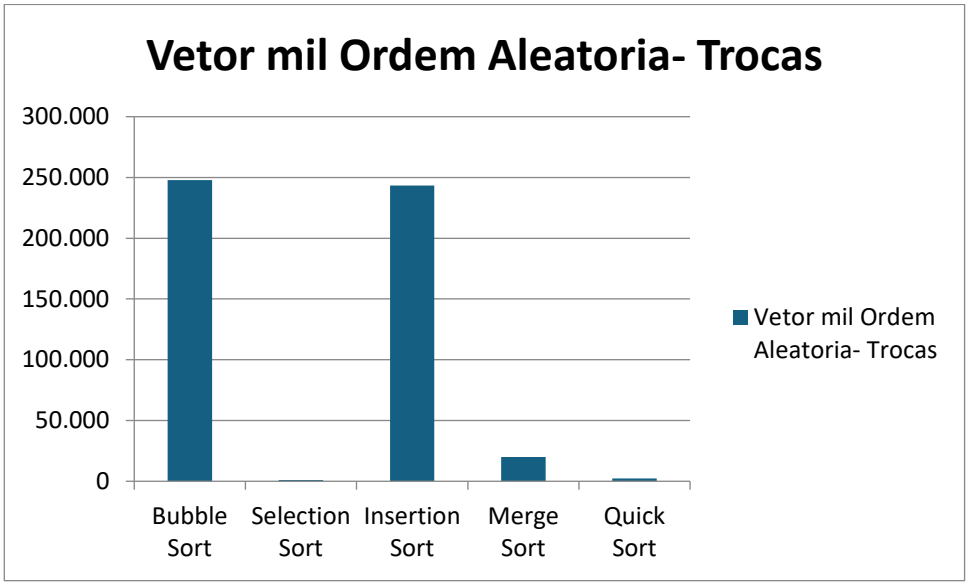
No primeiro vetor contendo mil elementos é possível observar que a máquina não demonstrou dificuldade em calcular todas as operações já que o tempo de todas não passou de zero segundos. Quando esse vetor é colocado em ordem crescente já ordenada, o método mais eficiente foi o **Bubble Sort** com menor número de comparações e zero trocas e o pior caso ocorreu com o método **Selection Sort** como demonstrado nos gráficos abaixo. O método **Merge Sort** também resultou em grande número de trocas.



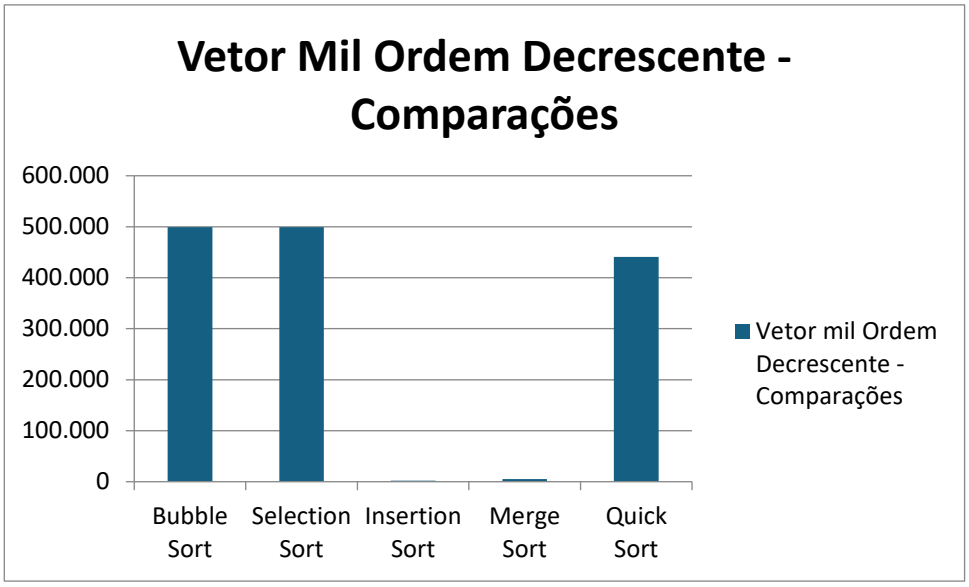


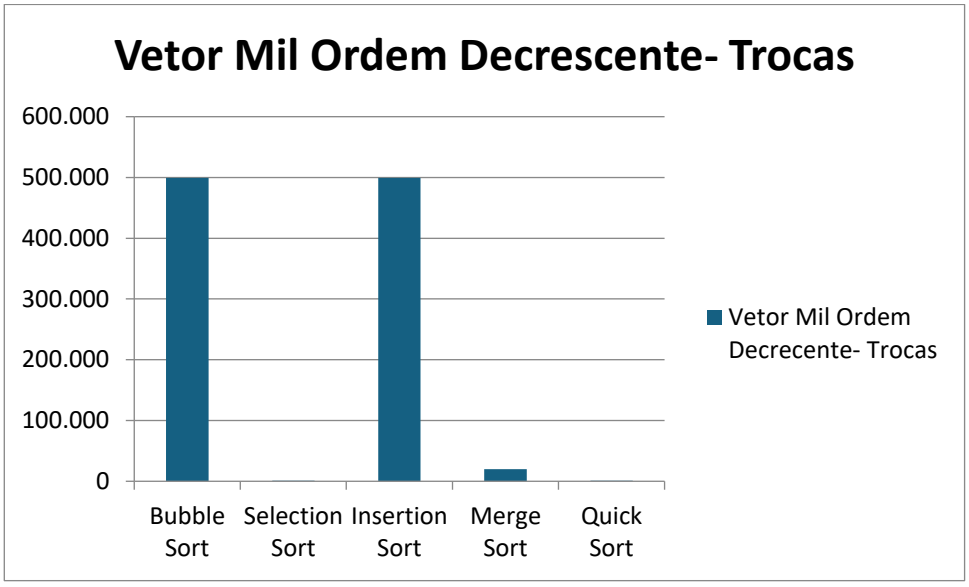
No caso do vetor organizado de forma aleatória, o método **Quick Sort** foi o de melhor eficiência, seguido pelo **Merge Sort**. Já o pior foi o **Bubble Sort** e por mais que o total de comparações e trocas tenha sido maior que o **Selection Sort**, esse contou com número elevado de comparações (maior do que o Bubble)**.**





E no caso de ordem decrescente o melhor método foi o **Merge Sort,** com resultados bem menores que os outros métodos, e o pior foi o **Bubble Sort**.


Vetor Mil Ordem Decrescente - Comparações
600.000
500.000
400.000
300.000
200.000
100.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor mil Ordem Decrescente - Comparações


Vetor Mil Ordem Decrescente- Trocas
600.000
500.000
400.000
300.000
200.000
100.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor Mil Ordem Decrecente- Trocas

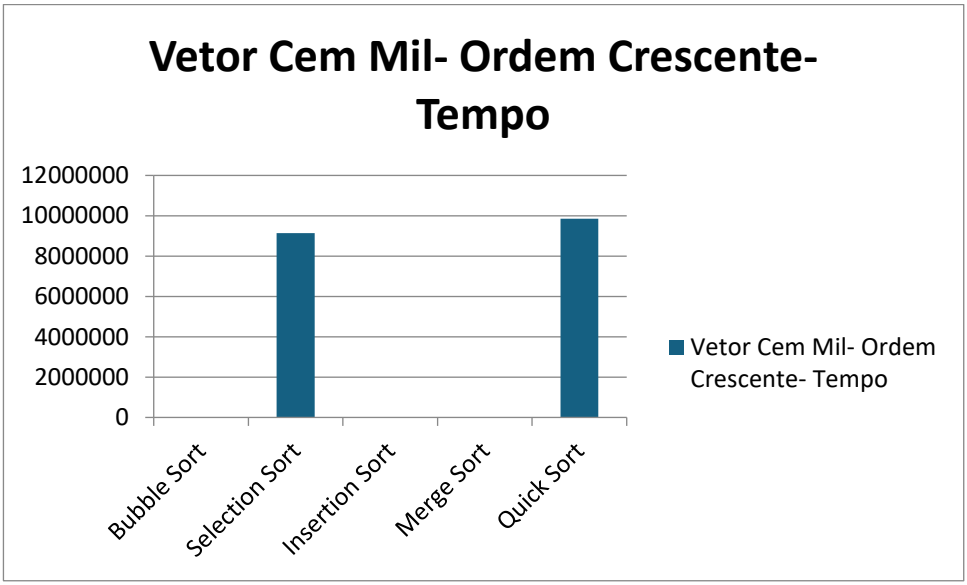
2.2 Vetor Com Cem Mil Elementos

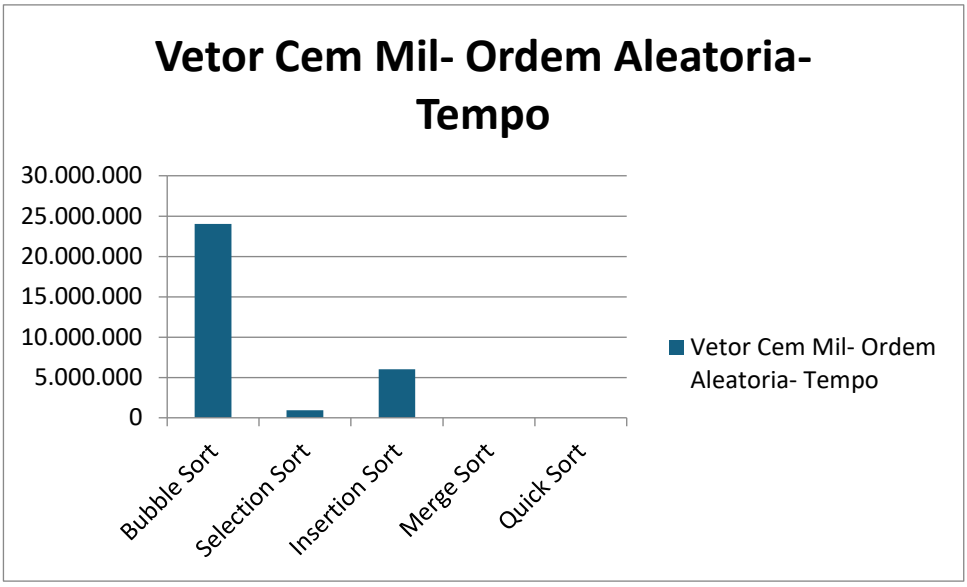
Vetor [100000] - Cem Mil									
Lista	**Ordem Crescente**			**Ordem Aleatória**			**Ordem Decrescente**		
Algoritmos	**Tempo(s)**	**Comp.**	**Trocas**	**Tempo(s)**	**Comp.**	**Trocas**	**Tempo(s)**	**Comp.**	**Trocas**
Bubble Sort	0.000000	99.999	0	24.011.000	704.396.801	1.822.065.030	16.646.000	704.980.704	704.970.336
Selection Sort	9.141.000	704.982.704	100.000	916.900	704.982.704	100.000	11.470.000	704.982.704	100.000
Insertion Sort	0.000000	199.998	0	6.024.000	199.998	1.823.525.897	12.217.000	199.998	704.970.234
Merge Sort	0.031000	853.904	3.337.856	0.047.000	1.533.403	3.337.856	0.047.000	822.676	3.337.856
Quick Sort	9.847.616	147.013.572	90.093	0.123215	50.804.724	257.104	10.300.757	182.136.717	96.199

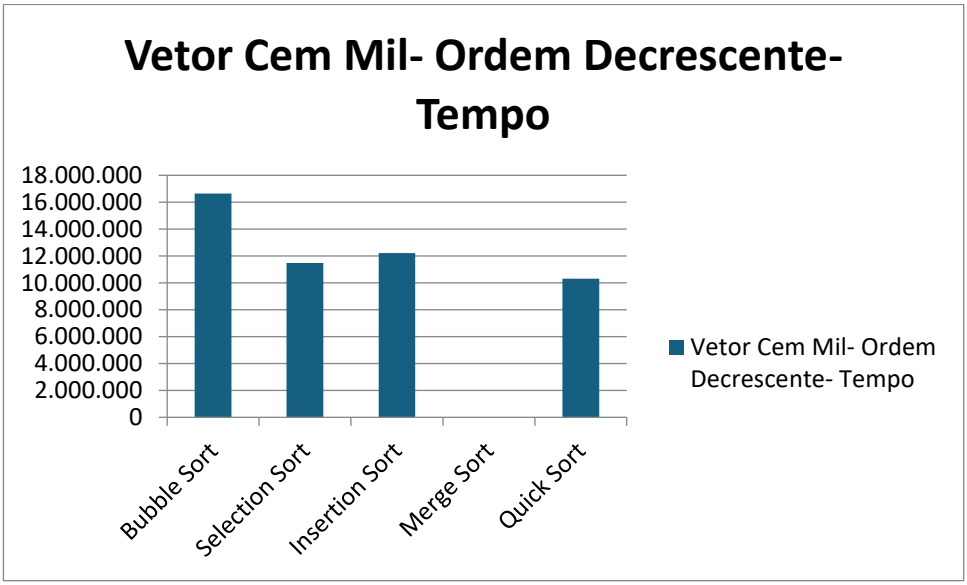
No segundo vetor foi possível traçar as diferenças de tempo de cada organização, sendo que na ordem crescente os métodos **Bubble, Merge** e **Insertion Sort** foram calculados mais rapidamente e o que tomou mais tempo foi o **Quick Sort**.

Na ordem aleatória o **Merge** e o **Quick Sort** foram os mais rápidos e o **Bubble Sort** o mais lento.

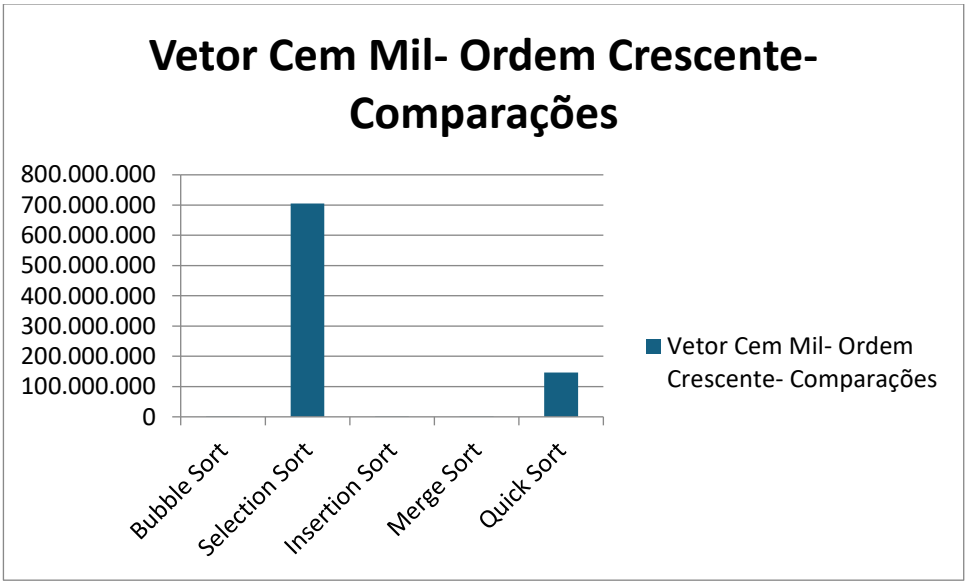
E na ordem decrescente O **Merge Sort** tomou menos tempo e o **Bubble Sort** tomou mais tempo.

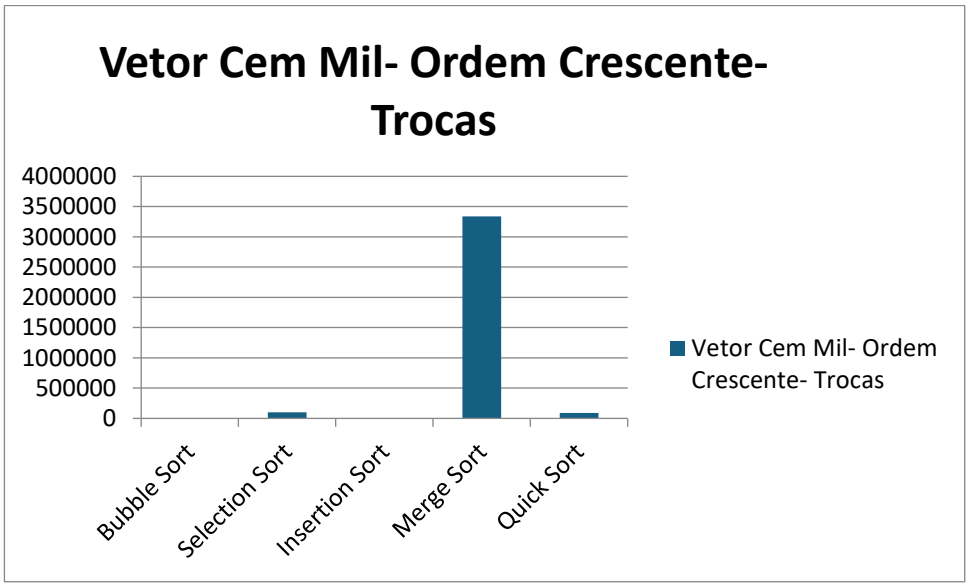



Vetor Cem Mil- Ordem Aleatoria- Tempo
30.000.000
25.000.000
20.000.000
15.000.000
10.000.000
5.000.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor Cem Mil- Ordem Aleatoria- Tempo

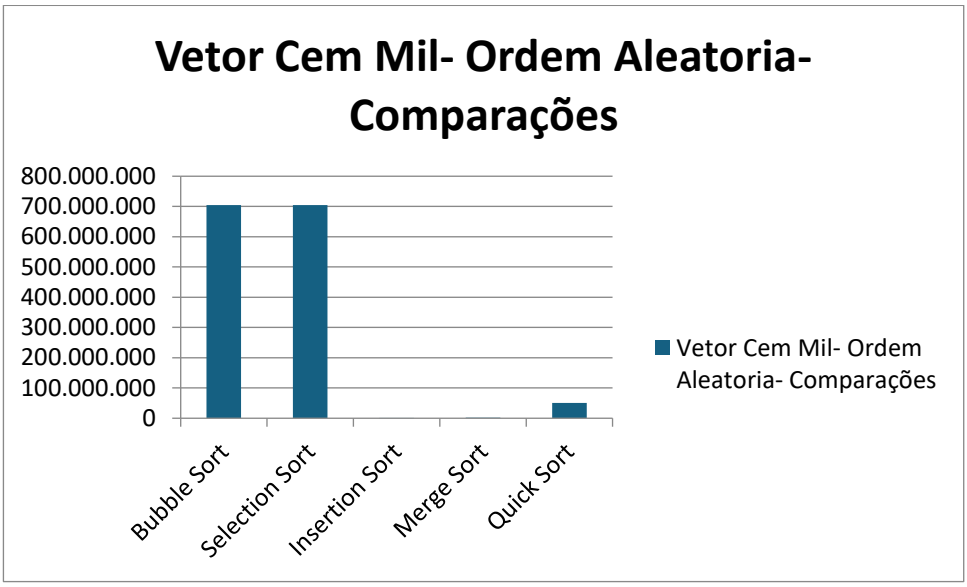

Vetor Cem Mil- Ordem Decrescente- Tempo
18.000.000
16.000.000
14.000.000
12.000.000
10.000.000
8.000.000
6.000.000
4.000.000
2.000.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor Cem Mil- Ordem Decrescente- Tempo

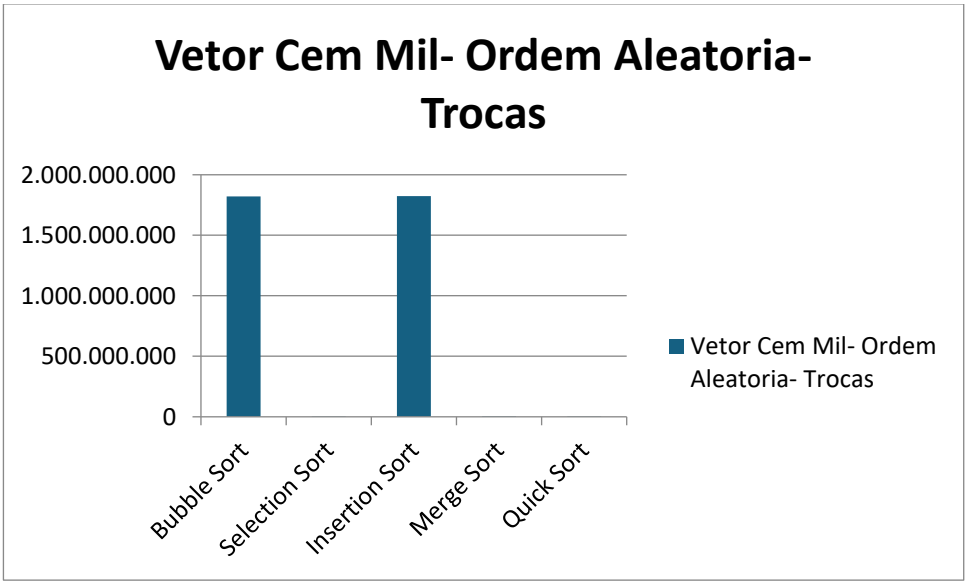
Sobre os melhores e piores casos, constatou-se que na ordem crescente e ordenada o método mais eficiente foi o **Bubble Sort** e o de menor eficiência foi o **Selection Sort.**

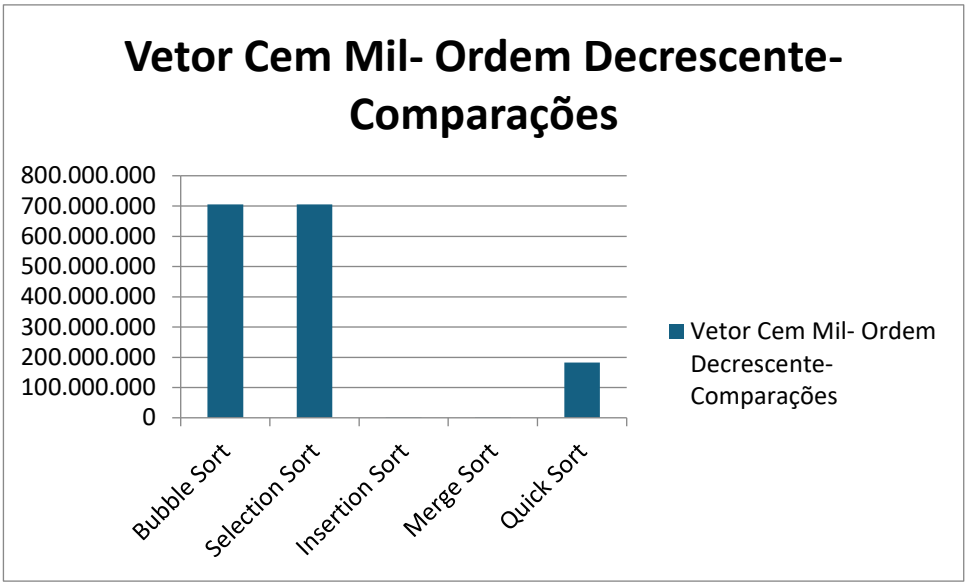


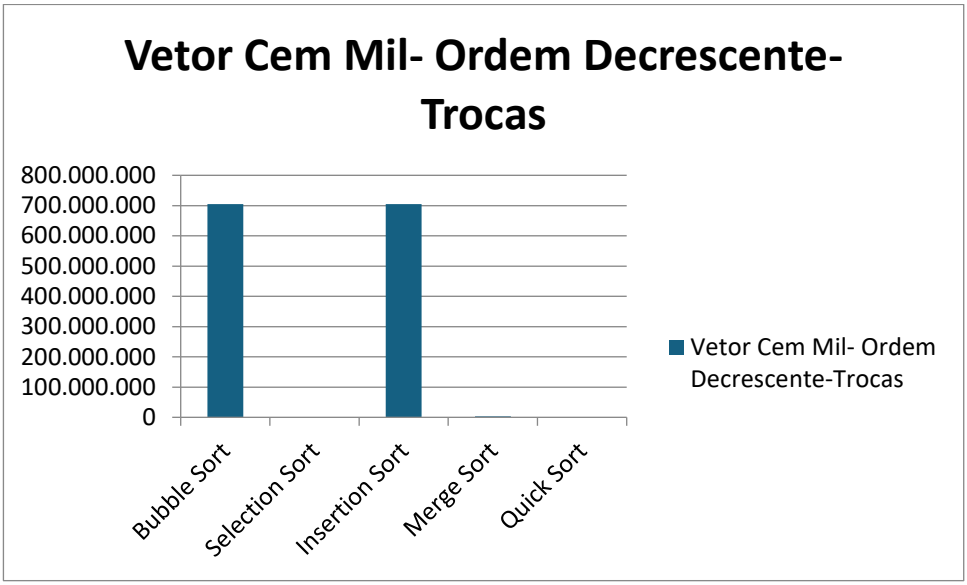


Na ordem aleatória e decrescente os melhores casos foram o **Merge Sort** e os piores foram o **Bubble Sort**.






Vetor Cem Mil- Ordem Decrescente- Comparações
800.000.000
700.000.000
600.000.000
500.000.000
400.000.000
300.000.000
200.000.000
100.000.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor Cem Mil- Ordem Decrescente- Comparações


Vetor Cem Mil- Ordem Decrescente- Trocas
800.000.000
700.000.000
600.000.000
500.000.000
400.000.000
300.000.000
200.000.000
100.000.000
0
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick Sort
Vetor Cem Mil- Ordem Decrescente-Trocas

3. Conclusão

Concluiu-se que os resultados da avaliação foram distintos do trabalho proposto por Souza, Ricarte e de Almeida Lima.

O melhor método utilizado para vetores em ordem crescente ordenada foi o **Bubble Sort** devido ao fato de seu funcionamento ser mais simples e não depender de divisão e conquista ou intercalação como outros. Os piores casos ficaram com o **Selection Sort,** já que o número de comparações a ser feito é alto não possuindo muita utilidade no caso desse tipo de ordem.

Já na ordem aleatória, o **Bubble Sort** aparece como pior caso e o **Merge** e **Quick Sort** como melhores casos. Esses dois últimos possuem tempo de execução menores quando se trata de vetores com alta quantidade de elementos.

No caso da ordem decrescente, os piores casos também foram com o método **Bubble Sort** e os melhores com o **Merge Sort** exclusivamente.

Nesse presente trabalho o método que obteve mais sucesso foi o **Merge Sort** pela menor quantidade de comparações e trocas e pelo tempo de execução menor que os outros.

4. Fontes

Souza, J. E. G., Ricarte, J. V. G., & Lima, N. C. de A. (2017). **Algoritmos de Ordenação: Um Estudo Comparativo**. *Anais Do Encontro De Computação Do Oeste Potiguar ECOP/UFERSA (ISSN 2526-7574)*, *1*(1). Recuperado de https://periodicos.ufersa.edu.br/ecop/article/view/7082.