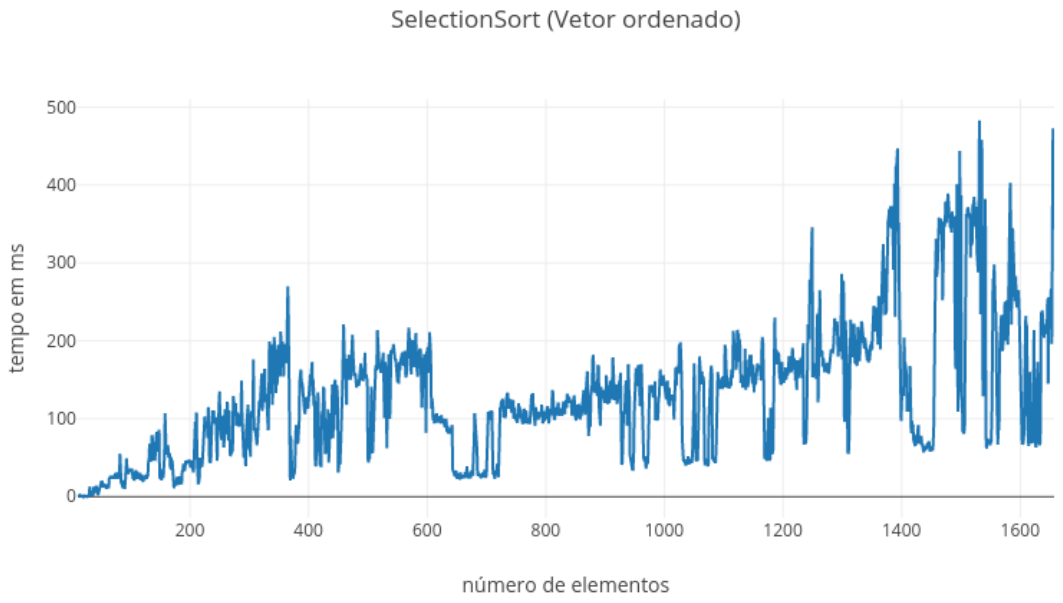
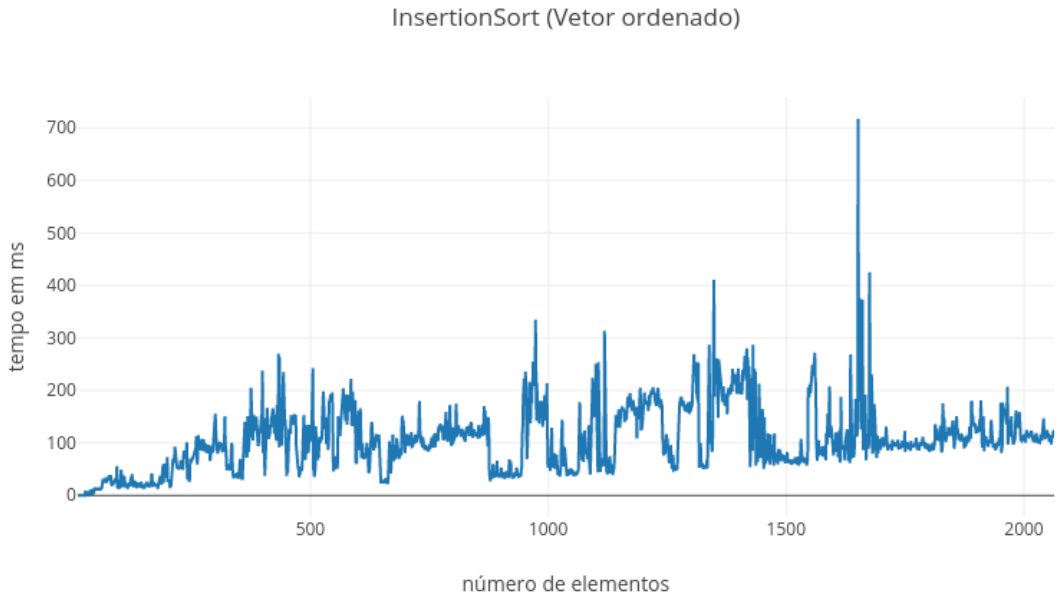
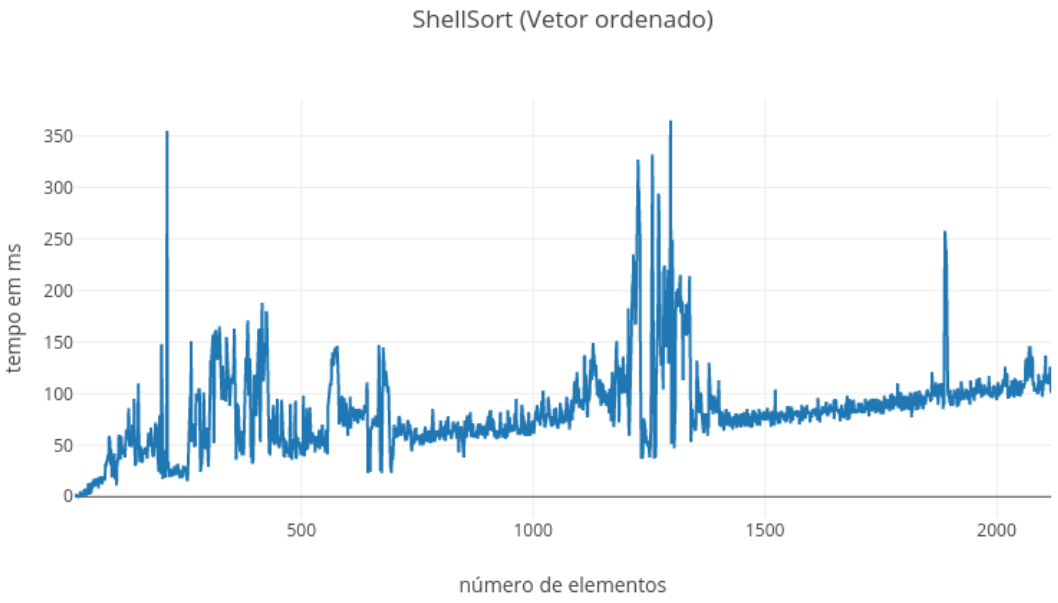
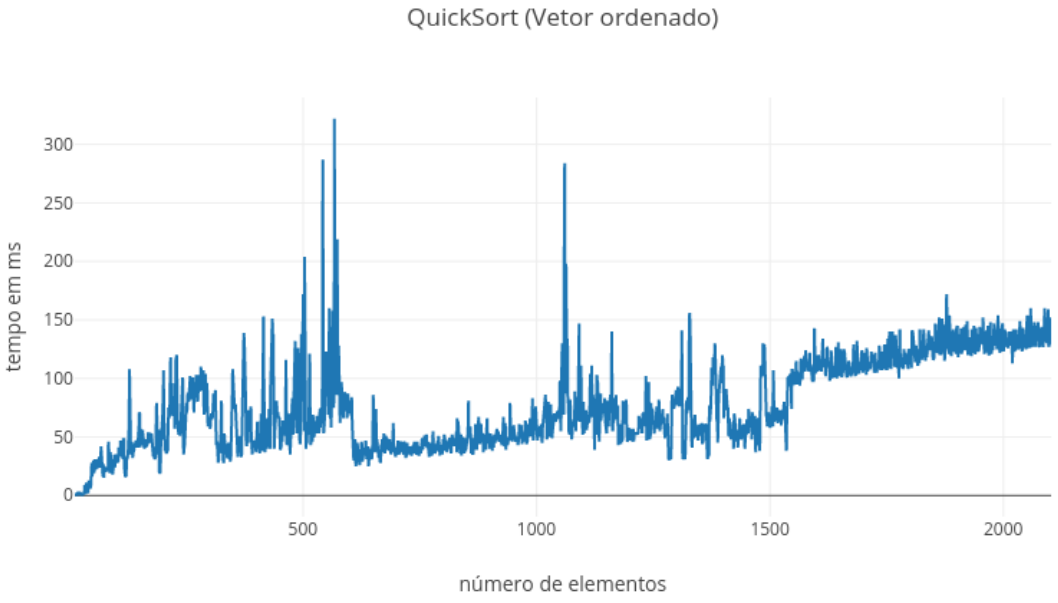
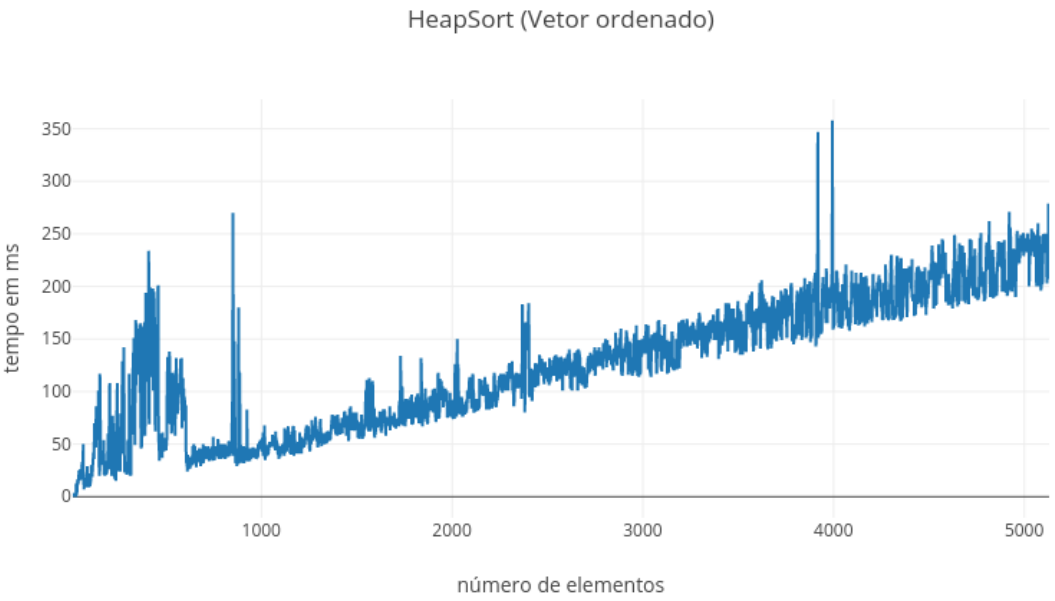
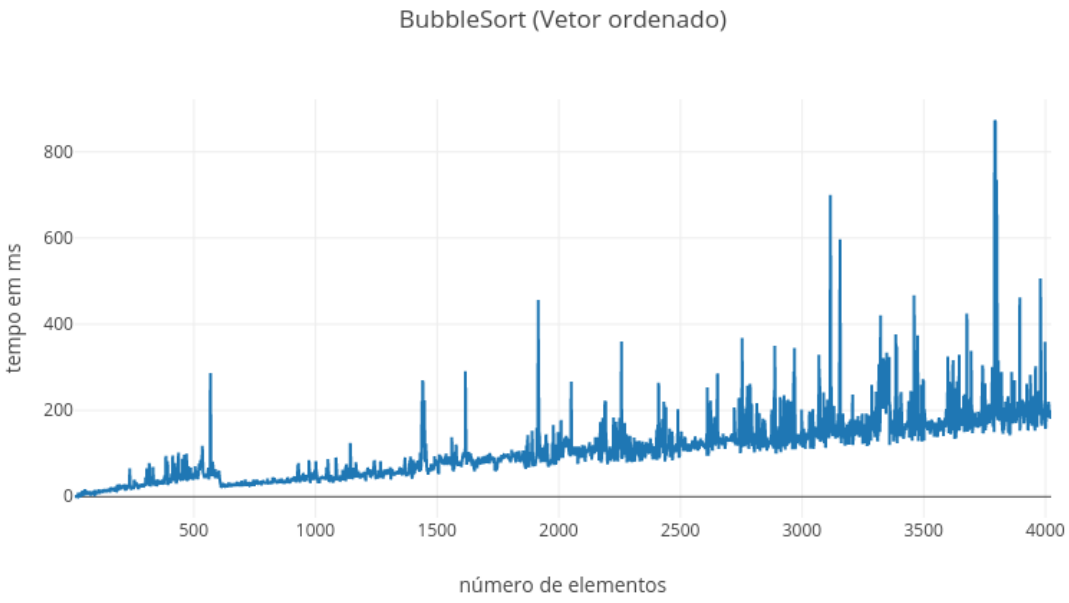

SelectionSort (Vetor ordenado)
tempo em ms
0
100
200
300
400
500
200
400
600
800
1000
1200
1400
1600
número de elementos

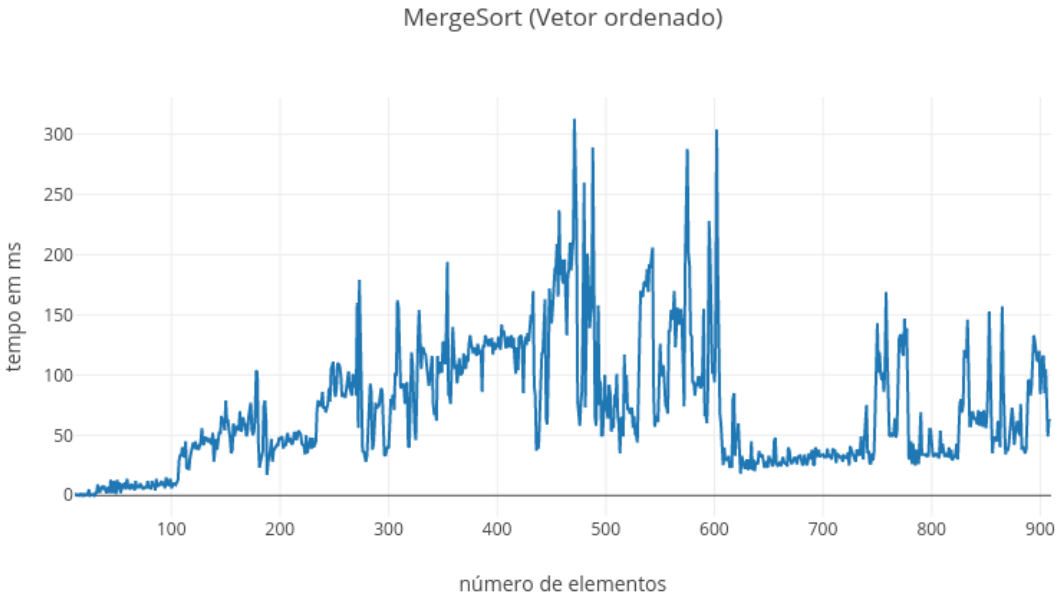

InsertionSort (Vetor ordenado)
tempo em ms
0
100
200
300
400
500
600
700
500
1000
1500
2000
número de elementos

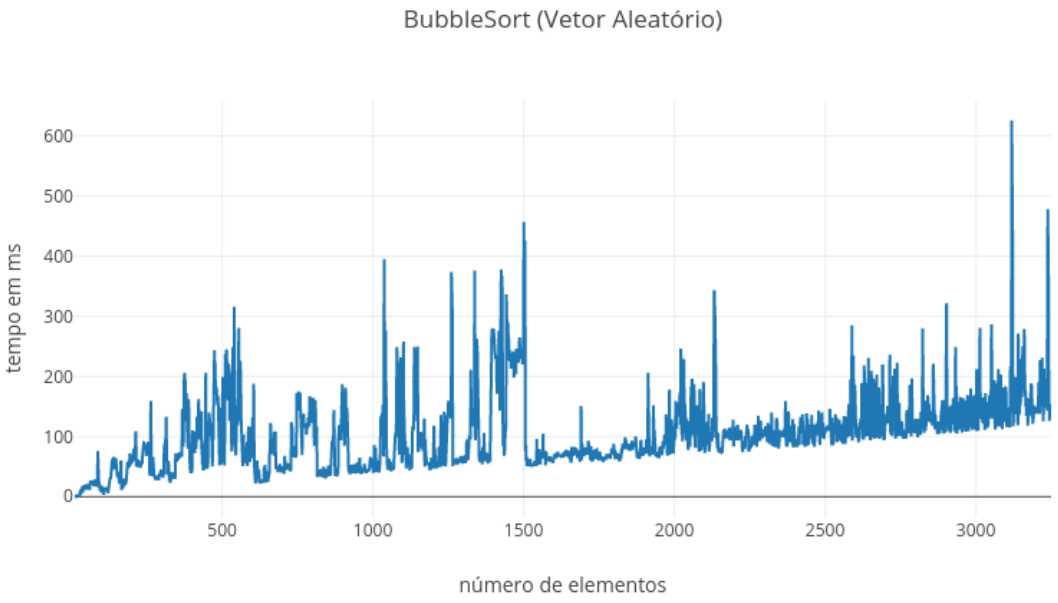

ShellSort (Vetor ordenado)
tempo em ms
0
50
100
150
200
250
300
350
500
1000
1500
2000
número de elementos

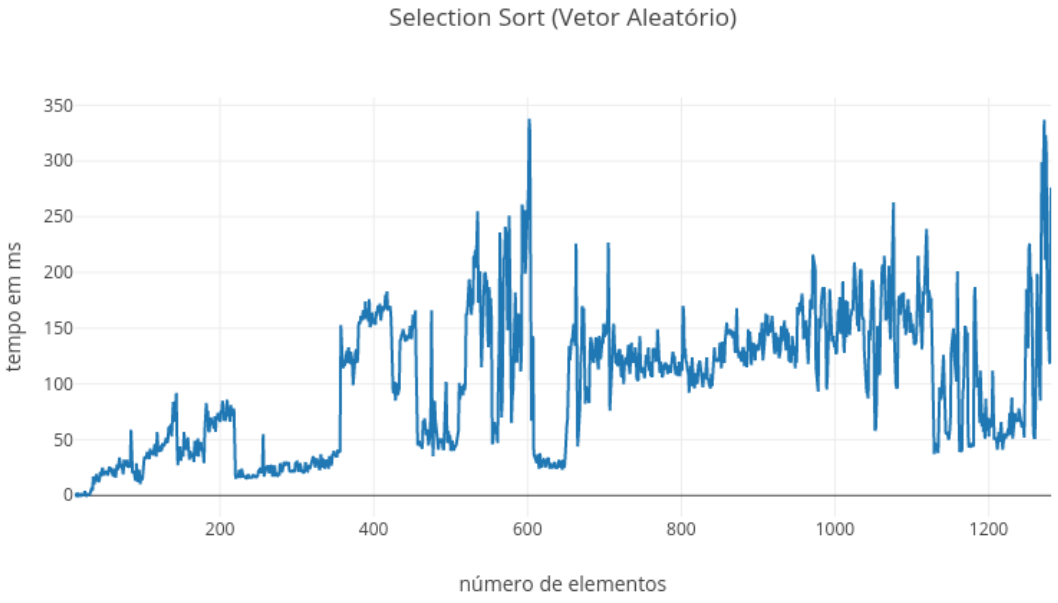

QuickSort (Vetor ordenado)
tempo em ms
0
50
100
150
200
250
300
500
1000
1500
2000
número de elementos

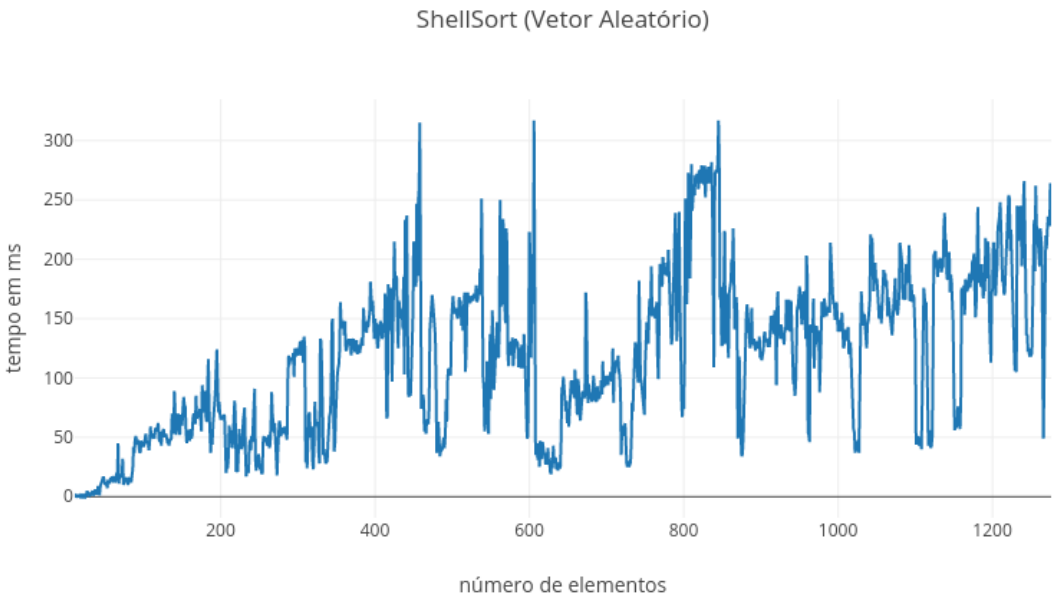

HeapSort (Vetor ordenado)
tempo em ms
350
300
250
200
150
100
50
0
1000
2000
3000
4000
5000
número de elementos

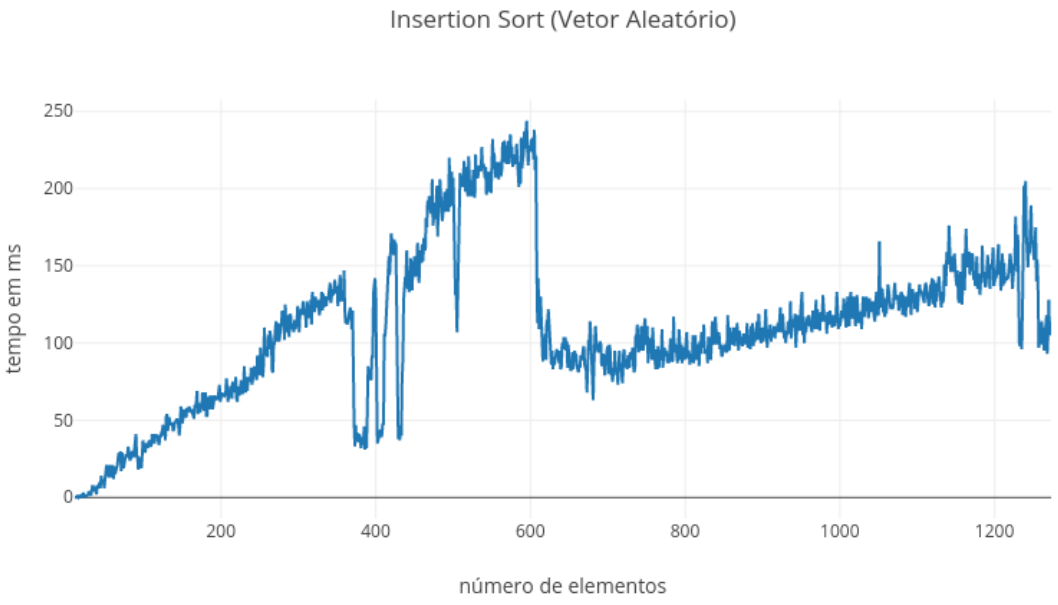

BubbleSort (Vetor ordenado)
tempo em ms
0
200
400
600
800
500
1000
1500
2000
2500
3000
3500
4000
número de elementos

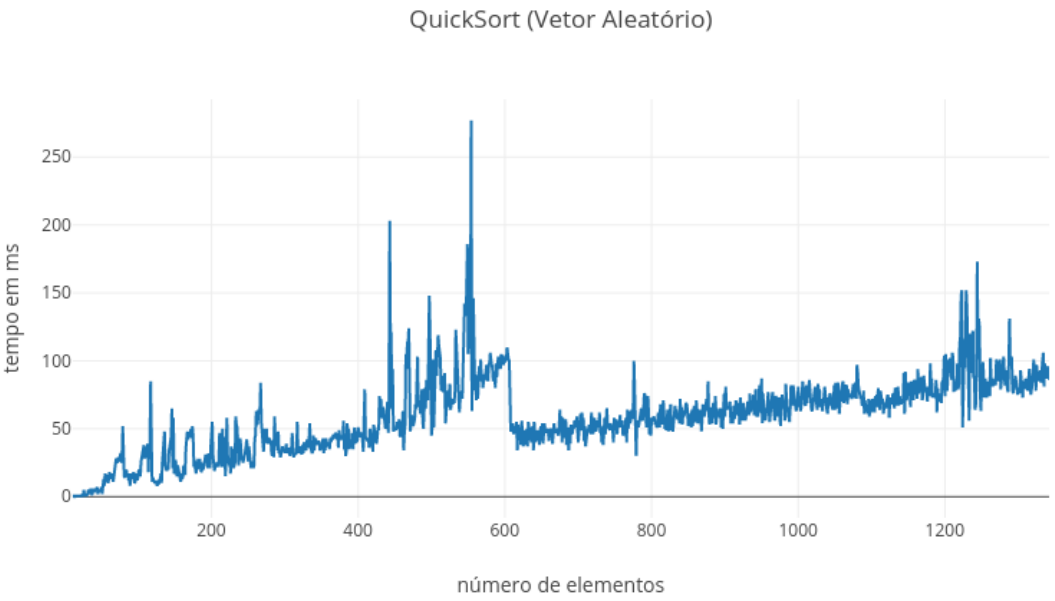

MergeSort (Vetor ordenado)
tempo em ms
0
50
100
150
200
250
300
100
200
300
400
500
600
700
800
900
número de elementos

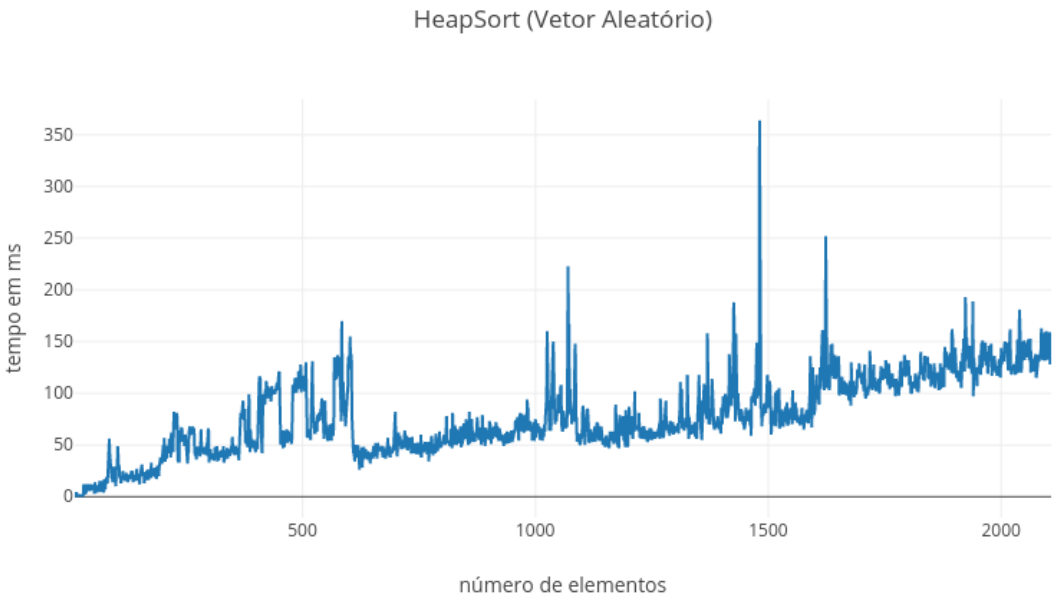

BubbleSort (Vetor Aleatório)
tempo em ms
0
100
200
300
400
500
600
500
1000
1500
2000
2500
3000
número de elementos

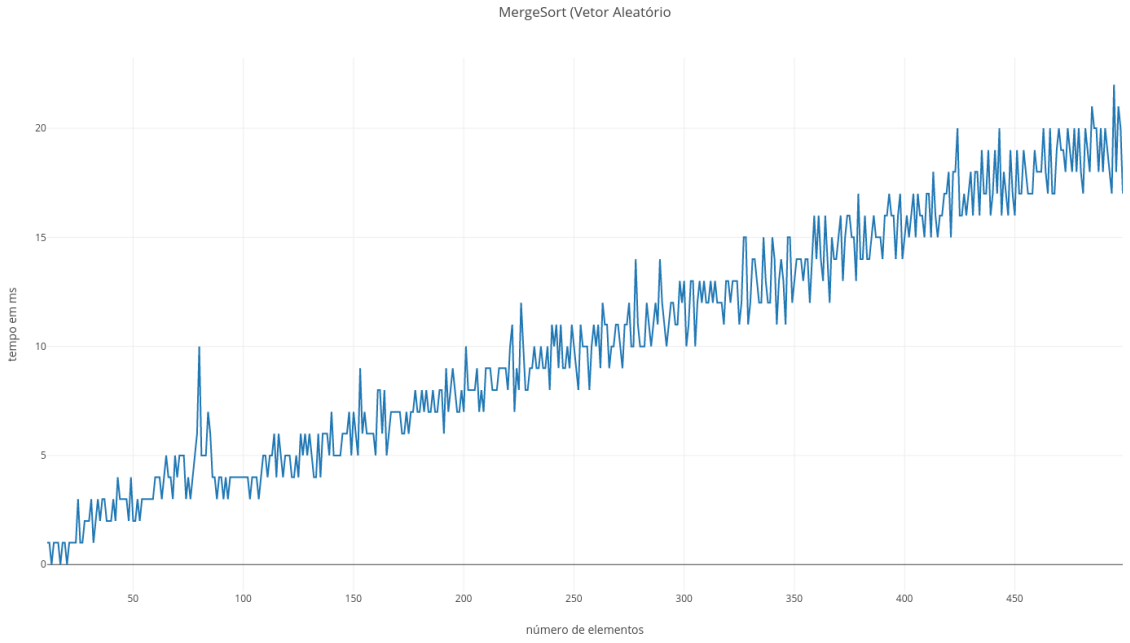

Selection Sort (Vetor Aleatório)
tempo em ms
0
50
100
150
200
250
300
350
200
400
600
800
1000
1200
número de elementos

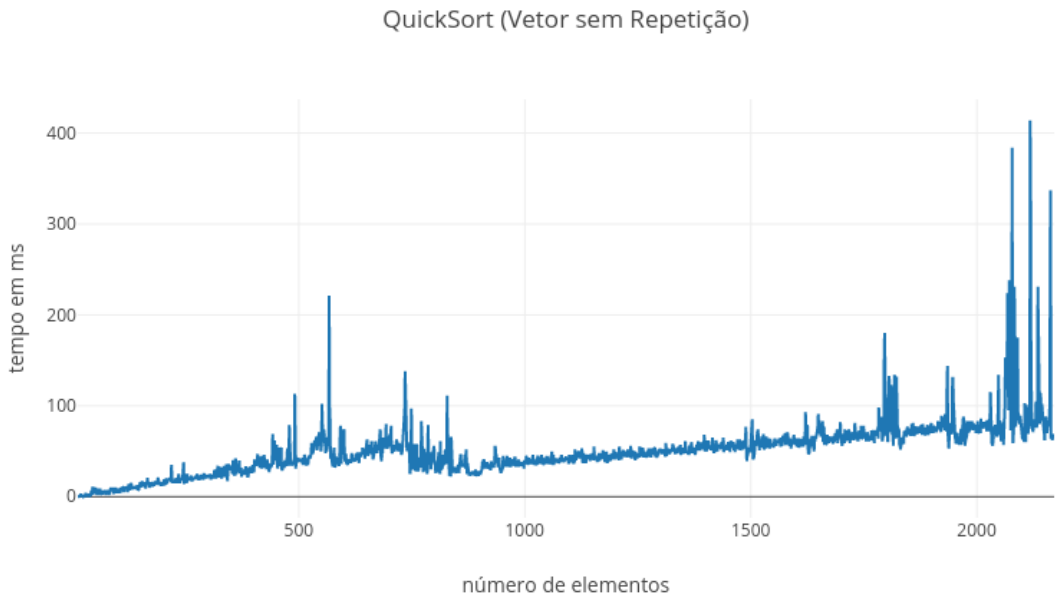

ShellSort (Vetor Aleatório)
tempo em ms
300
250
200
150
100
50
0
200
400
600
800
1000
1200
número de elementos

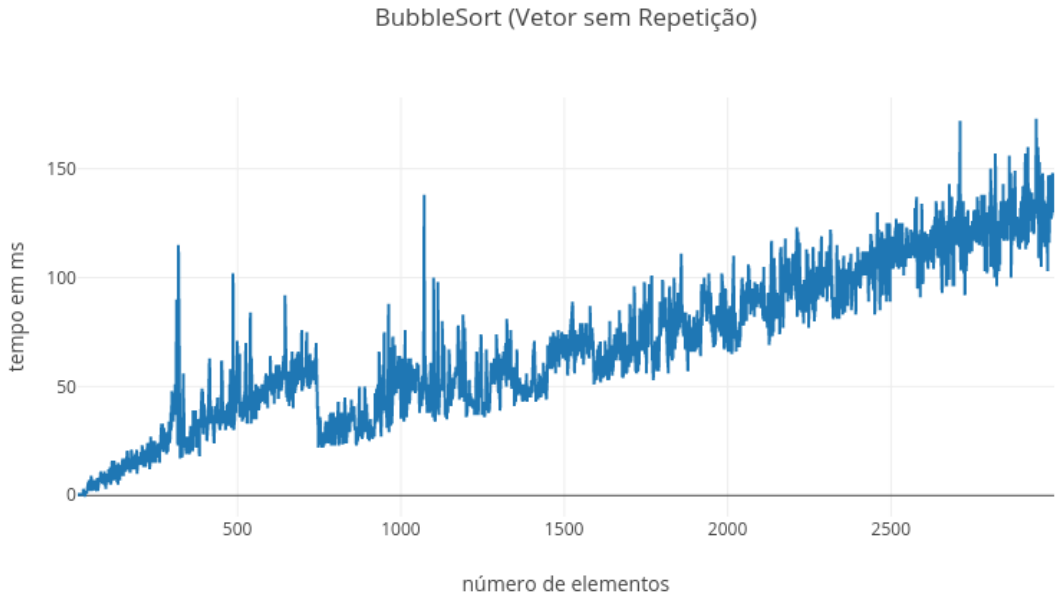

Insertion Sort (Vetor Aleatório)
tempo em ms
250
200
150
100
50
0
200
400
600
800
1000
1200
número de elementos

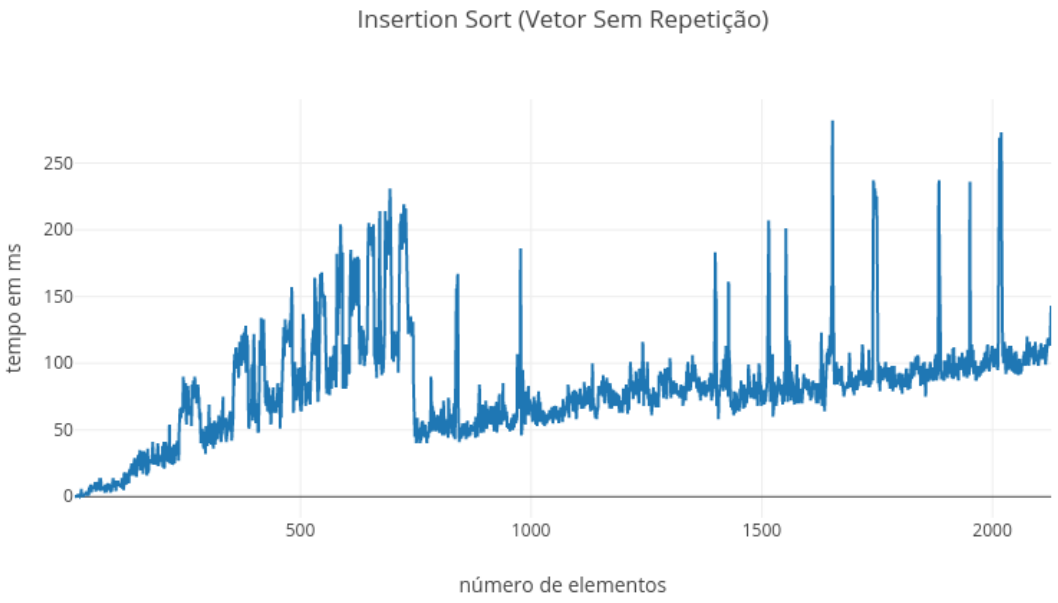

QuickSort (Vetor Aleatório)
tempo em ms
250
200
150
100
50
0
200
400
600
800
1000
1200
número de elementos

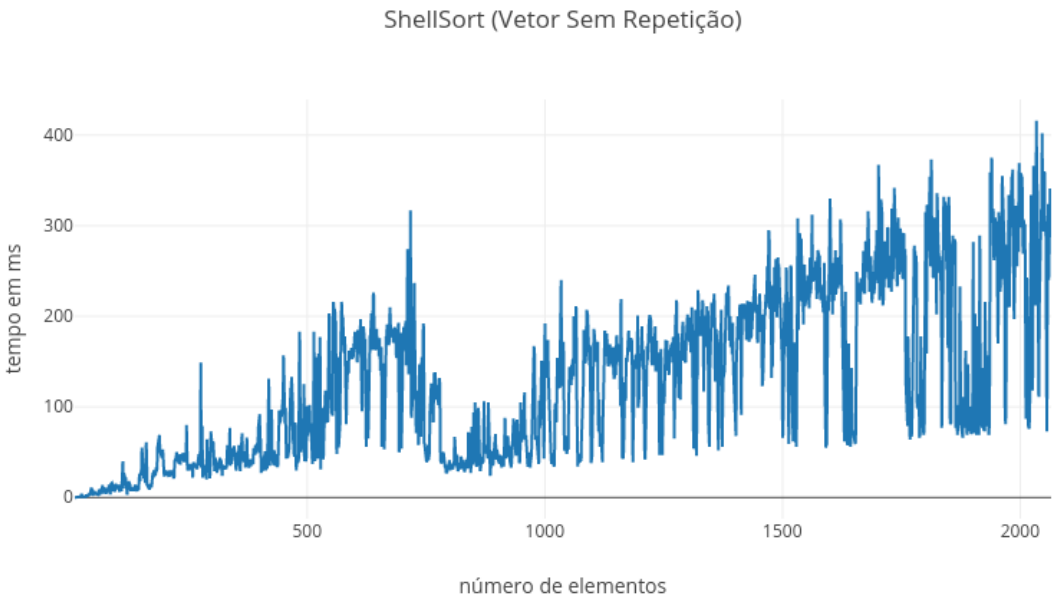

HeapSort (Vetor Aleatório)
tempo em ms
0
50
100
150
200
250
300
350
500
1000
1500
2000
número de elementos

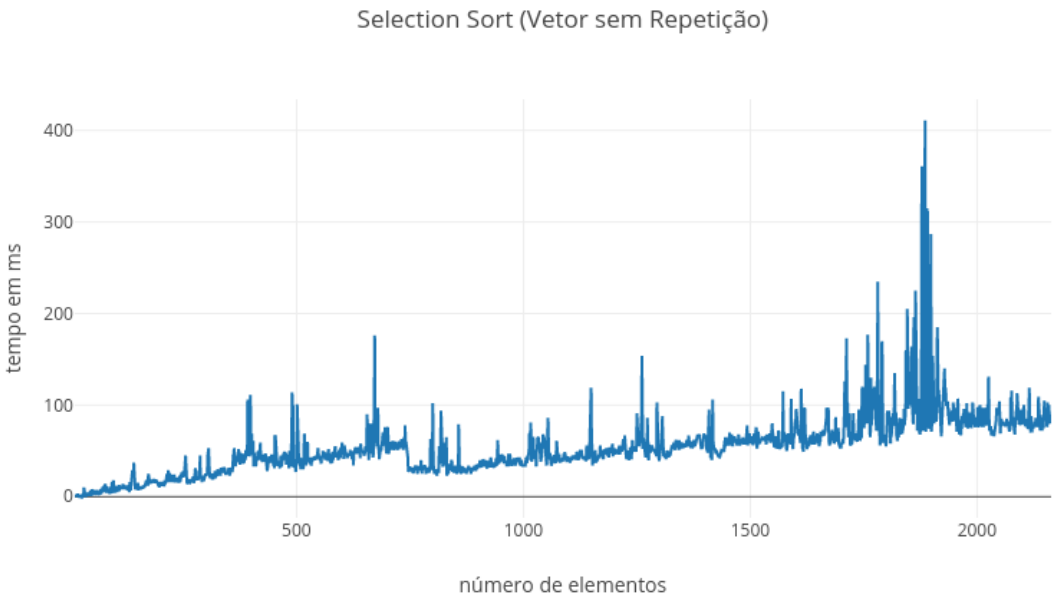

MergeSort (Vetor Aleatório
tempo em ms
0
5
10
15
20
50
100
150
200
250
300
350
400
450
número de elementos

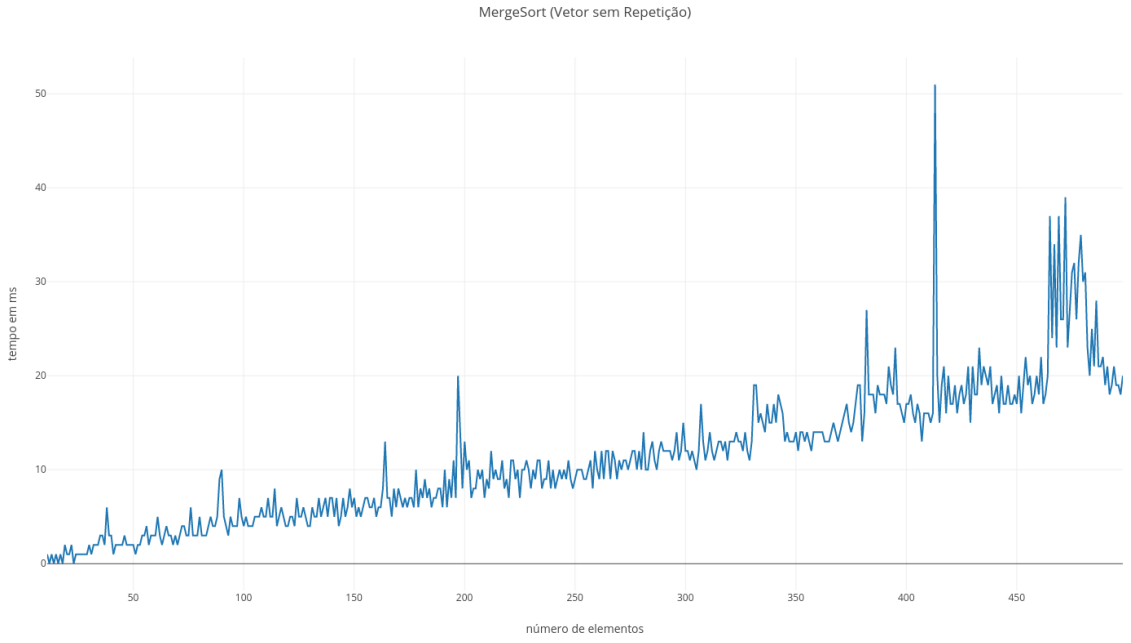

QuickSort (Vetor sem Repetição)
tempo em ms
0
100
200
300
400
500
1000
1500
2000
número de elementos

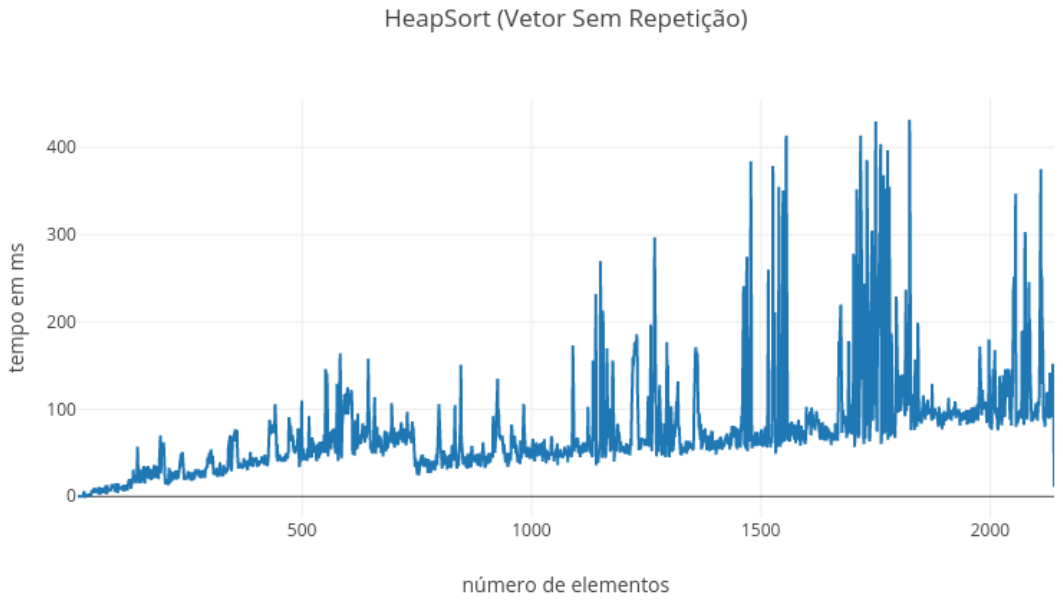

BubbleSort (Vetor sem Repetição)
tempo em ms
0
50
100
150
500
1000
1500
2000
2500
número de elementos

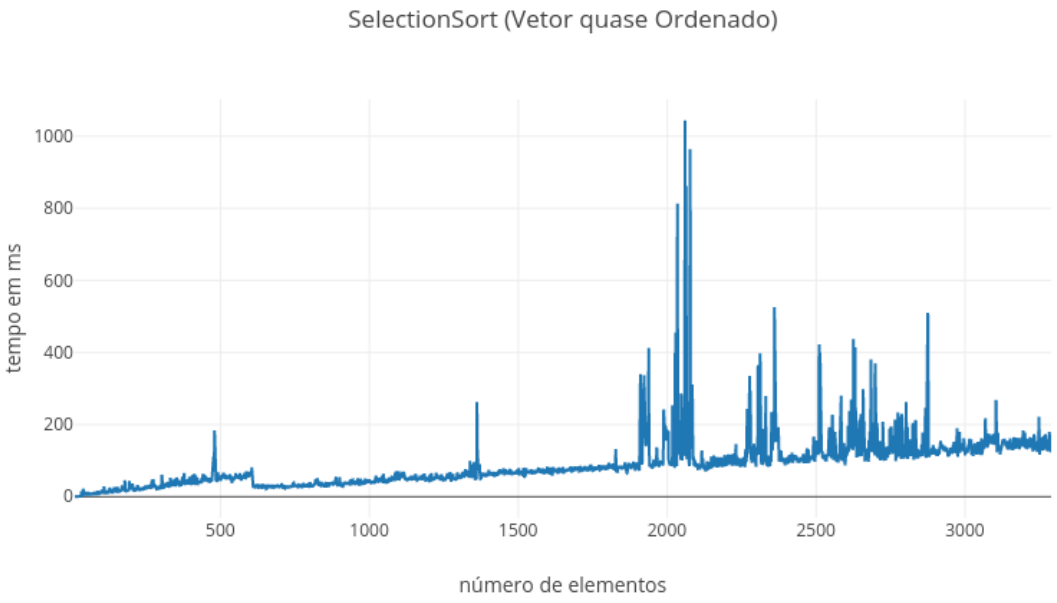

Insertion Sort (Vetor Sem Repetição)
tempo em ms
250
200
150
100
50
0
500
1000
1500
2000
número de elementos

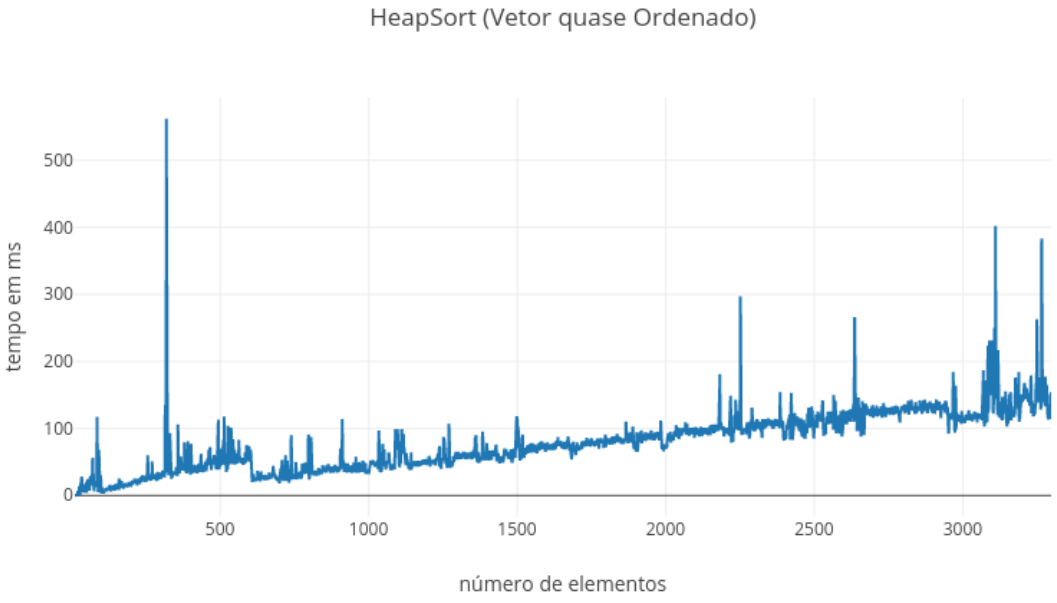

ShellSort (Vetor Sem Repetição)
tempo em ms
400
300
200
100
0
500
1000
1500
2000
número de elementos

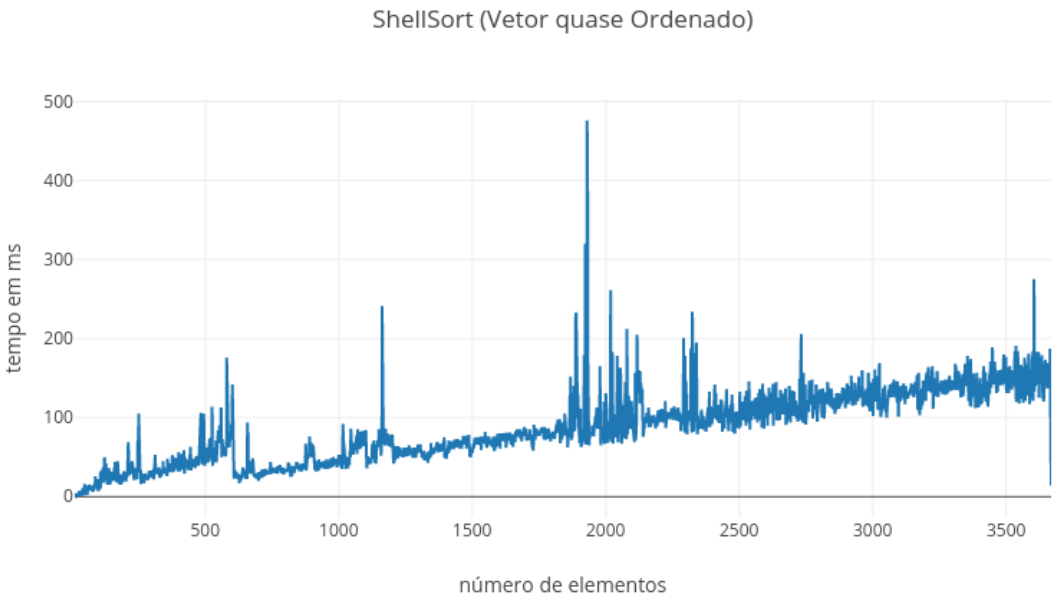

Selection Sort (Vetor sem Repetição)
tempo em ms
400
300
200
100
0
500
1000
1500
2000
número de elementos

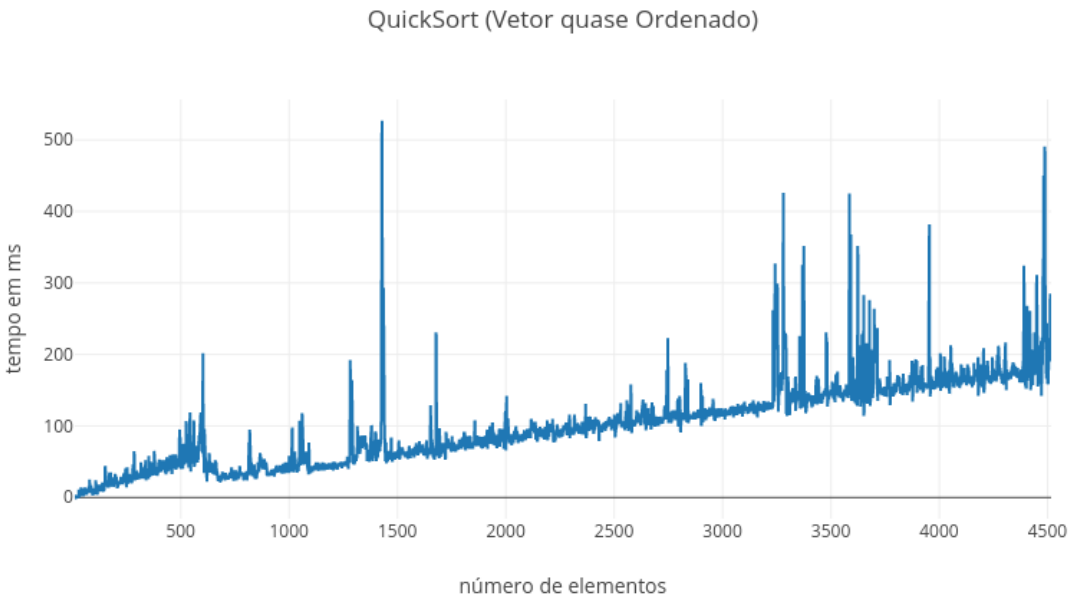

MergeSort (Vetor sem Repetição)
tempo em ms
0
10
20
30
40
50
50
100
150
200
250
300
350
400
450
número de elementos

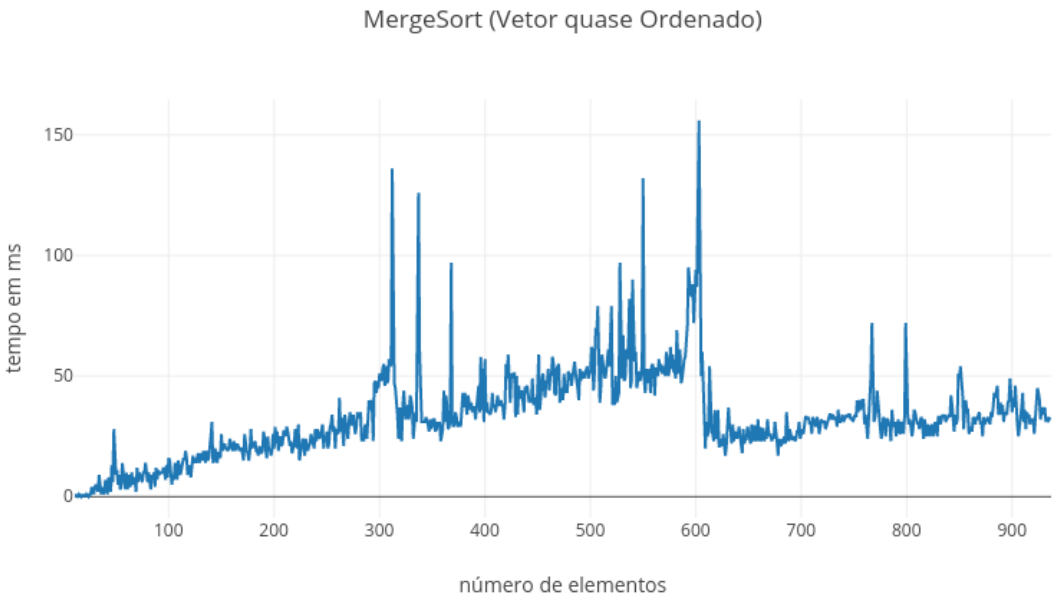

HeapSort (Vetor Sem Repetição)
tempo em ms
0
100
200
300
400
500
1000
1500
2000
número de elementos

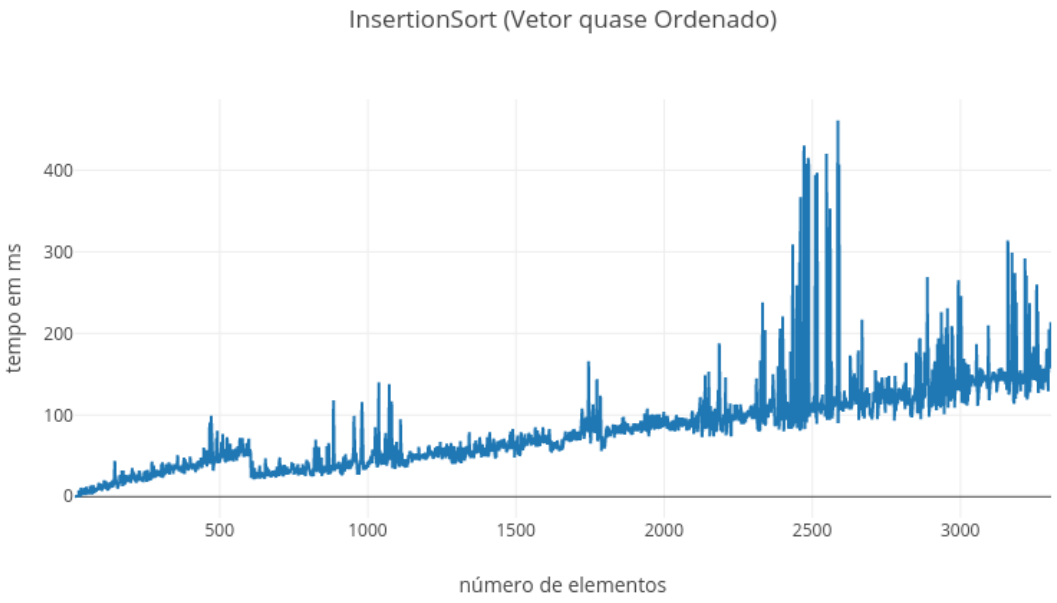

SelectionSort (Vetor quase Ordenado)
tempo em ms
1000
800
600
400
200
0
500
1000
1500
2000
2500
3000
número de elementos

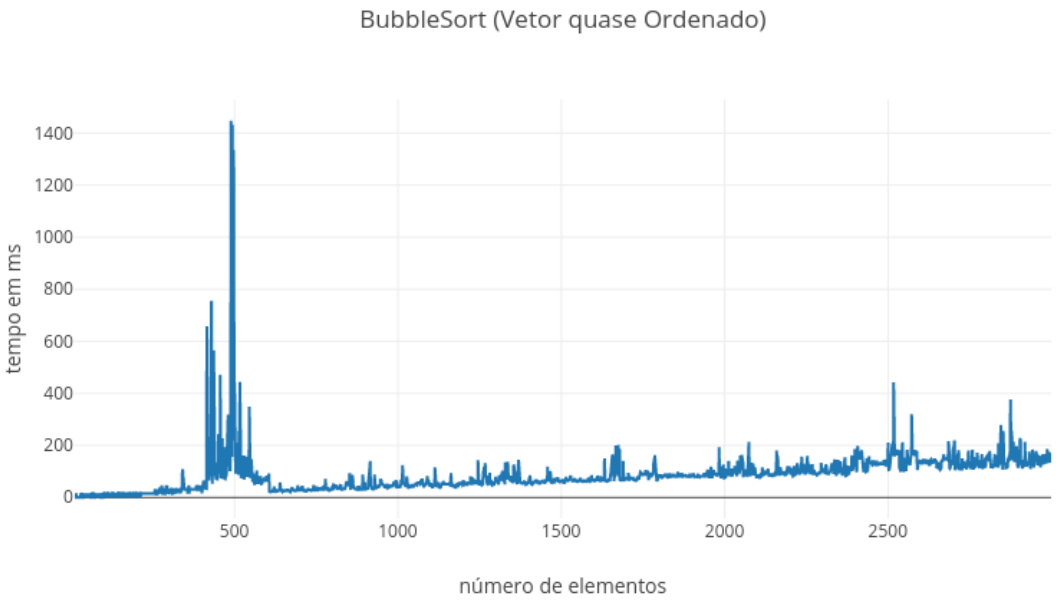

HeapSort (Vetor quase Ordenado)
tempo em ms
500
400
300
200
100
0
500
1000
1500
2000
2500
3000
número de elementos


ShellSort (Vetor quase Ordenado)
tempo em ms
0
100
200
300
400
500
500
1000
1500
2000
2500
3000
3500
número de elementos


QuickSort (Vetor quase Ordenado)
tempo em ms
0
100
200
300
400
500
500
1000
1500
2000
2500
3000
3500
4000
4500
número de elementos


MergeSort (Vetor quase Ordenado)
tempo em ms
150
100
50
0
100
200
300
400
500
600
700
800
900
número de elementos


InsertionSort (Vetor quase Ordenado)
tempo em ms
0
100
200
300
400
500
1000
1500
2000
2500
3000
número de elementos


BubbleSort (Vetor quase Ordenado)
tempo em ms
0
200
400
600
800
1000
1200
1400
500
1000
1500
2000
2500
número de elementos

/**

LAED1 - Trabalho Pratico 2
Aluno: Igor Miranda Oliveira
Matricula: 201312040080
Data: 10/11/2017

**/

Neste trabalho, foram implementados algoritmos não-eficientes (BubbleSort, InsertionSort e SelectionSort) e eficientes (QuickSort, HeapSort, ShellSort e MergeSort) para ordenar sequências de elementos. A linguagem C foi utilizada em todos os algoritmos, o sistema operacional utilizado foi Windows, em um dispositivo com um processador Intel i5-7200u.

Todos os elementos das sequências correspondem a valores inteiros e diferentes sequências iniciais: ordenadas (crescentemente), inversamente ordenadas (decrescentemente), quase ordenadas (metade do vetor foi ordenada) e aleatórias (para cada execução de um algoritmo). Também foram analisadas sequências sem repetições. Os resultados (número de itens e tempo de execução) foram escritos em arquivos csv (Comma separated values) e exportados para gráficos, que podem ser visualizados nos arquivos em pdf anexados.

A biblioteca geraVetor.h foi criada para gerar diferentes tipos de vetores, que foram utilizados pelos demais algoritmos. As implementações do livro texto da disciplina ordenaram os vetores criados.

Análise de Complexidade

Algoritmo	Complexidade
Bubblesort	$O(n^2)$
Inserção	$O(n^2)$
Seleção	$O(n^2)$
Heapsort	$O(n\ logn)$
Mergesort	$O(n\ logn)$
Quicksort	$O(n\ logn)$
Shellsort	$O(n\ logn)$

Análise dos algoritmos não-eficientes (BubbleSort, InsertionSort e SelectionSort)

Em todos testes, exceto o de vetores sem elementos repetidos, o InsertionSort foi mais constante, sendo o mais eficiente de todos, seguido pelo SelectionSort e por fim o BubbleSort. O InsertionSort apresentou uma queda no

tempo de execução para sequências aleatórias com mais de 600 elementos. O BubbleSort apresentou um comportamento crescente.

Em testes com vetores sem elementos repetidos, o mais eficiente foi o SelectionSort, seguido pelo InsertionSort e por fim o BubbleSort. O SelectionSort manteve-se abaixo de 100ms exceto para n próximo de 1700. O BubbleSort apresentou um comportamento crescente.

Análise dos algoritmos eficientes (QuickSort, HeapSort, ShellSort e MergeSort)

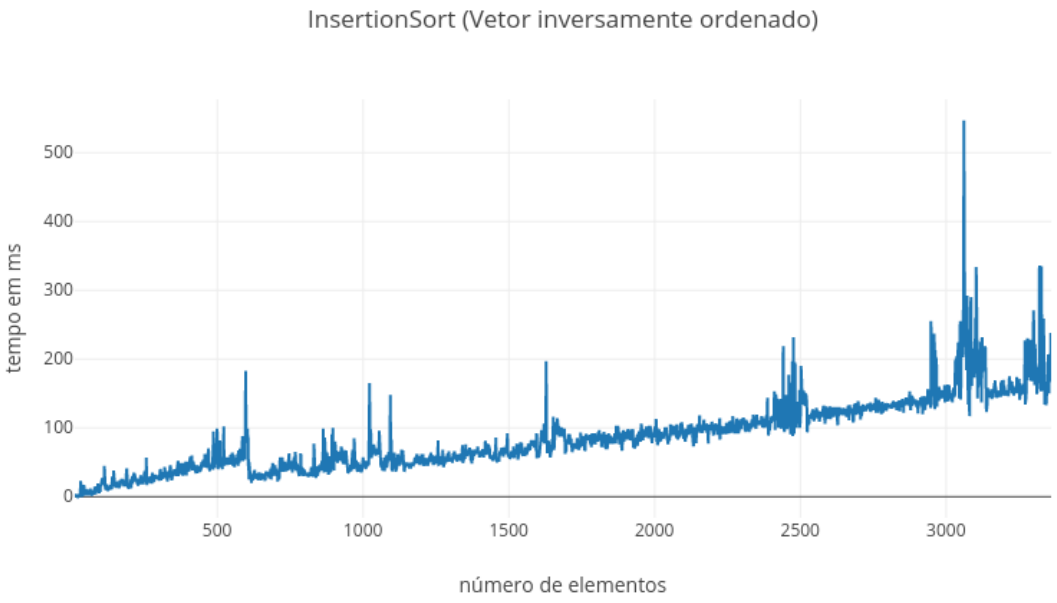
Nos testes com vetores aleatórios e sem repetição os algoritmos com melhor resultado foram o MergeSort seguido pelo QuickSort e o HeapSort, que apresentaram comportamentos similares e por fim o ShellSort, que apresentou tempos de execução dispersos.

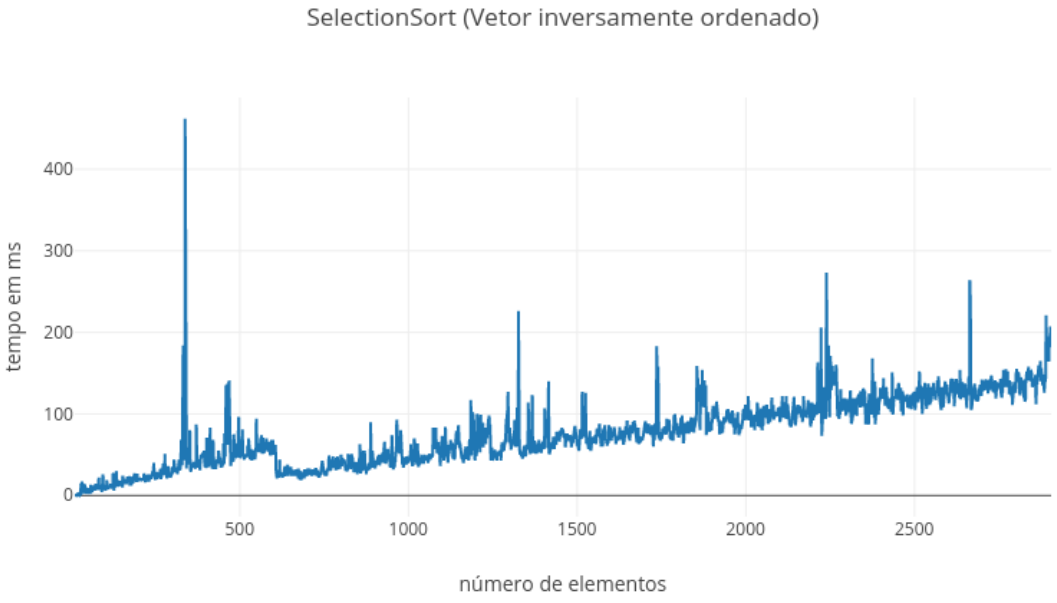
Em testes com vetores ordenados, quase ordenados e inversamente ordenados os algoritmos tiveram desempenho similar, entretanto o que performou melhor foi o QuickSort e por fim o HeapSort.

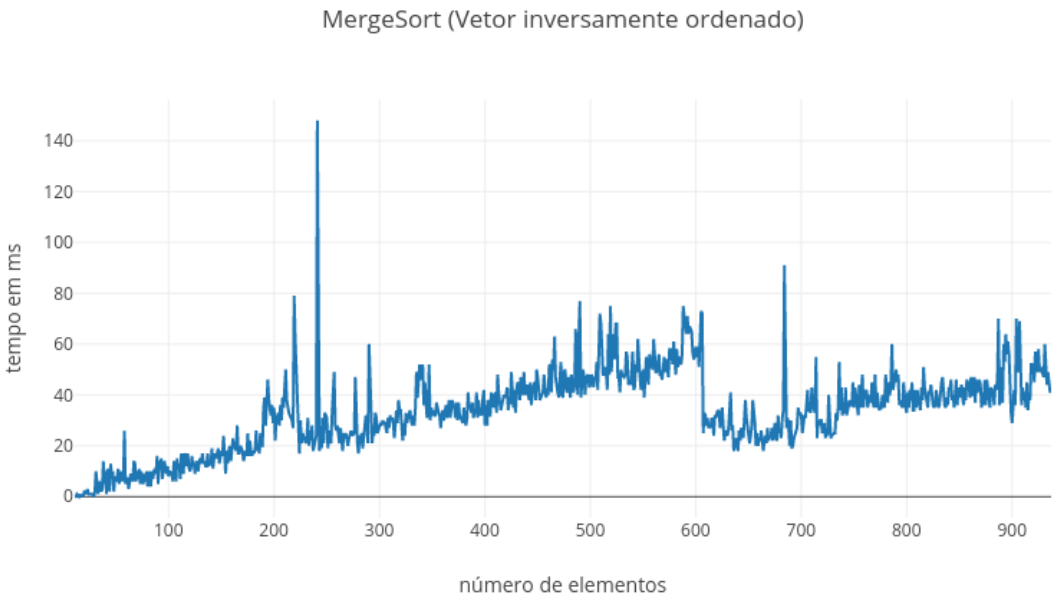
Referências

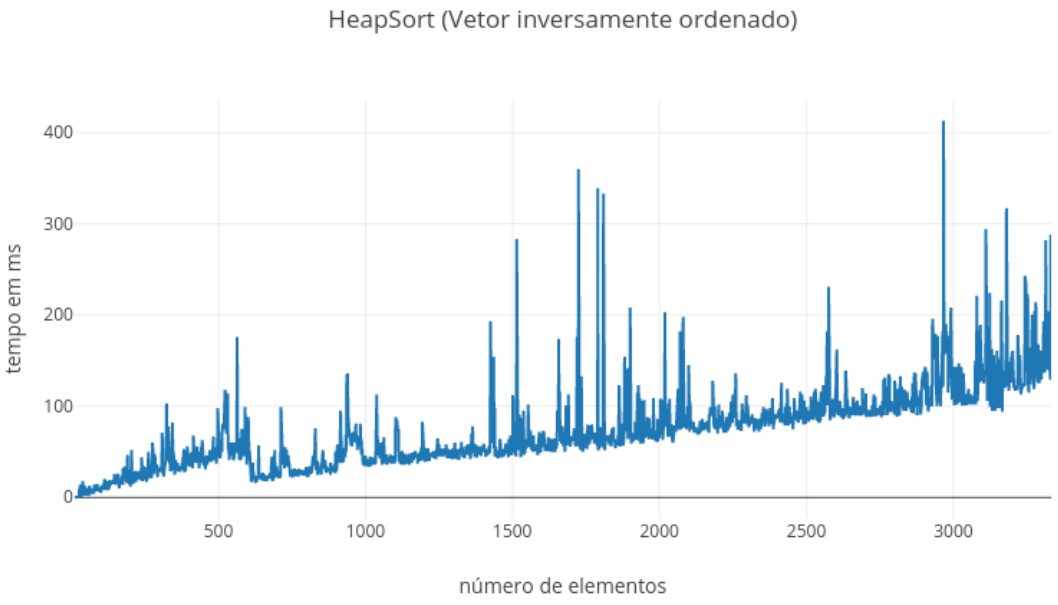
- Ziviani, N. Projeto de algoritmos: com implementações em Java e C++. 3 ed. Editora Cengage Learning, 2007.
- Loureiro, A. A. F. Projeto e Análise de Algoritmos: Análise de Complexidade. Notas de aula, 2010.

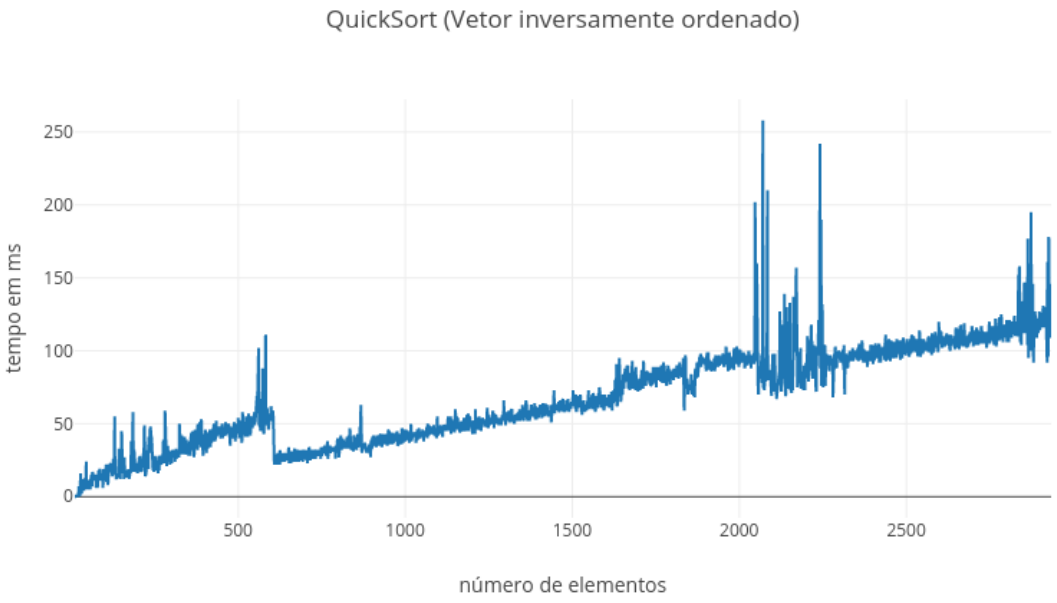
- https://stackoverflow.com/questions/5064379/generating-unique-random-numbers-in-c
- http://www2.dcc.ufmg.br/livros/algoritmos/implementacoes.php
- https://linustechtips.com/main/topic/856289-analysing-different-sorting-algorithms/

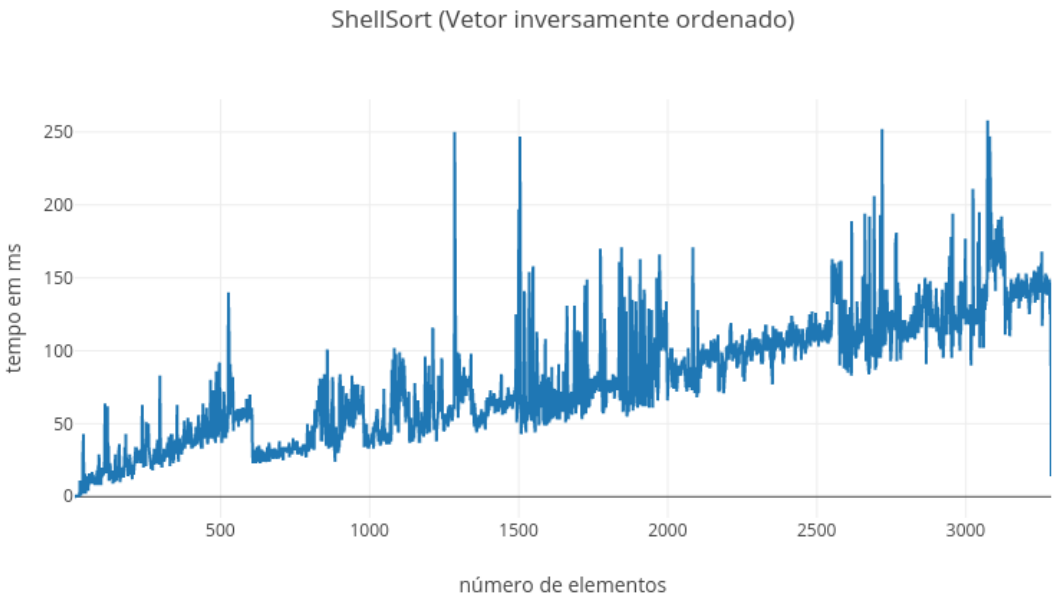

InsertionSort (Vetor inversamente ordenado)
tempo em ms
0
100
200
300
400
500
500
1000
1500
2000
2500
3000
número de elementos

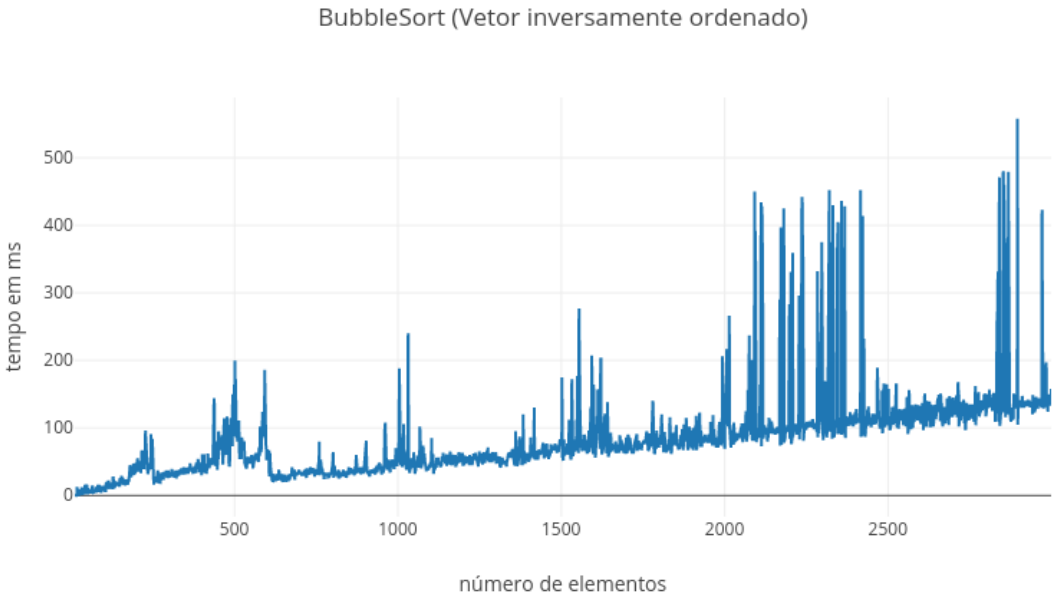

SelectionSort (Vetor inversamente ordenado)
tempo em ms
0
100
200
300
400
500
1000
1500
2000
2500
número de elementos


MergeSort (Vetor inversamente ordenado)
tempo em ms
0
20
40
60
80
100
120
140
100
200
300
400
500
600
700
800
900
número de elementos


HeapSort (Vetor inversamente ordenado)
tempo em ms
0
100
200
300
400
500
1000
1500
2000
2500
3000
número de elementos


QuickSort (Vetor inversamente ordenado)
tempo em ms
250
200
150
100
50
0
500
1000
1500
2000
2500
número de elementos


ShellSort (Vetor inversamente ordenado)
tempo em ms
0
50
100
150
200
250
500
1000
1500
2000
2500
3000
número de elementos


BubbleSort (Vetor inversamente ordenado)
tempo em ms
0
100
200
300
400
500
500
1000
1500
2000
2500
número de elementos