Análise dos algoritmos de ordenação

Introdução:

O presente trabalho é sobre os algoritmos de ordenação: Bubble Sort, Selection Sort, Insertion Sort, Shaker Sort, Quicksort e Mergesort. São usados para organizar listas de números ou strings de acordo com o que o programador necessita.

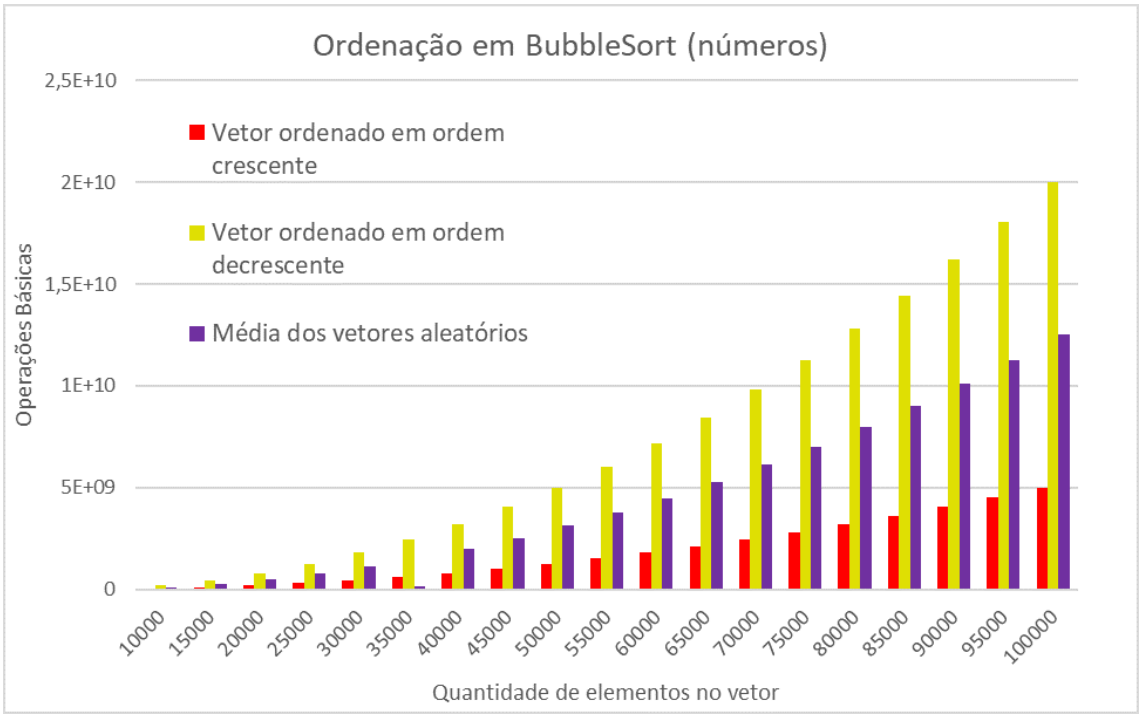
Analisaremos e compararemos o tempo de execução dos algoritmos ordenando vetores de 10000 até 100000 elementos. Gerando valores aleatórios, será testado cada um dos algoritmos ordenando em ordem crescente e decrescente.

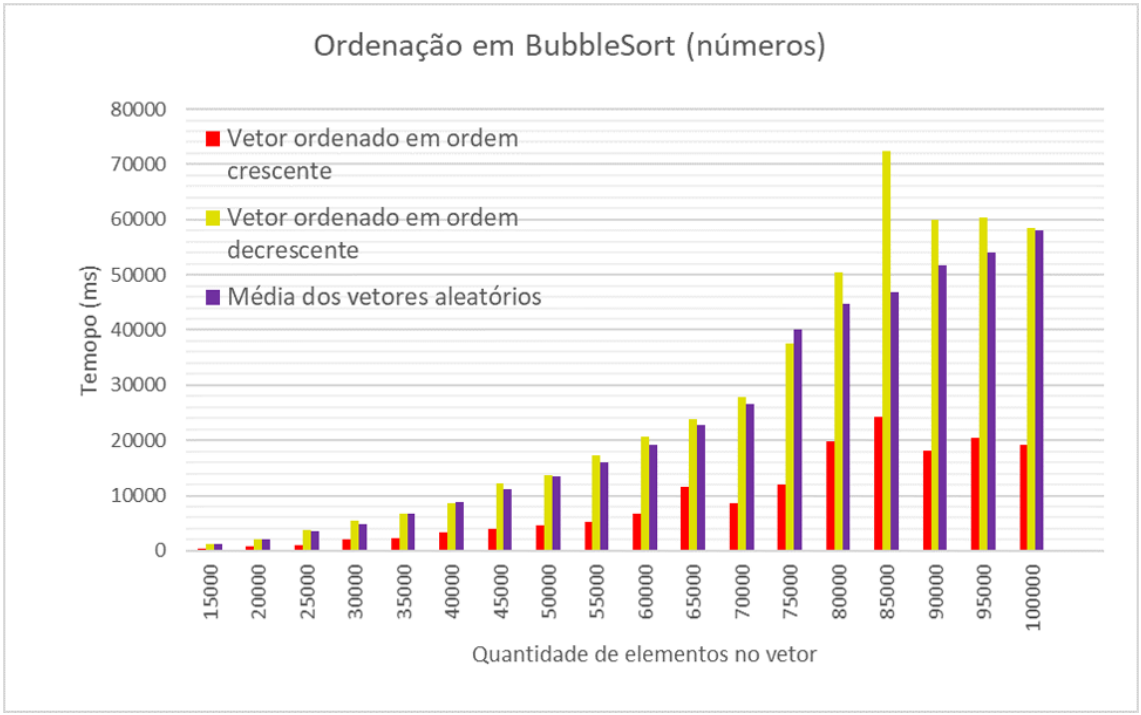
Metodologia:

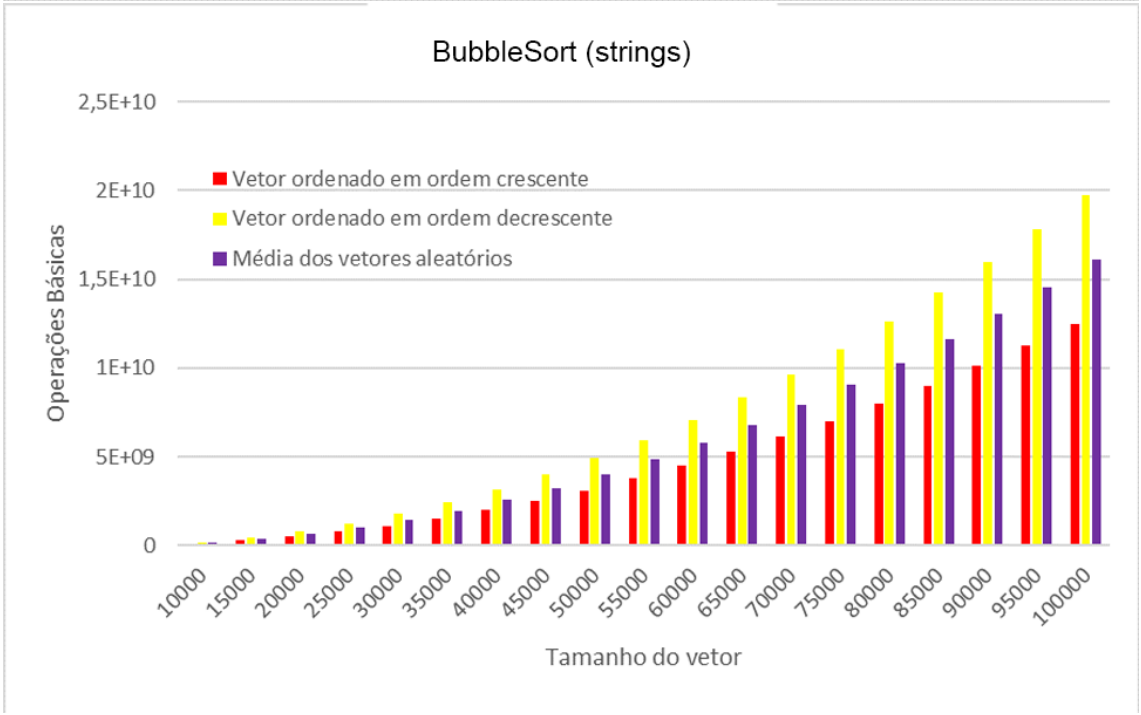
Para a realização desse trabalho foram usados algoritmos para gerar vetores de números e strings aleatórios, ordená-los em cada um dos casos e medir o tempo e as operações básicas gastos. Com o resultado dos algoritmos, foram feitos gráficos para expressar o comportamento de cada método de ordenação.

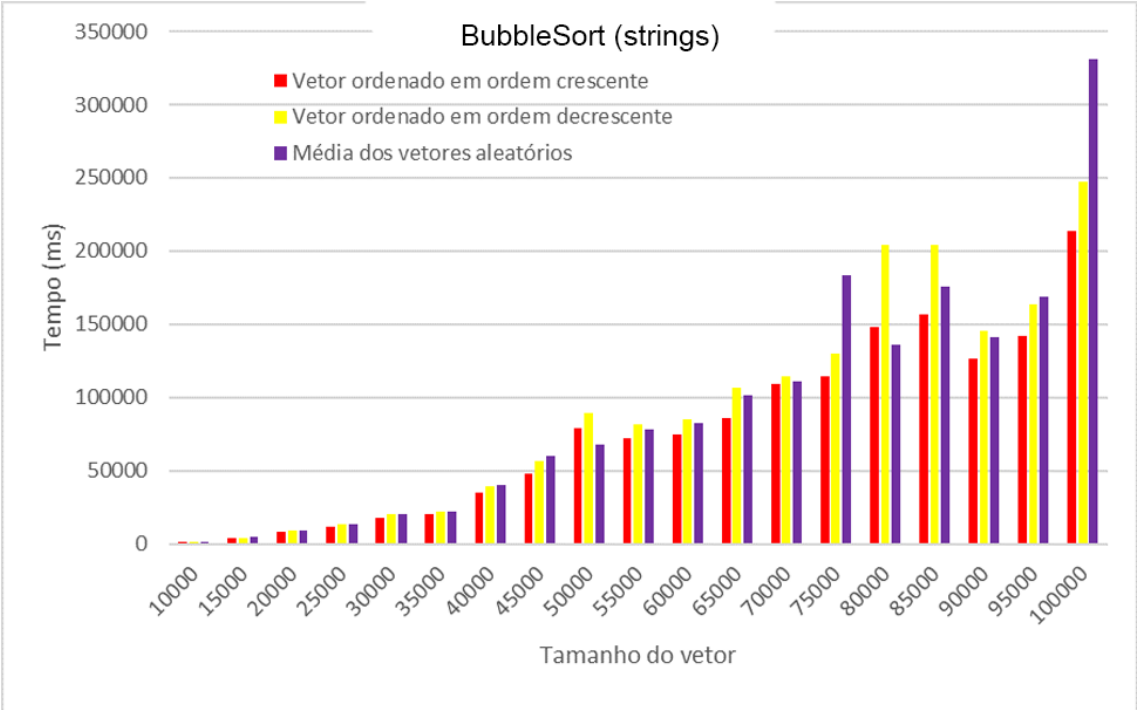
Análise dos algoritmos de ordenação:

Bubble Sort –



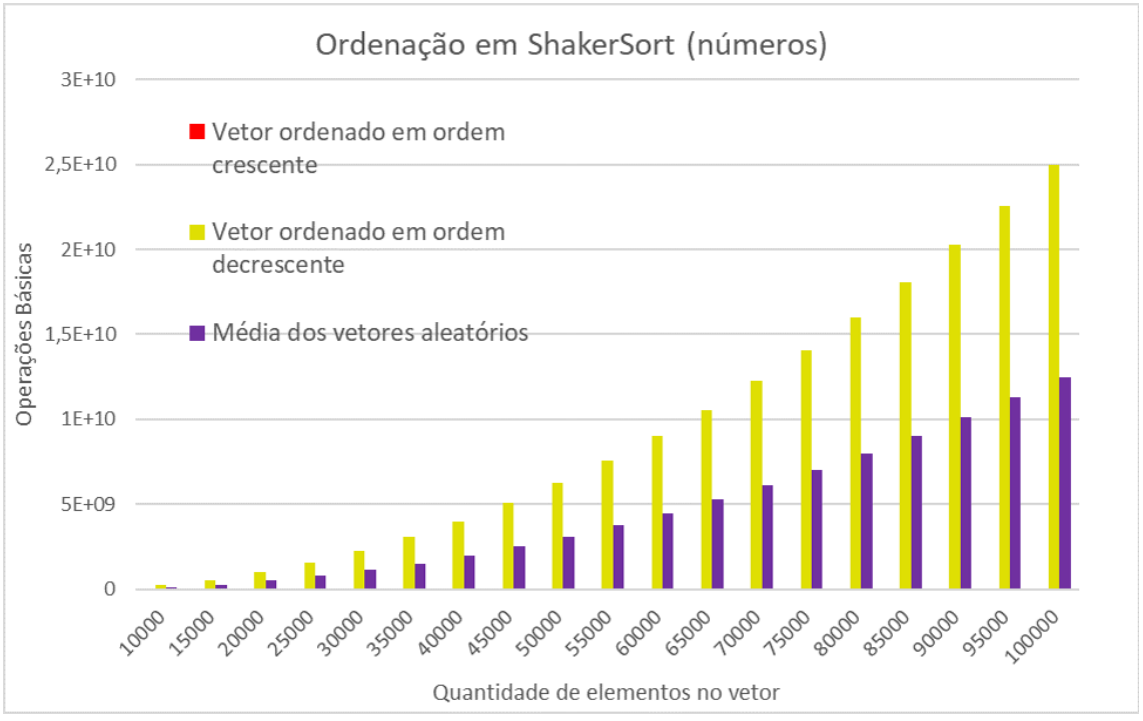

Ordenação em BubbleSort (números)
Temopo (ms)
80000
70000
60000
50000
40000
30000
20000
10000
0
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Quantidade de elementos no vetor

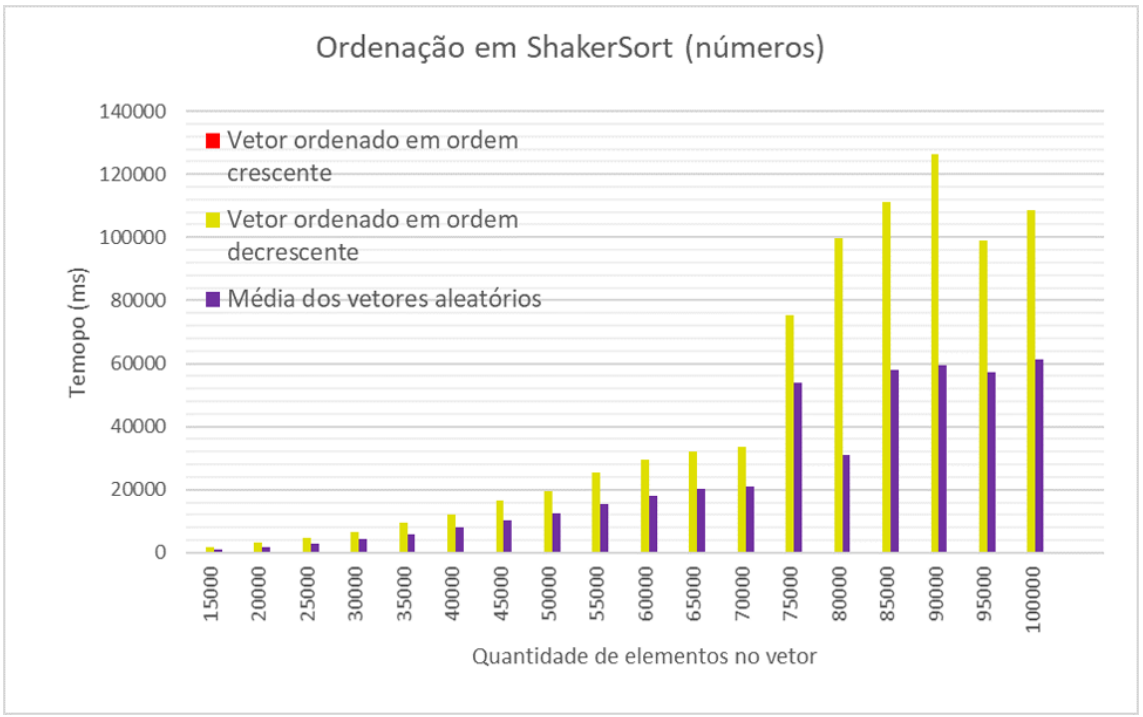

BubbleSort (strings)
Operações Básicas
2,5E+10
2E+10
1,5E+10
1E+10
5E+09
0
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Tamanho do vetor

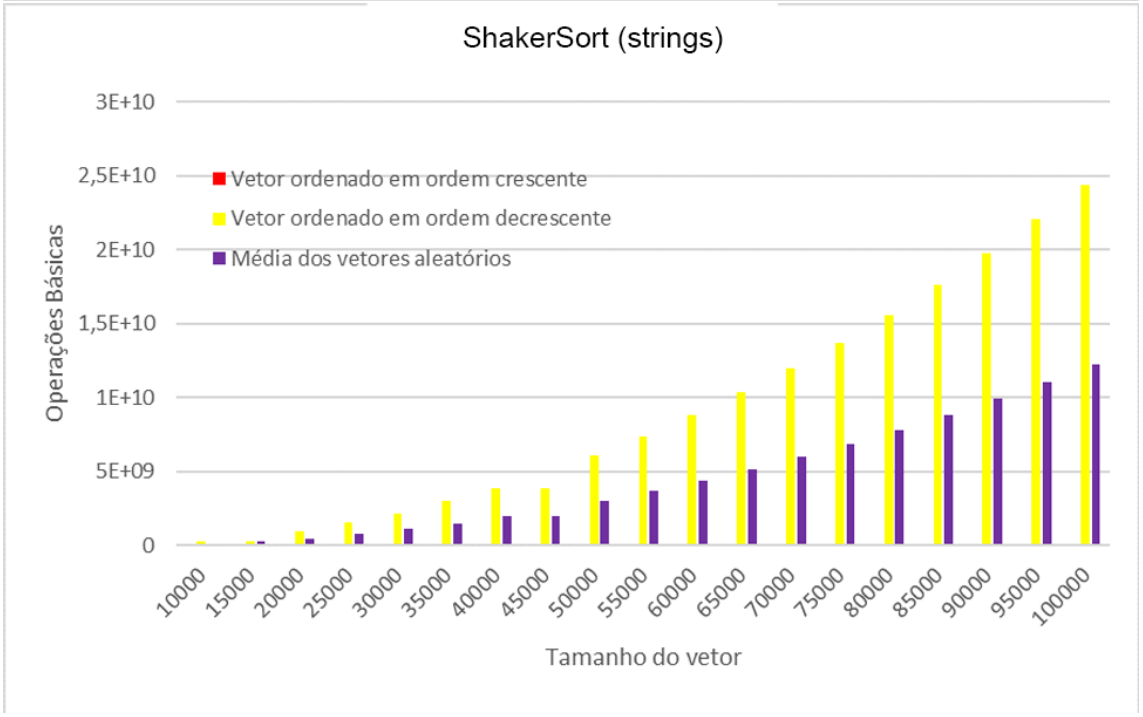


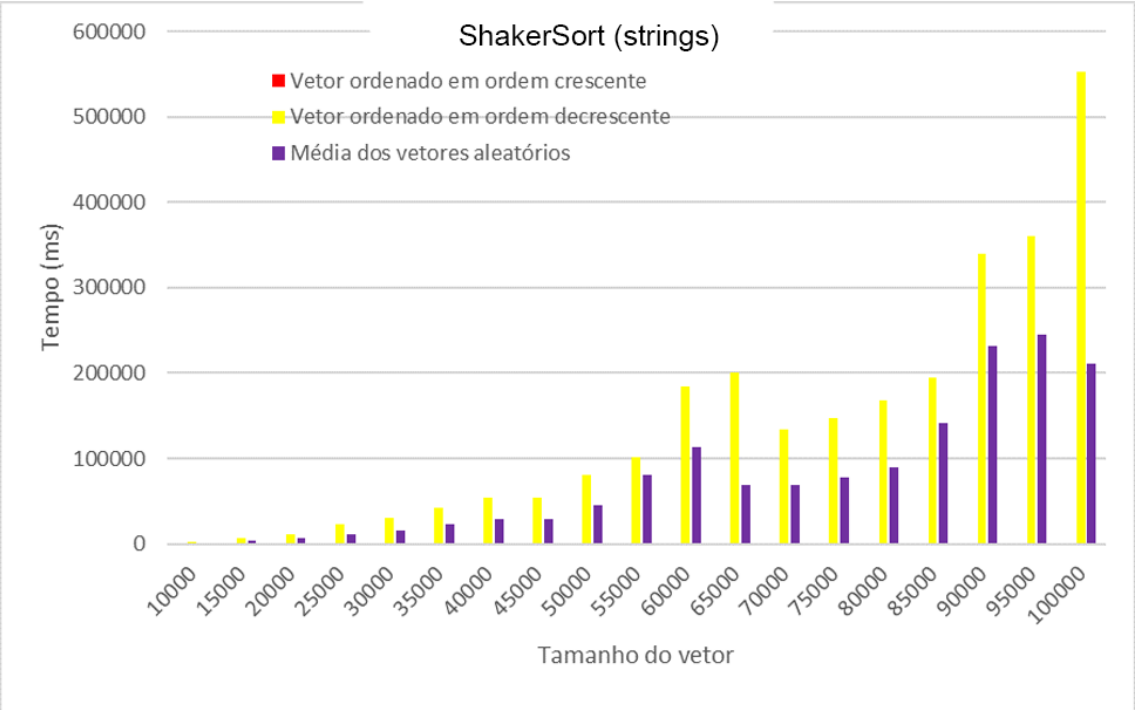
Esse é o mais simples dos algoritmos de ordenação, porém é um dos mais ineficientes, passando de um segundo de tempo de execução na maioria dos casos, chegando até a mais de um minuto de tempo de execução. O bubble sort tem como vantagem a simplicidade do código, podendo ser facilmente alterado conforme a vontade do usuário. O gráfico desse algoritmo se assemelha ao de uma função quadrática.

Shaker Sort –



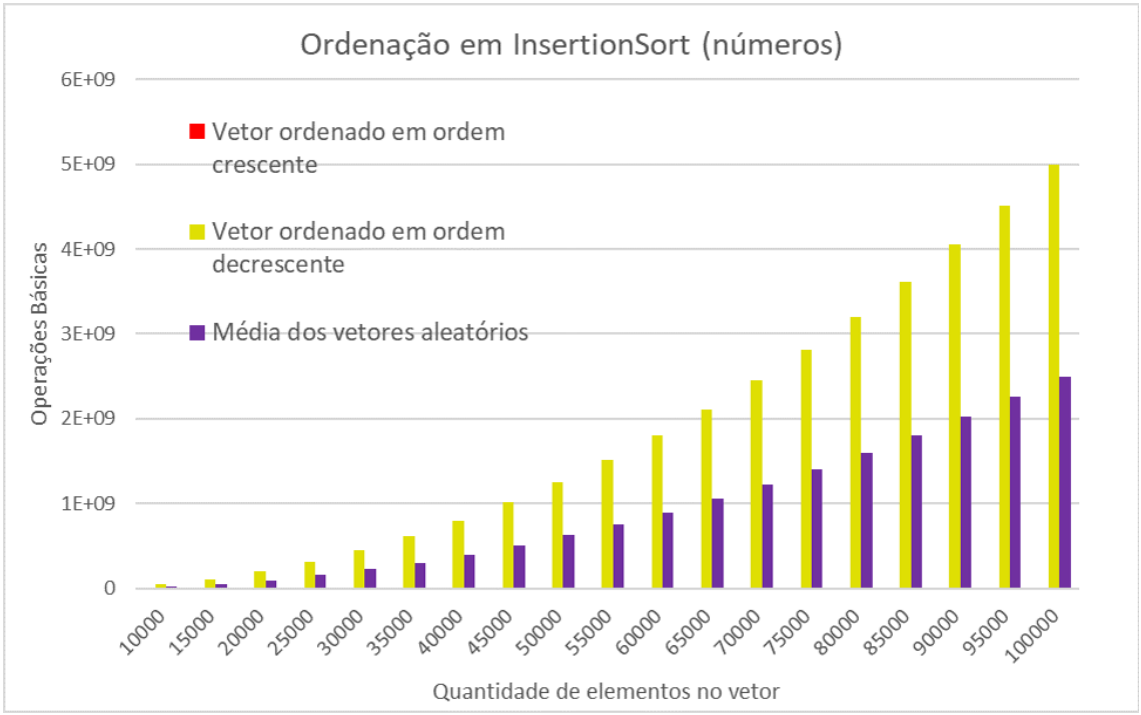

Ordenação em ShakerSort (números)
140000
120000
100000
80000
60000
40000
20000
0
Temopo (ms)
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Quantidade de elementos no vetor

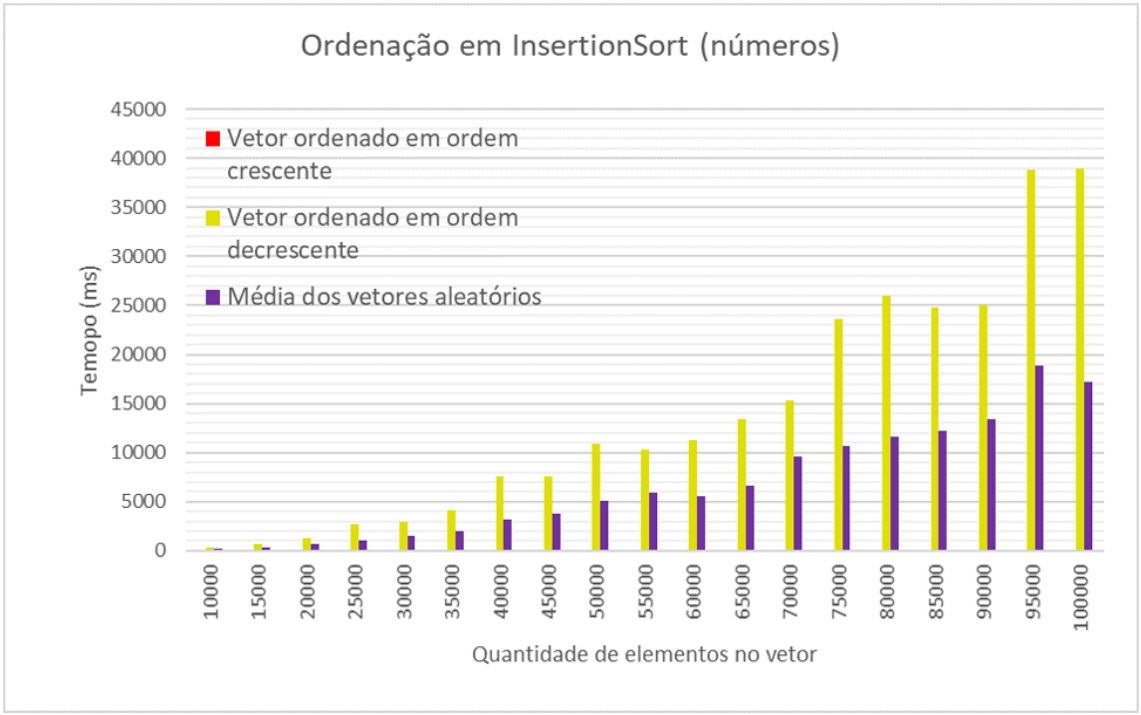

ShakerSort (strings)
3E+10
2,5E+10
2E+10
1,5E+10
1E+10
5E+09
0
Operações Básicas
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Tamanho do vetor

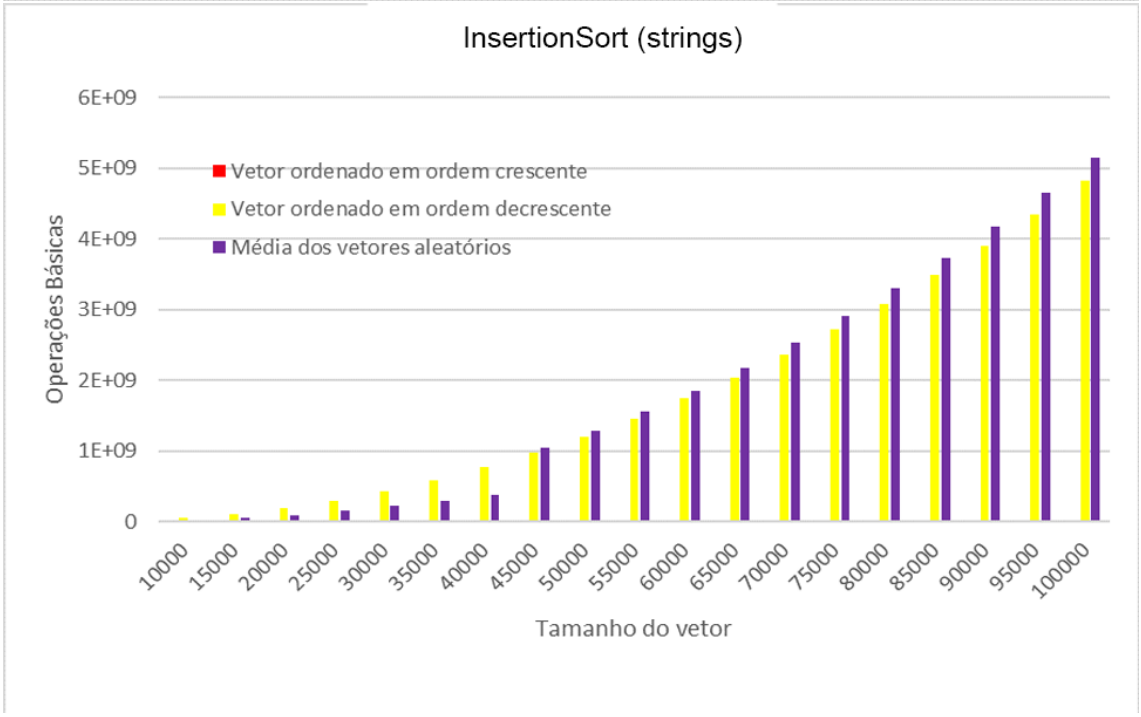


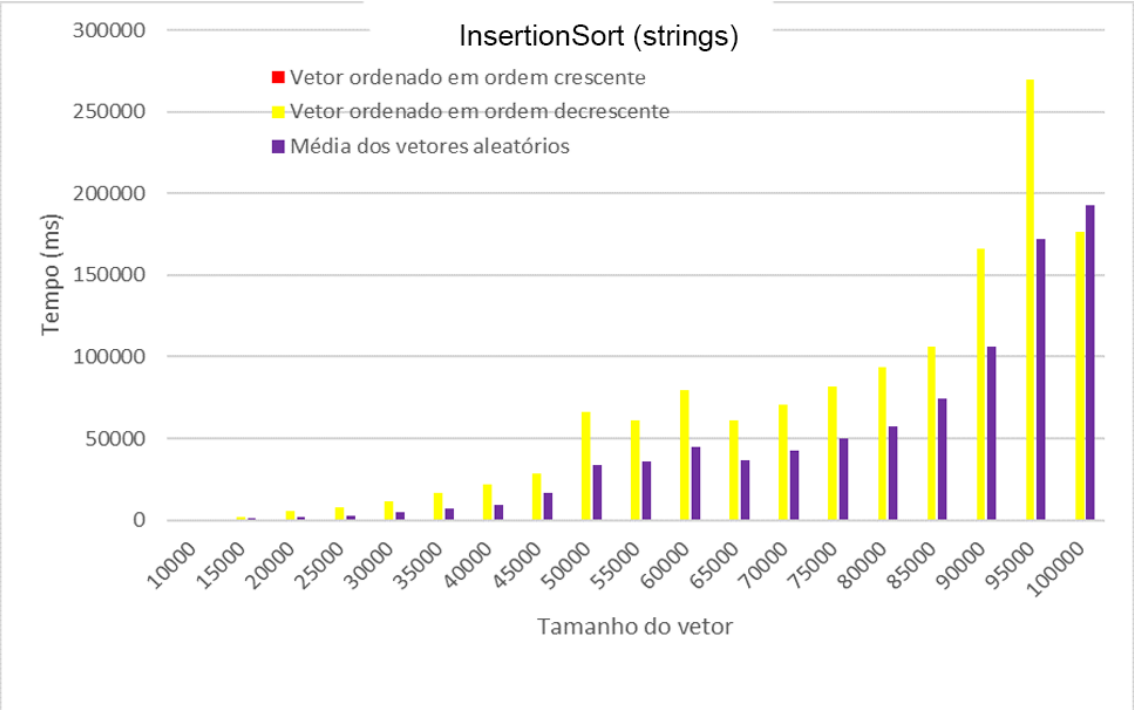
O shaker sort é tão lento quanto o bubble sort na maioria dos casos, mas nos casos em que o vetor já está ordenado, ele é extremamente rápido, pois, nesses casos, ele se resume a dois "For" do tamanho do vetor. Em vetores em ordem decrescente e aleatório o gráfico se assemelha ao de uma função quadrática e em vetores ordenados o gráfico é linear.

Insertion Sort –



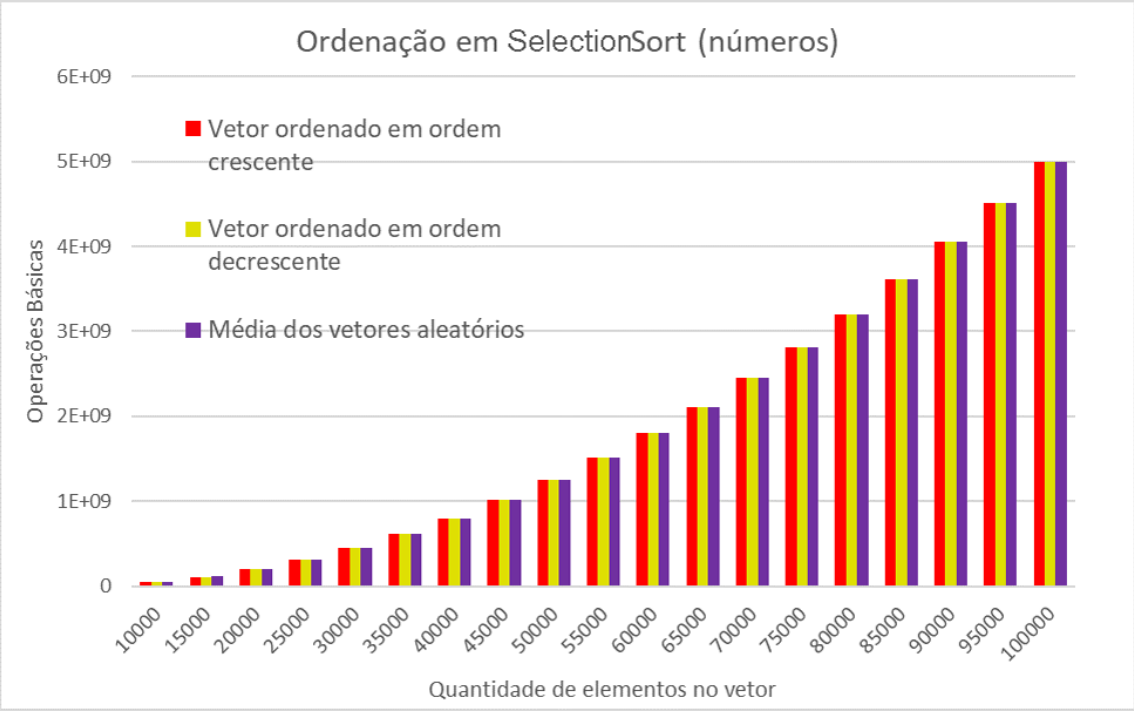

Ordenação em InsertionSort (números)
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Temopo (ms)
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Quantidade de elementos no vetor

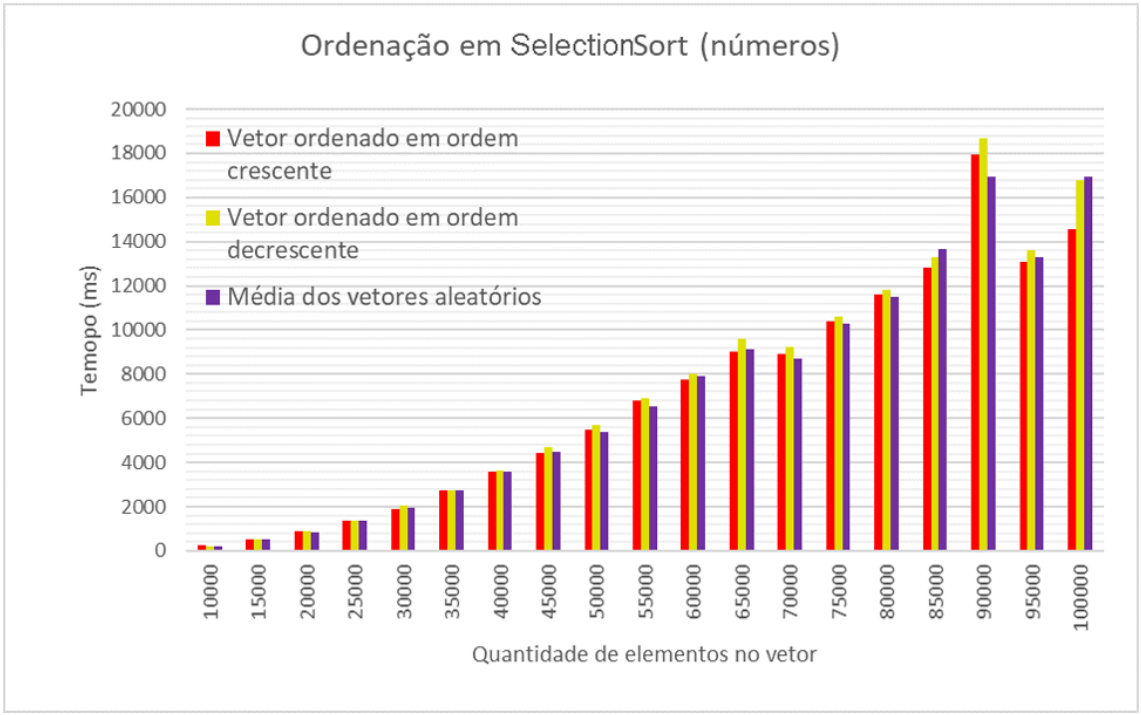

InsertionSort (strings)
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
Operações Básicas
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Tamanho do vetor

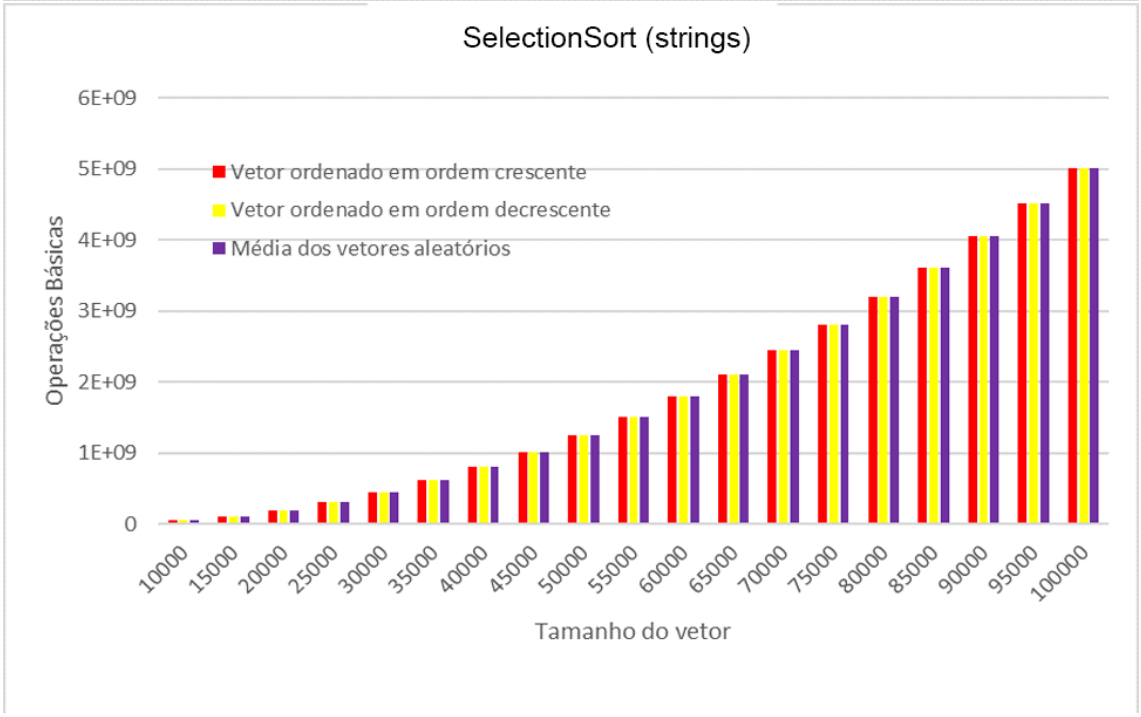


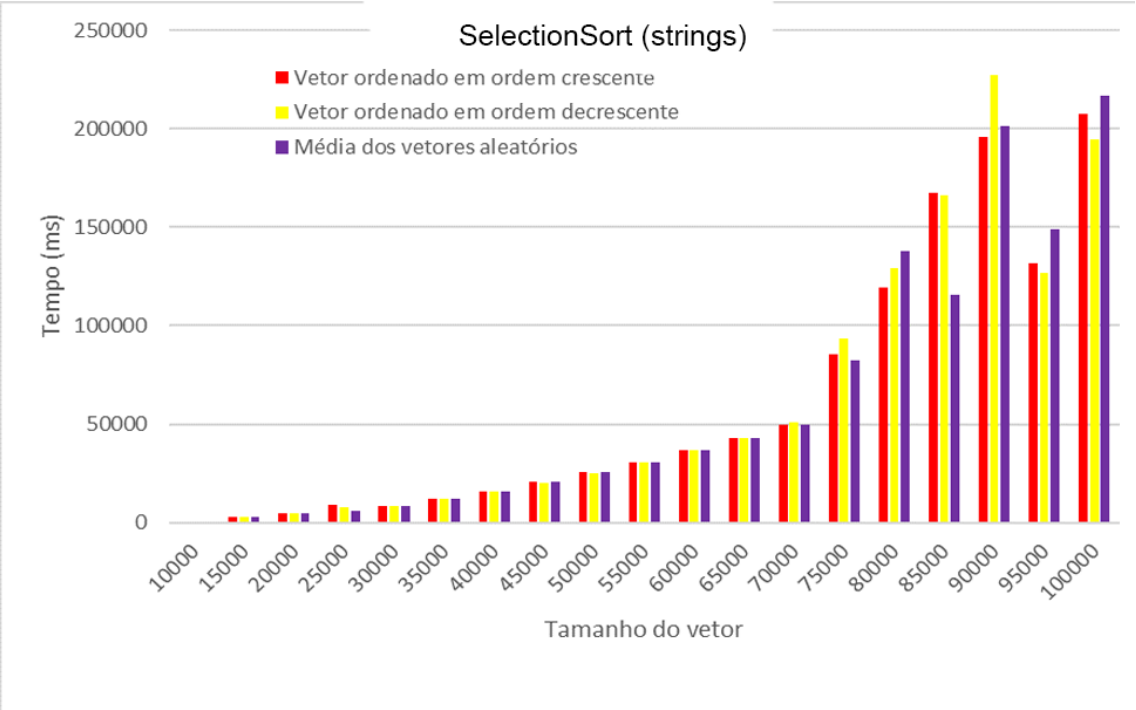
Assim como o shaker sort, o insertion também é muito rápido para vetores já ordenados, e em outros casos é mais rápido que o bubble e o shaker, porém ainda muito lento em vetores muito grandes (como os que foram testados no trabalho). O gráfico desse algoritmo é quadrático.

Selection Sort –



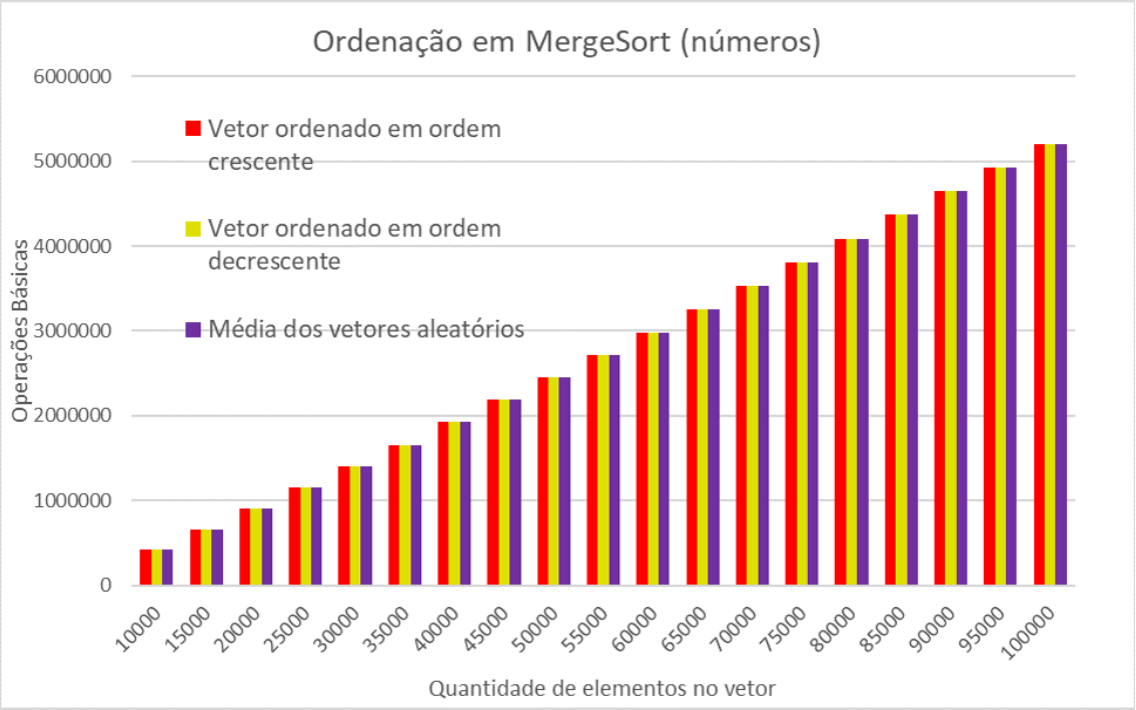

Ordenação em SelectionSort (números)
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
Temopo (ms)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Quantidade de elementos no vetor

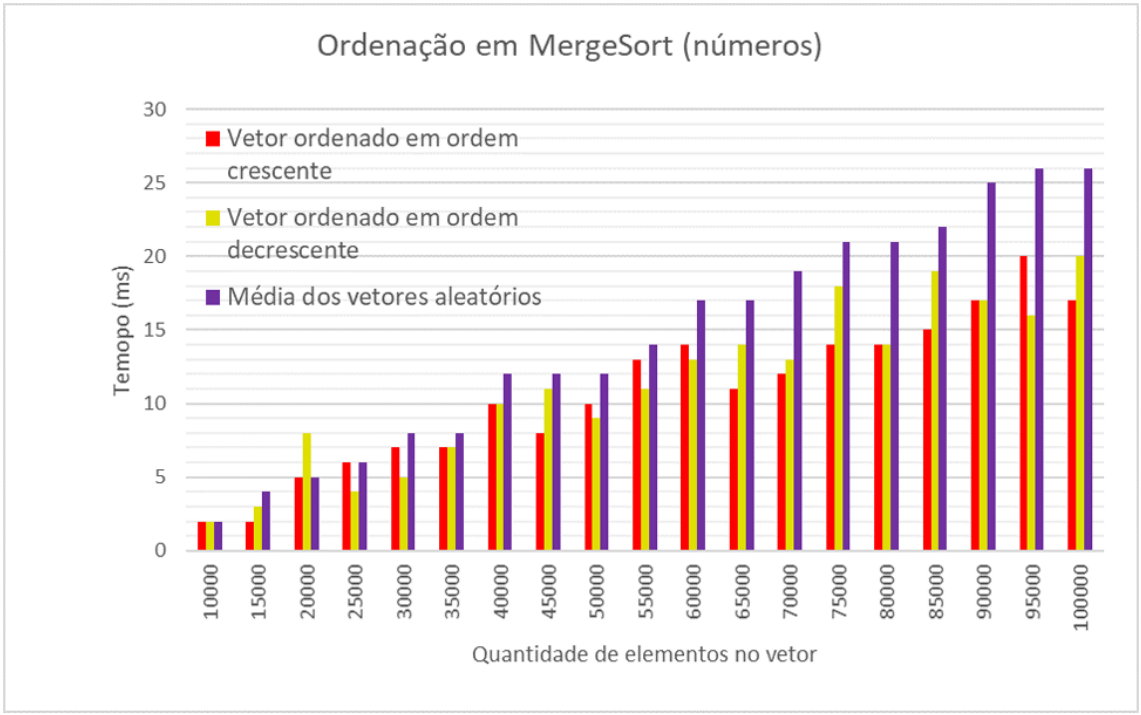

SelectionSort (strings)
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
Operações Básicas
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Tamanho do vetor

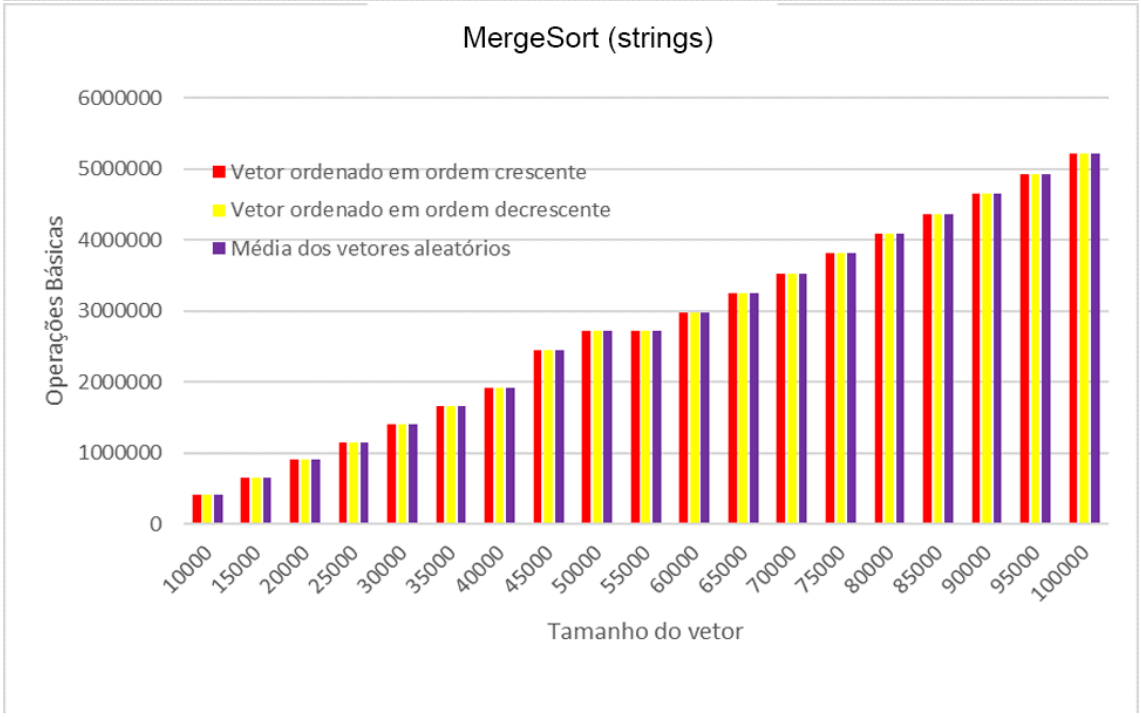


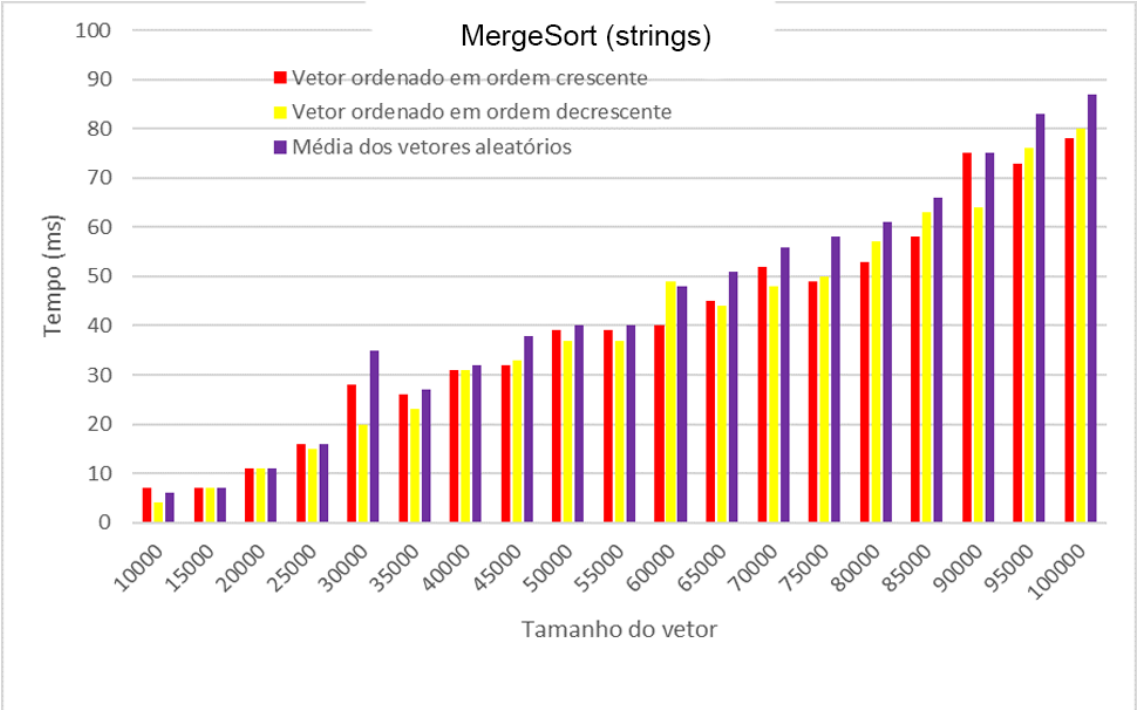
Esse algoritmo teve um desempenho quase igual nos três tipos de vetores ordenados, as operações básicas foram exatamente iguais para os três testes em cada caso. Em geral, o selection sort foi mais rápido que todos os anteriores. O comportamento do gráfico do selection sort também é quadrático.

Merge Sort –



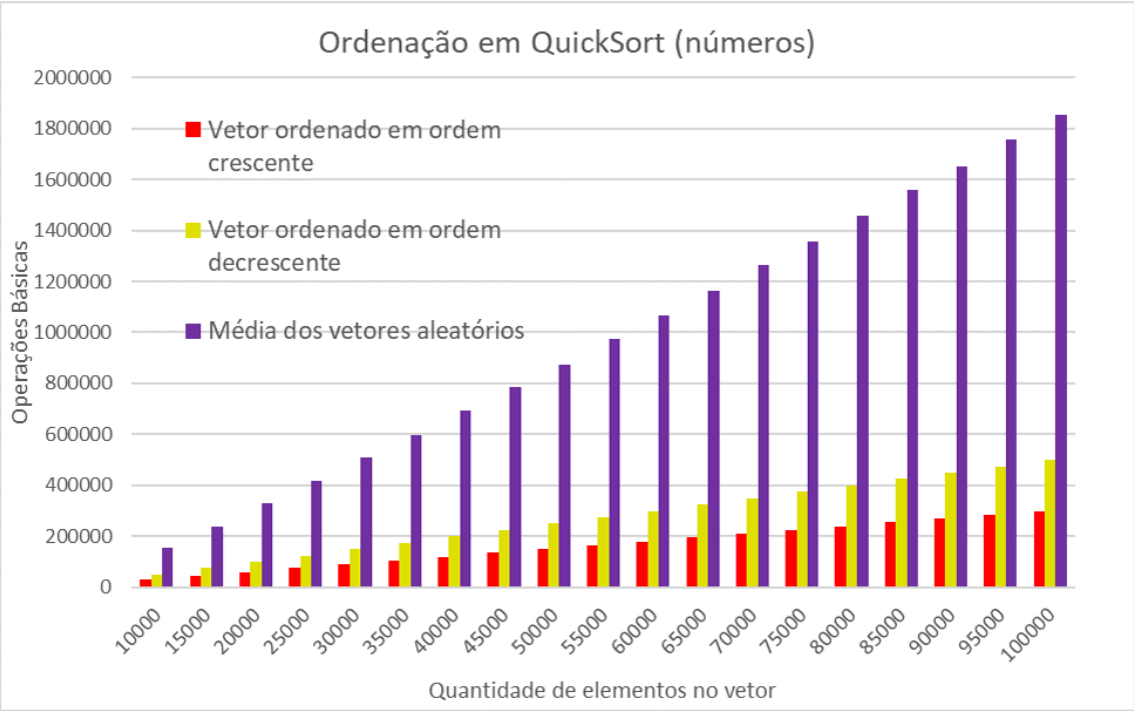

Ordenação em MergeSort (números)
Tempo (ms)
30
25
20
15
10
5
0
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Quantidade de elementos no vetor

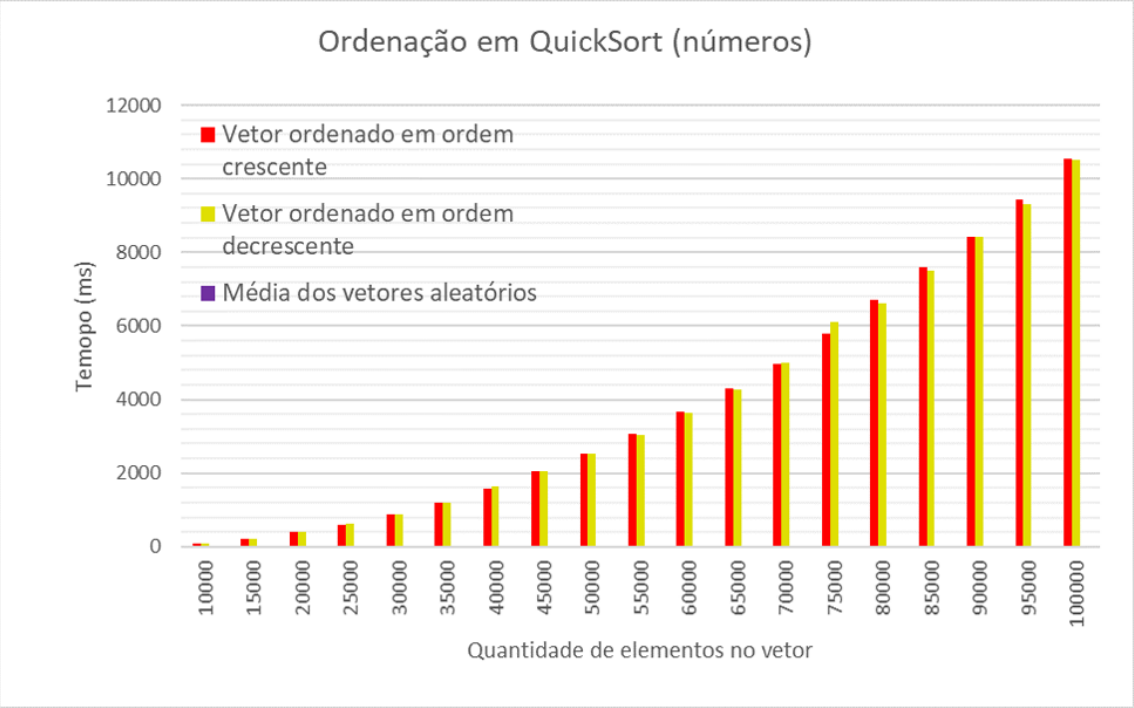

MergeSort (strings)
Operações Básicas
6000000
5000000
4000000
3000000
2000000
1000000
0
Vetor ordenado em ordem crescente
Vetor ordenado em ordem decrescente
Média dos vetores aleatórios
10000
15000
20000
25000
30000
35000
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000
95000
100000
Tamanho do vetor



O merge sort foi extremamente rápido em todos os casos, não passando de um segundo de tempo de execução em nenhum caso, também teve um desempenho parecido para vetores ordenados em ordem crescente, decrescente e os aleatórios, sendo assim, um algoritmo sem muita variação, podendo ser usado em diversos casos sem se preocupar com casos específicos. O gráfico do algoritmo é parecido com o da função logarítmica.

Quick Sort –





O tempo de execução do quick sort em vetores em ordem aleatória foi extremamente rápido, porém nos casos ordenados em ordem crescente e decrescente ele foi muito mais lento, isso acontece pois nesses casos, o desempenho do algoritmo tende ao desempenho do bubble sort. Esse algoritmo é muito rápido e também relativamente simples no código, só deve-se tomar cuidado com casos específicos. O gráfico do quick sort varia entre o quadrático e o logarítmico, tendendo ao quadrático em casos ordenados em ordem crescente e decrescente, e ao logarítmico em casos aleatórios.

Considerações finais

Neste trabalho cumprimos todos os objetivos que nos tínhamos proposto, observando os comportamentos de todos os algoritmos. O que se mostrou mais ineficiente foi o BubbleSort, mesmo com sua vantagem de ter o código simples e ser facilmente alterado, tem um tempo de execução muito longo. O ShakerSort é lento como o BubbleSort na maioria dos casos, mas em em vetores já ordenados é extremamente rápido. O Insertion Sort é como o Shaker em casos de vetores ordenados, e em outros casos já consegue ter uma execução mais rápida que o Bubble e o Shaker. Sendo mais rápido que todos os algoritmos citados acima, o SelectionSort teve um comportamento muito parecido nos três tipos de vetores ordenados. O QuickSort tende a ser muito rápido em vetores aleatórios, mas é lento em vetores de ordem crescente e decrescente. O que se mostrou mais eficiente de todos os algoritmos para inteiros e também com strings foi o MergeSort.