Universidade Federal do Tocantins
Campus Universitário de Palmas
Ciências da computação

DAVID XAVIER BRITO

A análise empírica - Algoritmos de ordenação

PALMAS-TO
2019

DAVID XAVIER BRITO

Universidade Federal do Tocantins - UFT

Este trabalho tem como objetivo apresenta a análise empírica **(Pratica**) também chamada de analise de desempenho dos principais algoritmos de ordenação.

Orientador: Warley

PALMAS-TO
2019

Resumo

Para evidenciar o desempenho de alguns algoritmos de ordenação foi necessário a realização de uma análise empírica, também conhecida como análise de desempenho ou bateria de teste. O intuito dessa pesquisa e estudar a variação das estruturas desses algoritmos baseando-se nas entradas sendo elas ascendente, descente e aleatória, verificando na pratica o comportamento de um algoritmo. O resultado dessa pesquisa consiste em coleta informações de um conjunto de elementos número de comparações, número de movimentações. A linguagem programação utilizada foi a Python, uma linguagem interpretada, imperativa, orientada a objetos de fácil entendimento. Todos os códigos utilizados para esse estudo serão disponibilizados no GitHub.

Baterias de testes de algoritmos de ordenação

Para desenvolvimento da análise foi seguidos alguns passos, inicialmente foi realizada e implementação dos algoritmos de ordenação. Em seguida para a foi criado registros de chaves grandes, para realização dos testes, para isso foi implementado uma função para gerar as várias chaves de valores randômicos para serem inseridas no vetor no formato **csv**, um tipo especial de arquivo que pode ser criado e editado no Excel. Após essas ações, foi coletado o tempo gasto para ordenar conjuntos com números crescentes de elementos, por exemplo 1000, 5000, 10000, 20000, 50000 e 10000. Para conseguir os dados foram utilizados comandos da biblioteca multiplataforma capaz de recuperar informações sobre processos em execução é utilização do sistema (CPU, discos, etc.) conhecida como **psutil.** Também foi medido o tempo para ordenar estruturas já ordenas, inversamente ordenadas e aleatoriamente ordenas. Para cada caso-teste foi traçado gráficos do tipo *n x t* (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação). Também podem ser traçados gráficos do tipo *c x t* e *m x t*, relativos ao número de comparações na estrutura e número de movimentações realizadas, respectivamente.

CARACTESTICAS DO COMPUTADOR UTILIZADO PARA OS TESTES

Será apresentado as principais características da máquina utilizada para a realização dos testes.

- Marca: **DELL**
- Processador: **Intel(R) Core (TM) i5-7200U CPU @ 2.50 GHZ**
- Sistema Operacional: **Windows 10 Pro**
- Tipo de sistema: **64 bits, processador com base em x64**
- Memória (RAM): **8,00 GB**
- HD: **SSD Kingston 120 GB**

FERRAMENTAS UTILIZADAS

As ferramentas utilizadas foram Pycharm é um ambiente de desenvolvimento integrado usado em programação de computadores, especificamente para a linguagem Python. O Excel é um editor de planilhas produzidos é o Notepad++, um editor de texto e de código fonte, suporta várias linguagens de programação.

Figura 1: Ferramentas para desenvolvimento da pesquisar

BIBLIOTECAS UTILIZADAS

bibliotecas são uma coleção de subprogramas utilizados para compartilhar soluções por meio de funções permitindo que o desenvolvimento seja mais fácil e rápido. A seguir será apresentado as bibliotecas utilizadas para desenvolvimento dos testes.

Time: fornece várias funções relacionadas ao tempo, embora este módulo esteja sempre disponível, nem todas as funções estão disponíveis em todas as plataformas. A maioria das funções definidas neste módulo chama a biblioteca C da plataforma com o mesmo nome. Às vezes, pode ser útil consultar a documentação da plataforma, porque a semântica dessas funções varia entre as plataformas.

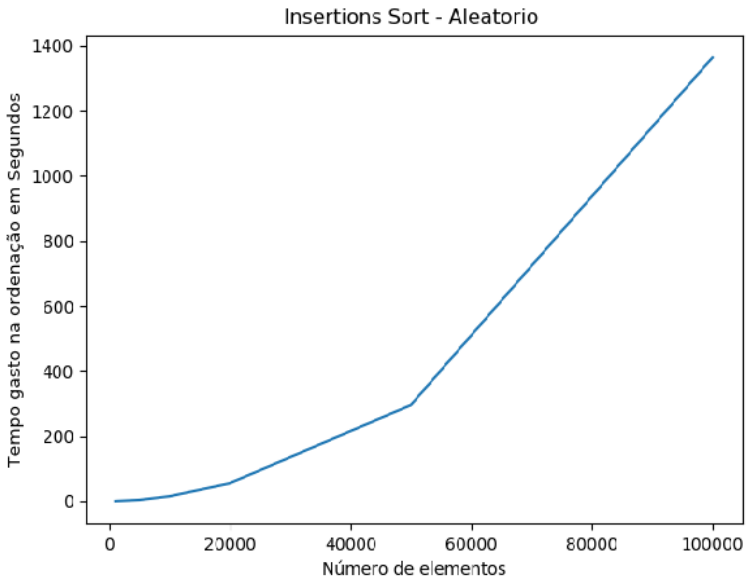
Psutil: (sistema python e utilitários de processo) é uma biblioteca multiplataformas para recuperar informações sobre processos em execução e utilização do sistema (CPU, memória, discos, rede, sensores) em Python. É útil principalmente para monitoramento de sistema, criação de perfil, limitação de recursos de processo e gerenciamento de processos em execução. Ele implementa muitas funcionalidades oferecidas pelas ferramentas de linha de comando do UNIX, tais como: ps, top, lsof, netstat, ifconfig, df, kill, free, nice, iônica, iostat, uptime, pidof, tty, taskset, pmap. O psutil atualmente suporta as seguintes plataformas:

Matplotlib: possibilita pode gerar gráficos, histogramas, espectros de potência, gráficos de barras, gráficos de erros, diagramas de dispersão, etc., com apenas algumas linhas de código. Por exemplo, veja os gráficos de amostra e a galeria de miniaturas.

Insertion Sort

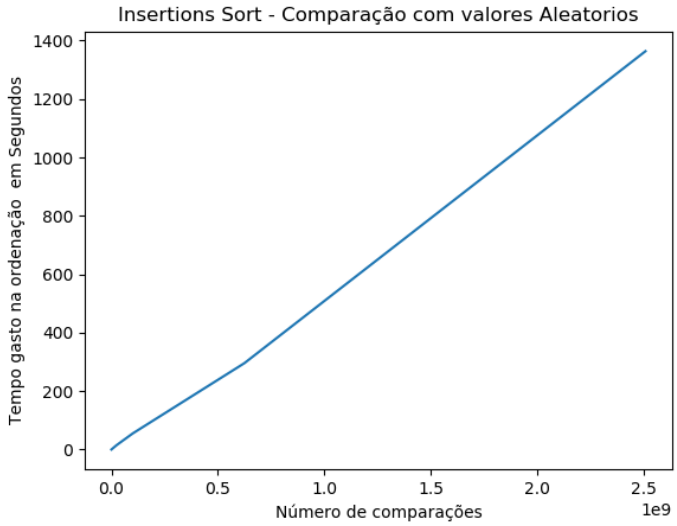
I**nsertion Sort**, ou ordenação por inserção, é o algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação. A seguir será apresentado gráficos com os resultados das pesquisar com suas respectivas entradas.

Figura 2: insertion sort com vetor de valores aleatórios tipo *n x t*



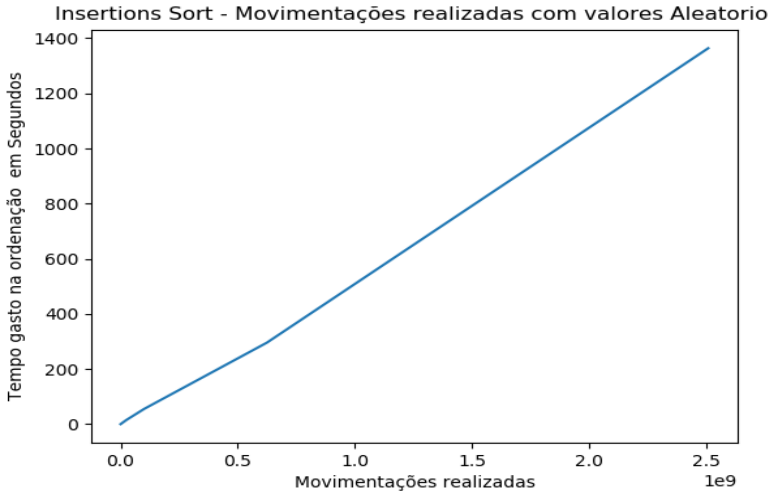
A figura 2 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico apresenta uma relação do tipo *n x t* (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura3: insertion sort comparação com vetor de valores aleatórios do tipo *c x t*



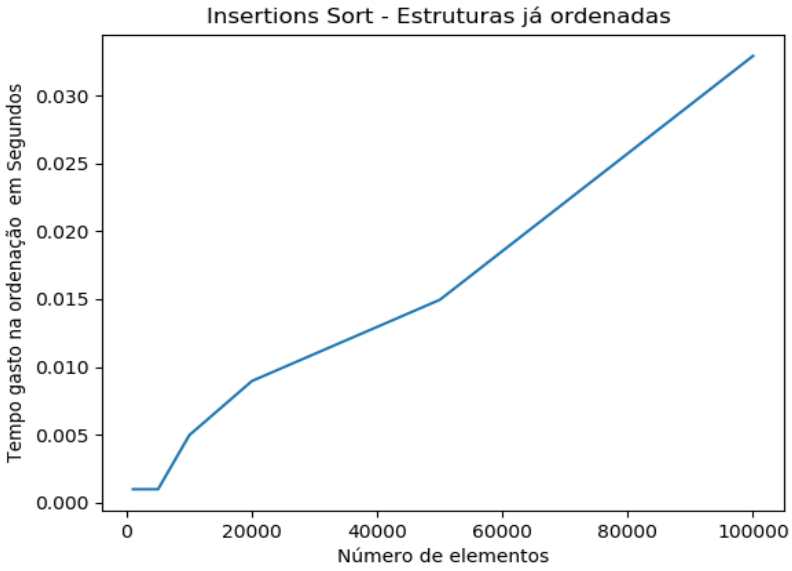
A figura 3 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo *c x t,* foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 4: insertion sort movimentações realizadas com vetor de valores aleatórios tipo m x t



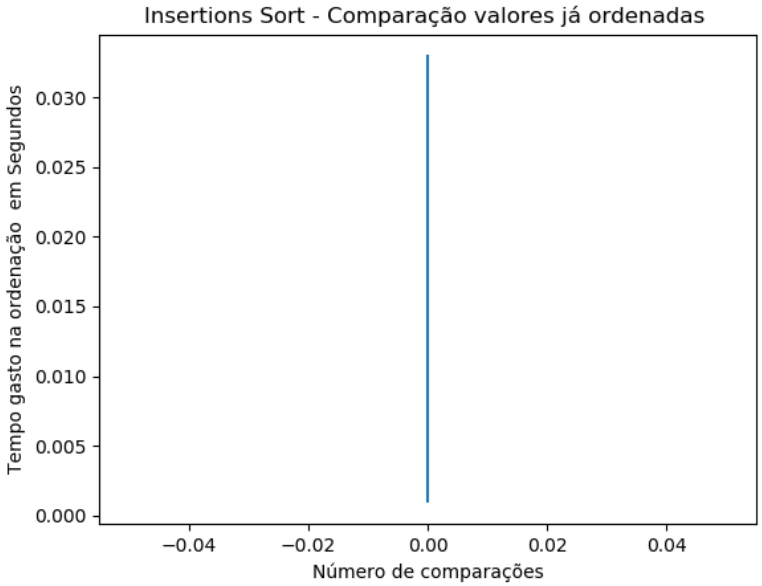
A figura 4 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo *m x t,* foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 5: insertion sort com vetor de valores já ordenados *n x t*



A figura 5 apresenta o gráfico gerado com vetores já ordenados. O gráfico apresenta uma relação do tipo *n x t* (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 6: insertion sort número de comparação com vetor de valores já ordenados do tipo *c x t*

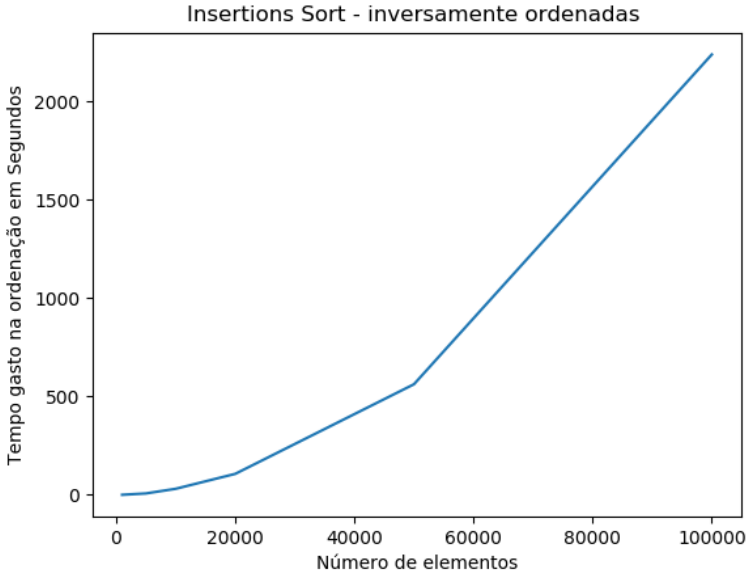


A figura 6 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 7: insertion sort movimentações realizadas com vetor de valores já ordenados do tipo *m x t*

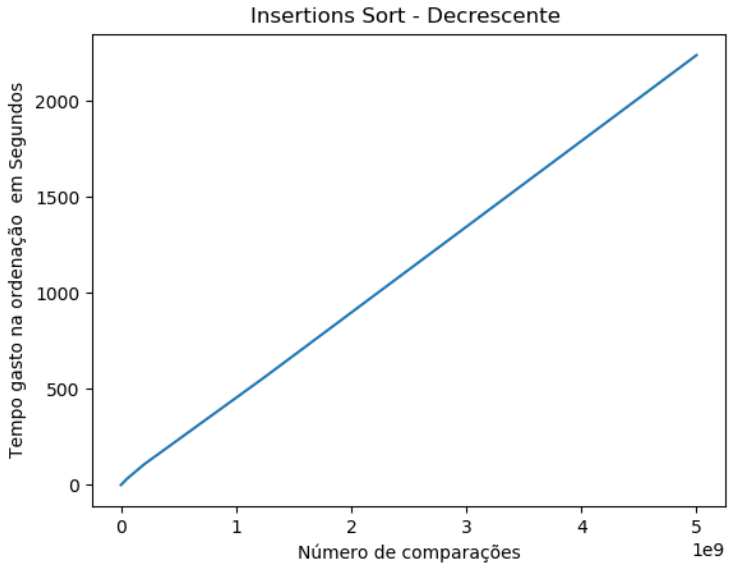
A figura 7 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo *m x t*, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 8: insertion sort com vetor de valores inversamente ordenadas *n x t*



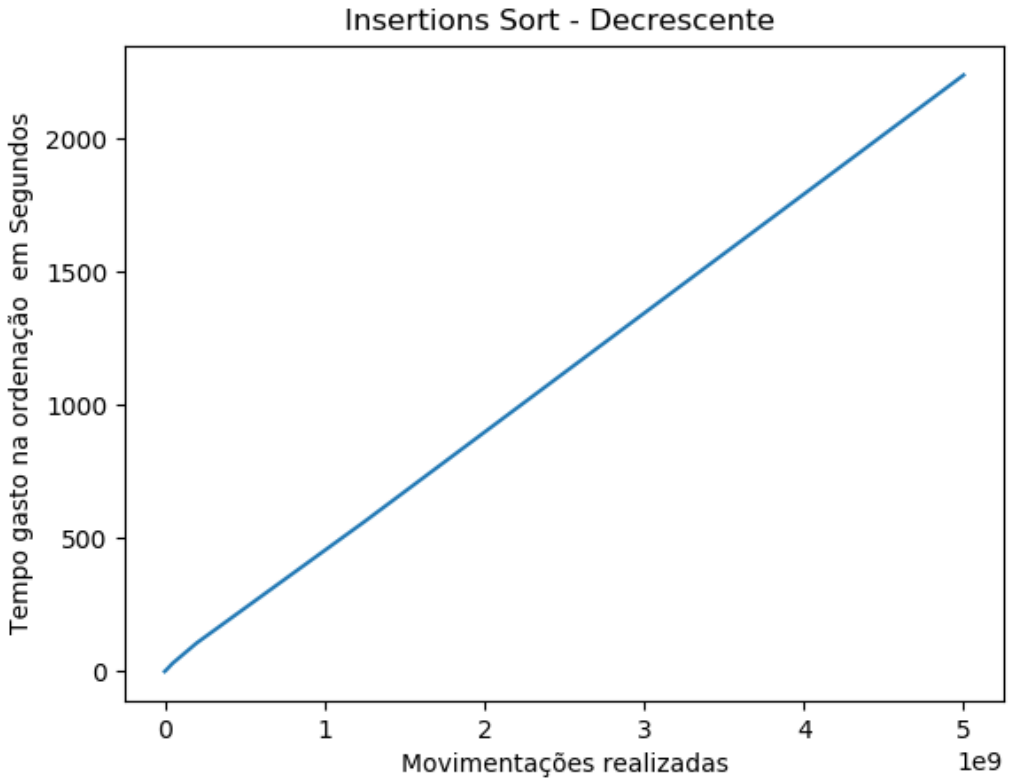
A figura 8 apresenta o gráfico gerado com vetores de valores inversamente ordenadas. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 9: insertion sort número de comparação com vetor de valores inversamente ordenadas do tipo *c x t*



A figura 9 apresenta o gráfico gerado com vetores de valores inversamente ordenadas. O gráfico foi do tipo *c x t*, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

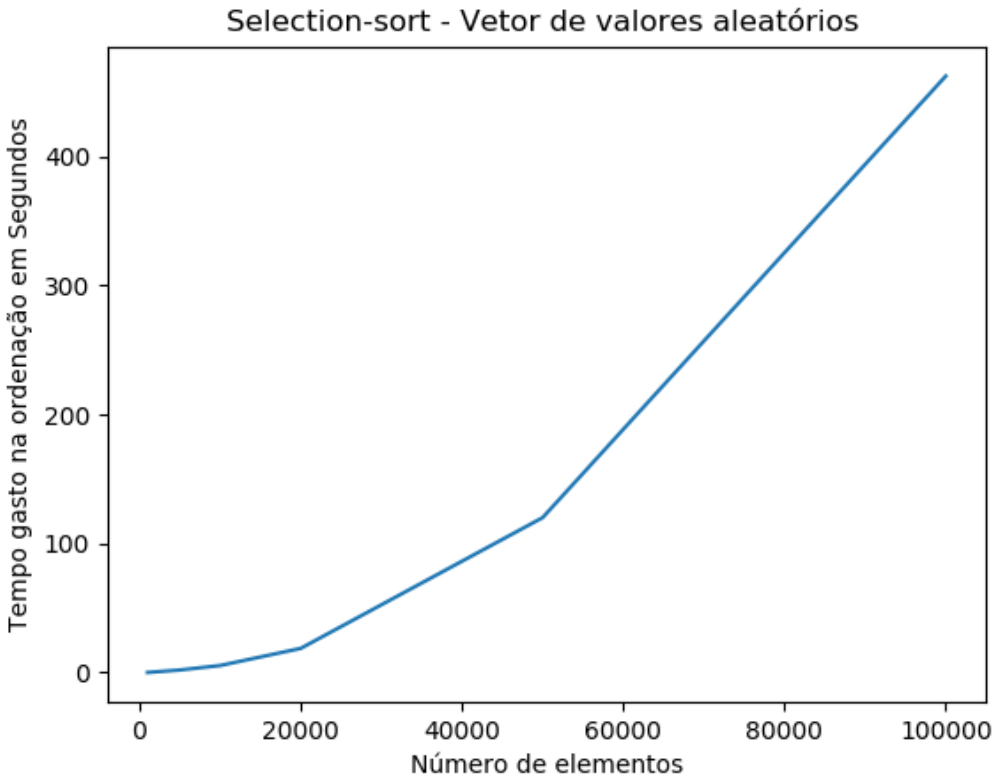
Figura 10: Movimentações realizadas com vetor de valores inversamente ordenadas do tipo *m x t*



Selection Sort

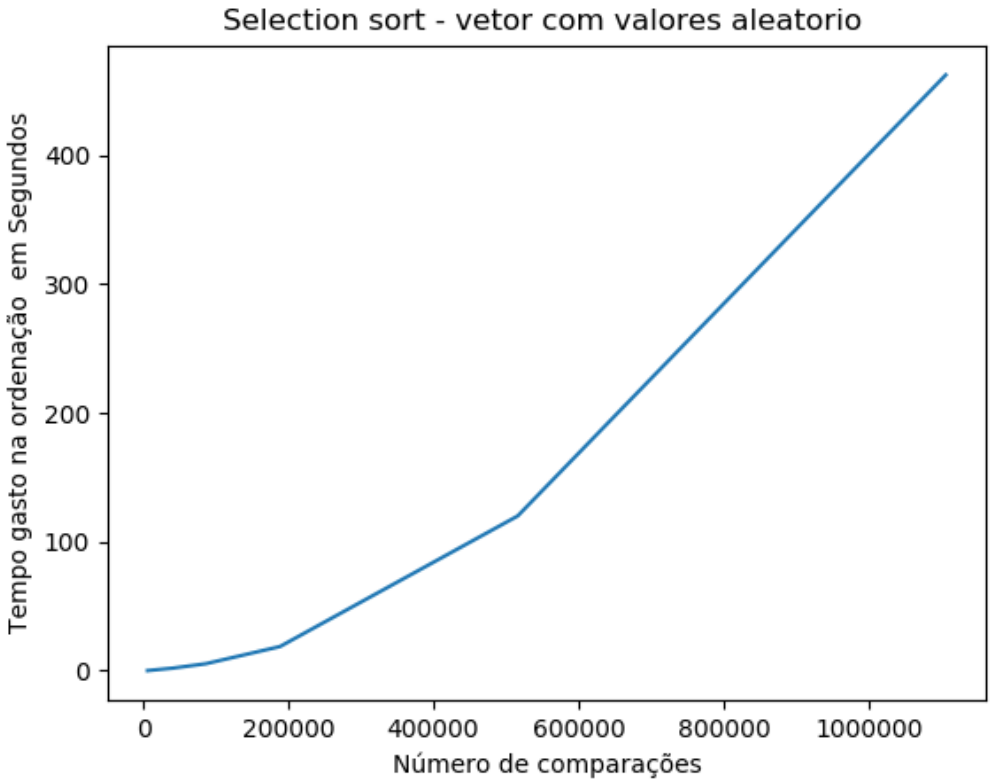
Selection sort é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os **n-1** elementos restantes, até os últimos dois elementos.

Figura 11: selection sort com vetor de valores aleatórios tipo n x t



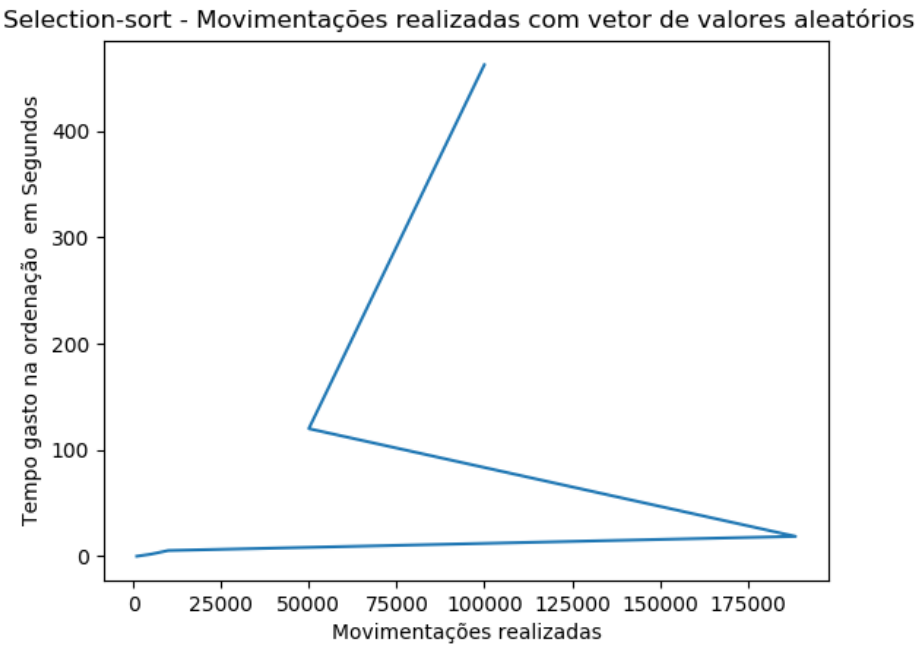
A figura 11 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 12: selection sort comparação com vetor de valores aleatórios do tipo c x t



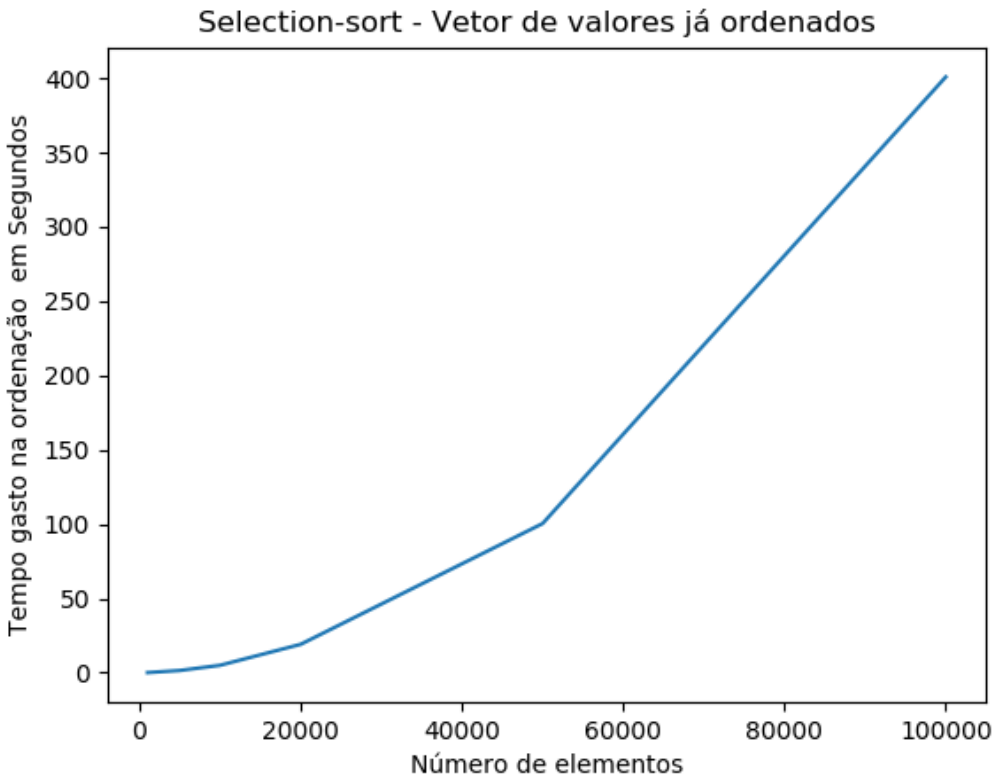
A figura 12 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 13: selection sort movimentações realizadas com vetor de valores aleatórios tipo m x t



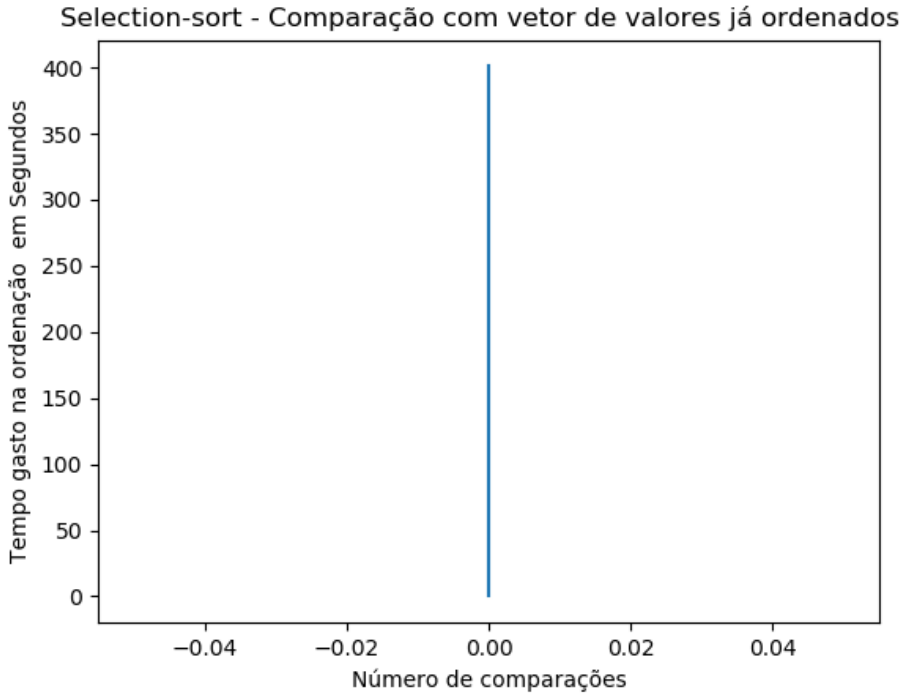
A figura 13 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 14: selection sort com vetor de valores já ordenados n x t



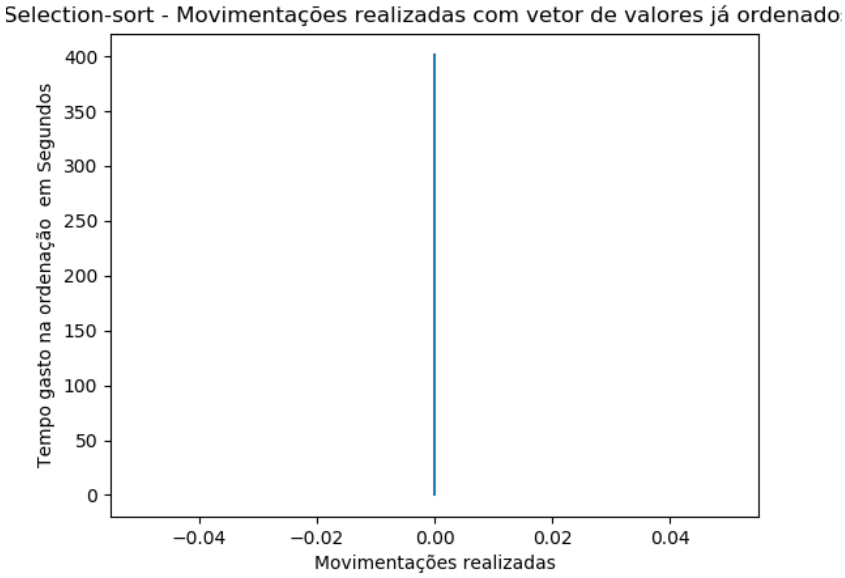
A figura 14 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 15: selection sort número de comparação com vetor de valores já ordenados do tipo c x t



A figura 15 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 16: selection sort movimentações realizadas com vetor de valores já ordenados do tipo m x t

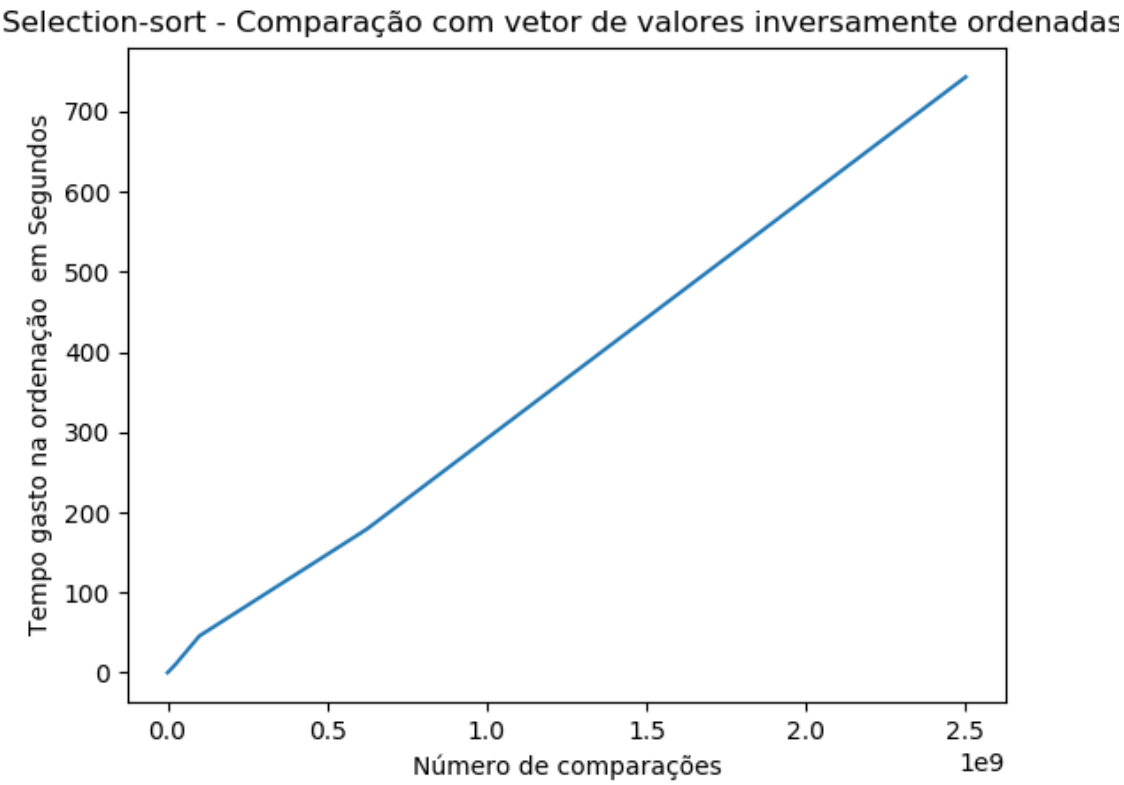


A figura 16 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 17: selection sort com vetor de valores inversamente ordenadas n x t

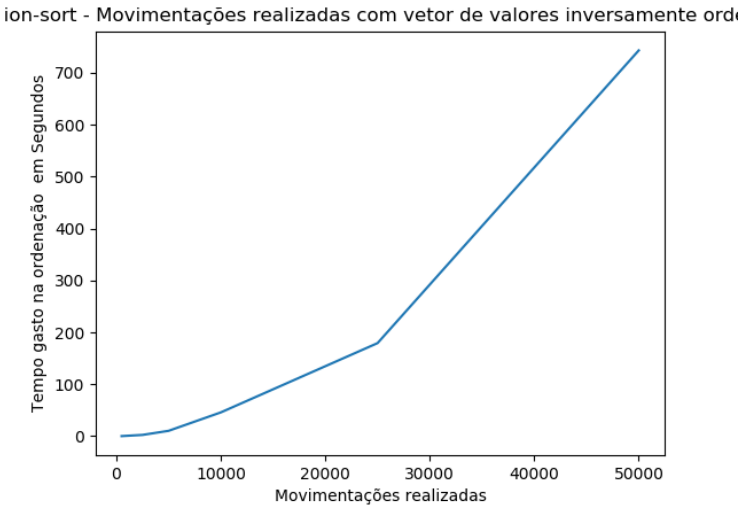
A figura 17 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 18: selection sort número de comparação com vetor de valores inversamente ordenadas do tipo c x t



A figura 18 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 19: selection sort movimentações realizadas com vetor de valores inversamente ordenadas do tipo m x t

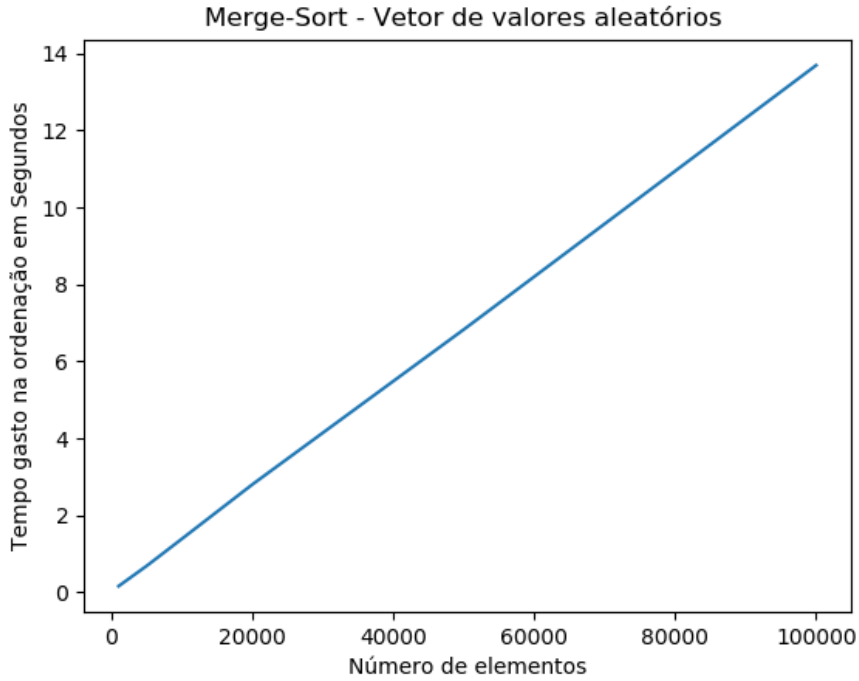


A figura 19 apresenta o gráfico gerado com vetores de valores de valores inversamente ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Merge sort

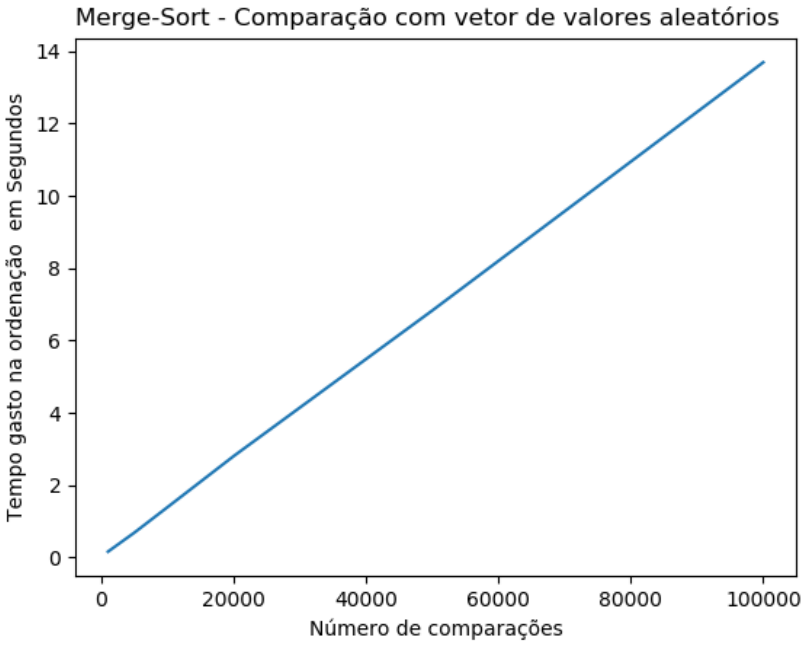
O merge sort, ou ordenação por mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar. Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo Merge Sort usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

Figura 20: merge sort com vetor de valores aleatórios tipo n x t



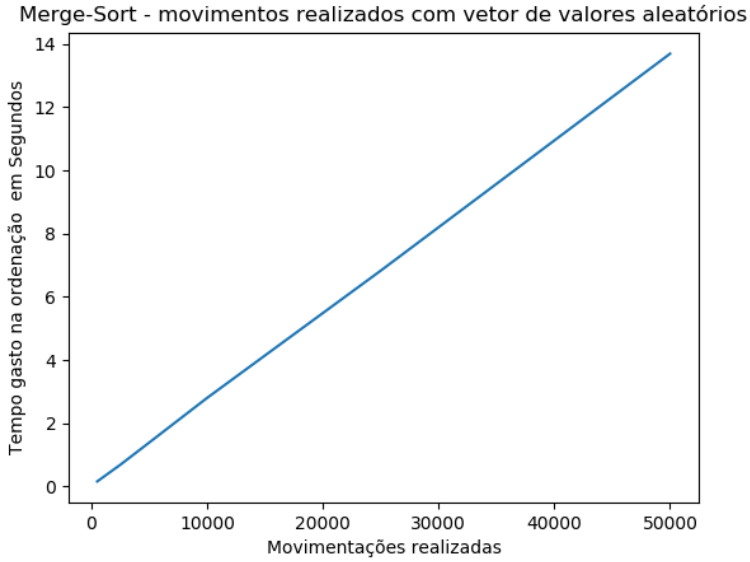
A figura 20 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 21: merge sort comparação com vetor de valores aleatórios do tipo c x t



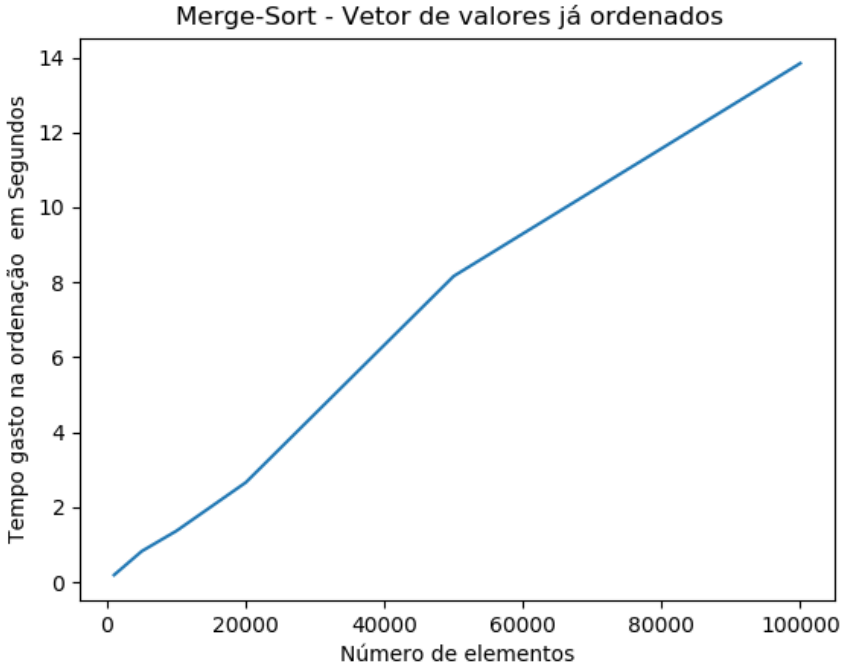
A figura 21 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 22: merge sort realizadas com vetor de valores aleatórios tipo m x t



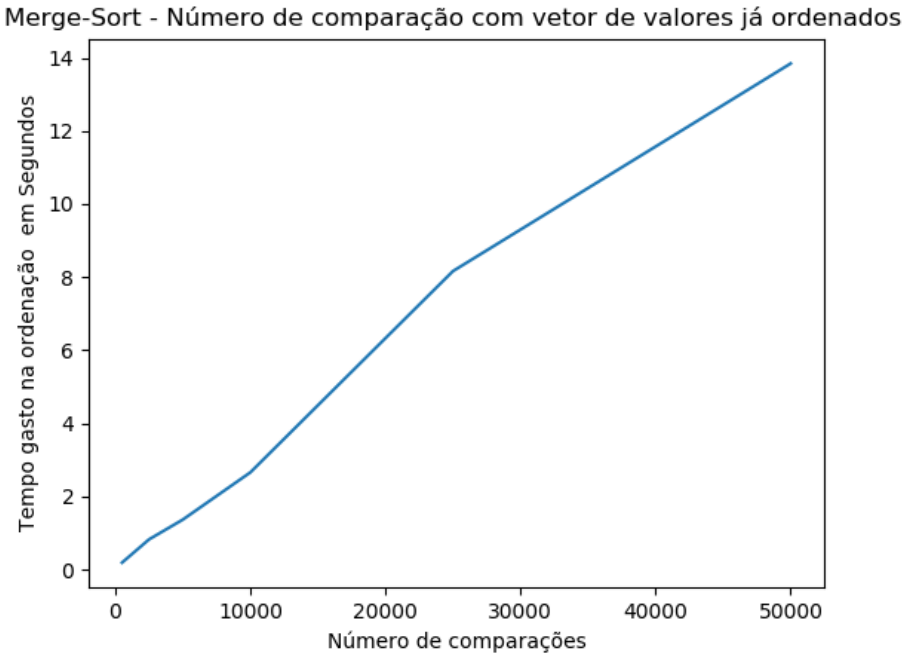
A figura 22 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 23: merge sort com vetor de valores já ordenados n x t



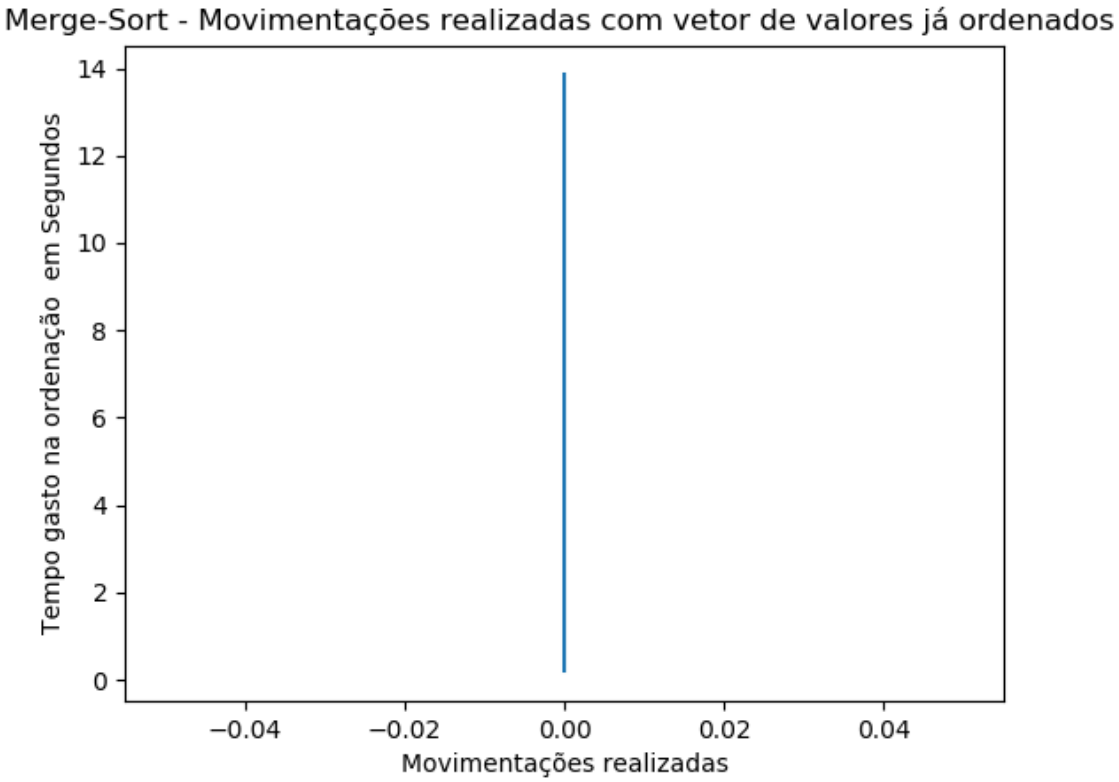
A figura 23 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 24: merge sort número de comparação com vetor de valores já ordenados do tipo c x t



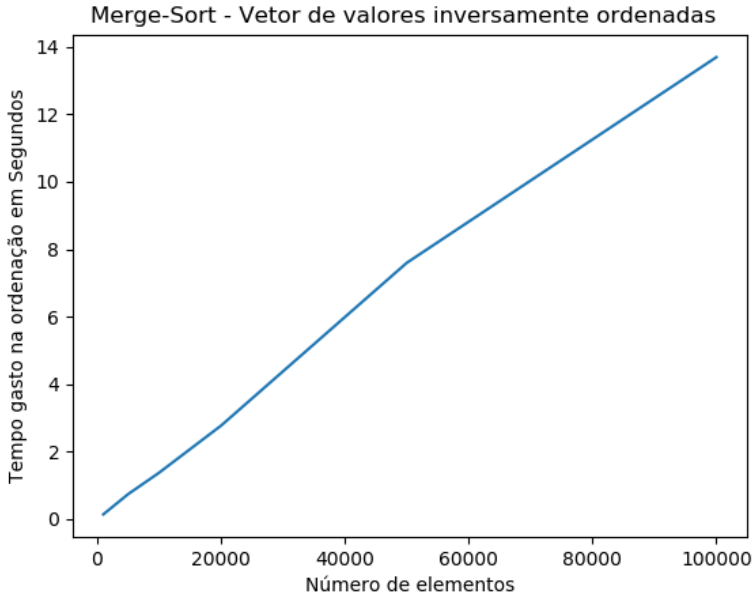
A figura 24 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 25: merge sort movimentações realizadas com vetor de valores já ordenados do tipo m x t



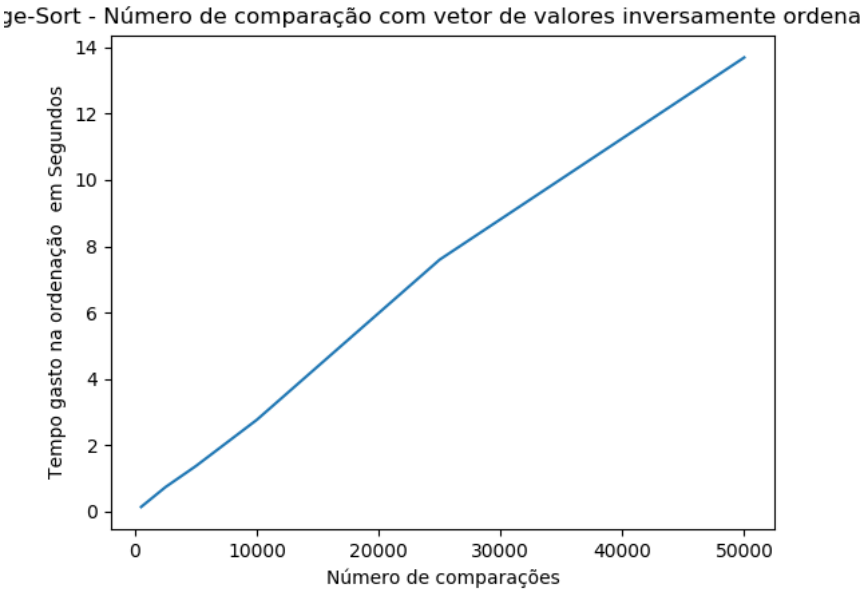
A figura 25 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 26: merge sort com vetor de valores inversamente ordenadas n x t



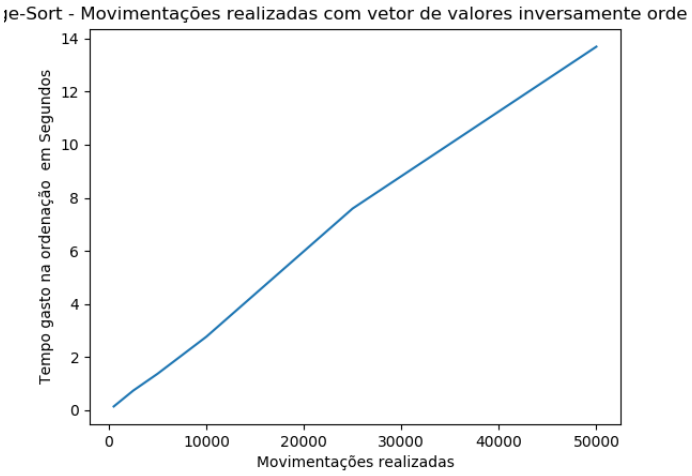
A figura 26 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 27: merge sort número de comparação com vetor de valores inversamente ordenadas do tipo c x t



A figura 27 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 28: merge sort movimentações realizadas com vetor de valores inversamente ordenadas do tipo m x t

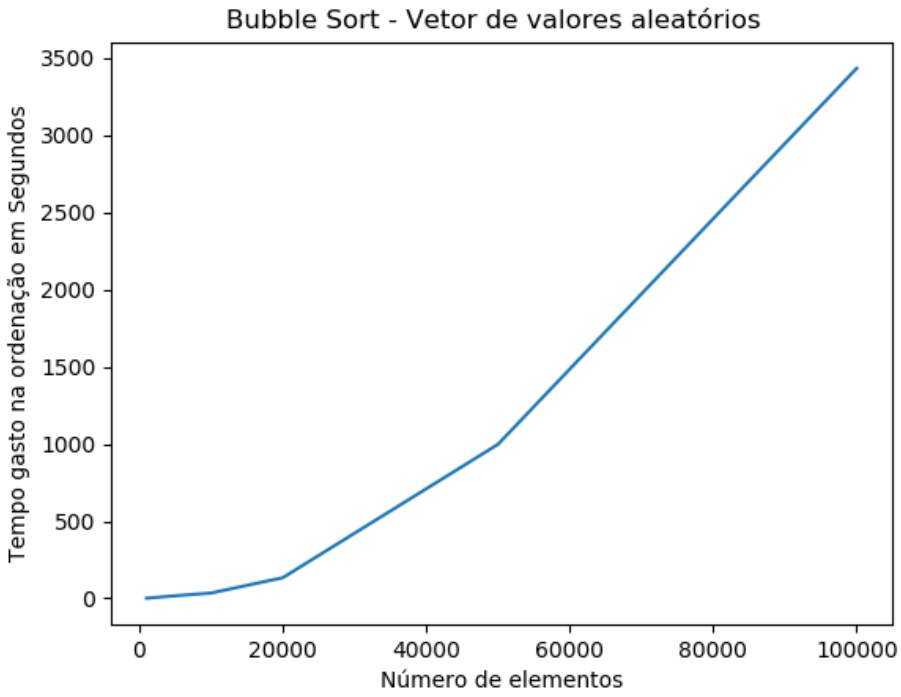


A figura 28 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Bubble Sort

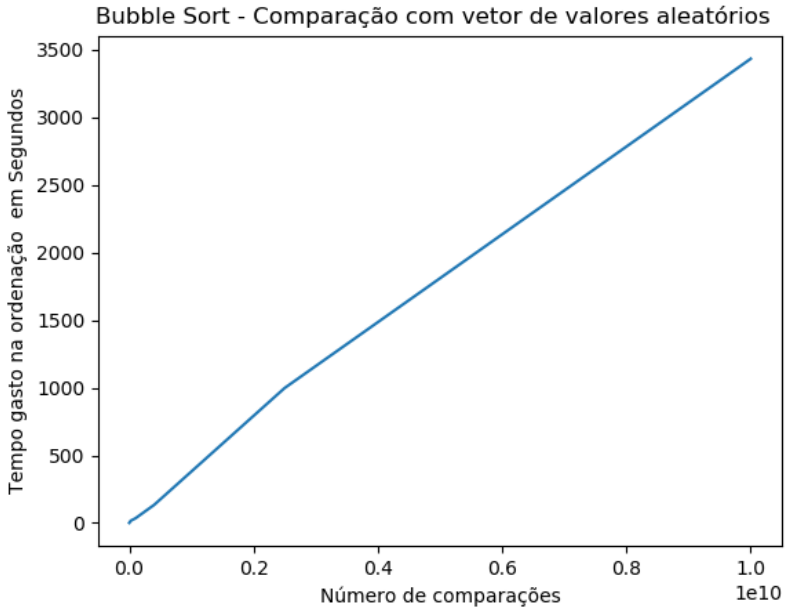
O bubble sort, é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

Figura 29: buble sort com vetor de valores aleatórios tipo n x t



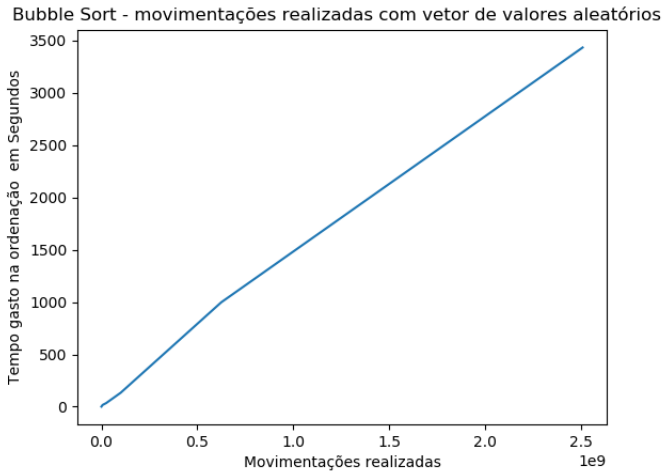
A figura 29 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 30: buble sort comparação com vetor de valores aleatórios do tipo c x t



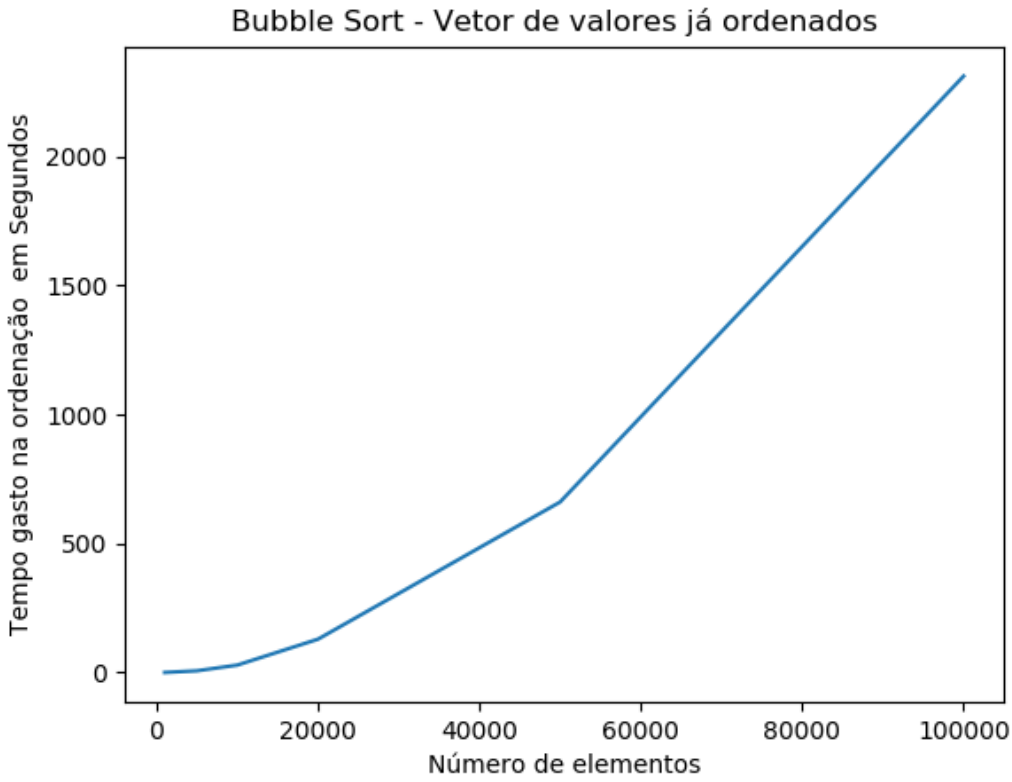
A figura 30 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 31: buble sort realizaas com vetor de valores aleatórios tipo m x t



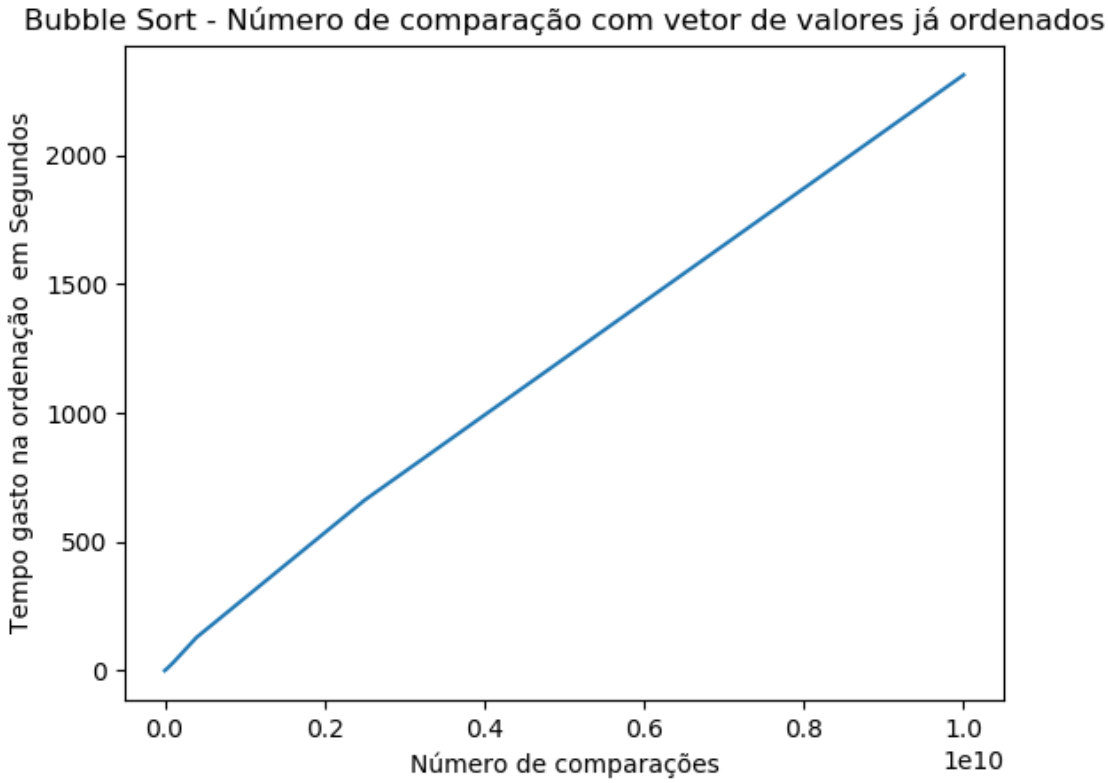
A figura 31 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 32: buble sort com vetor de valores já ordenados n x t



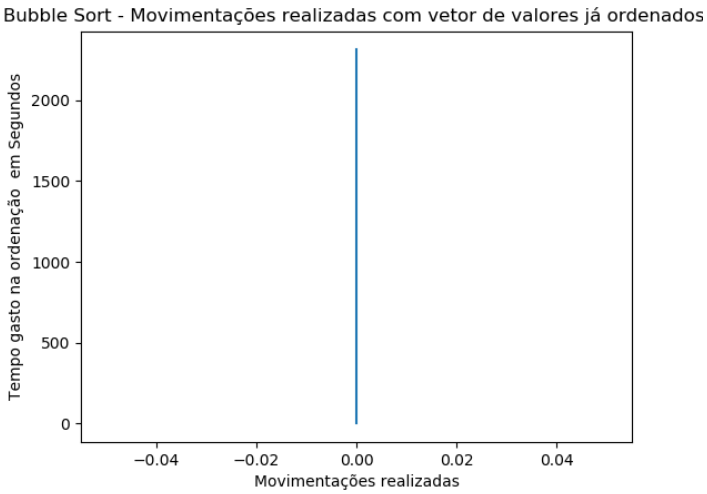
A figura 32 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 33: buble sort número de comparação com vetor de valores já ordenados do tipo c x t



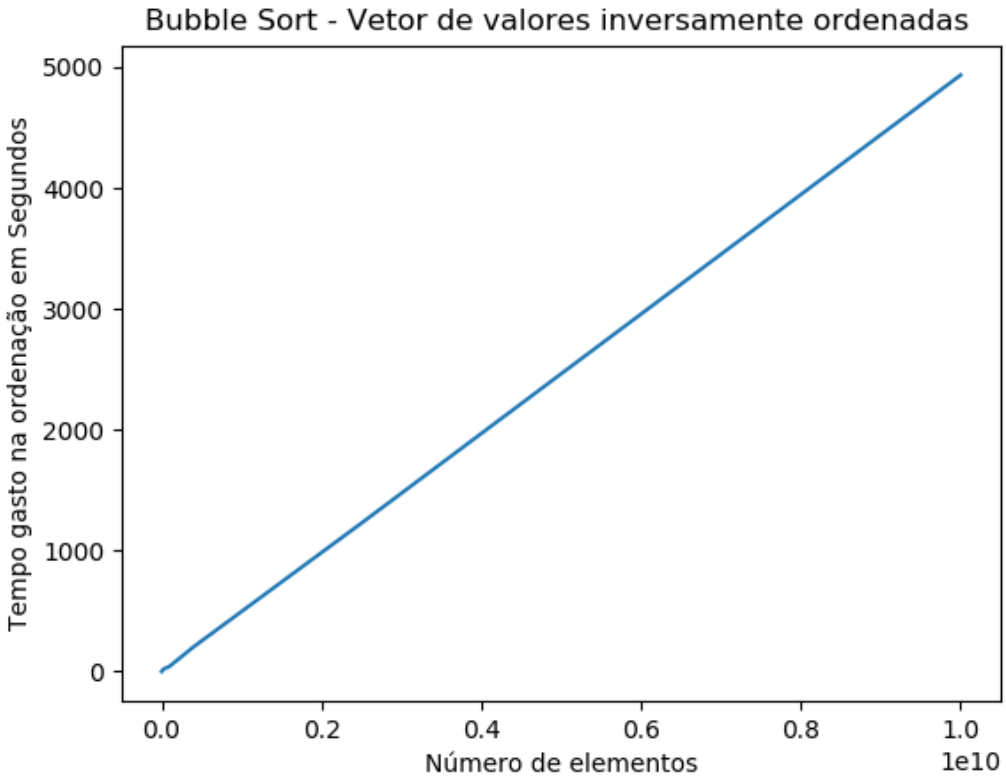
A figura 33 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 34:buble sort movimentações realizadas com vetor de valores já ordenados do tipo m x t



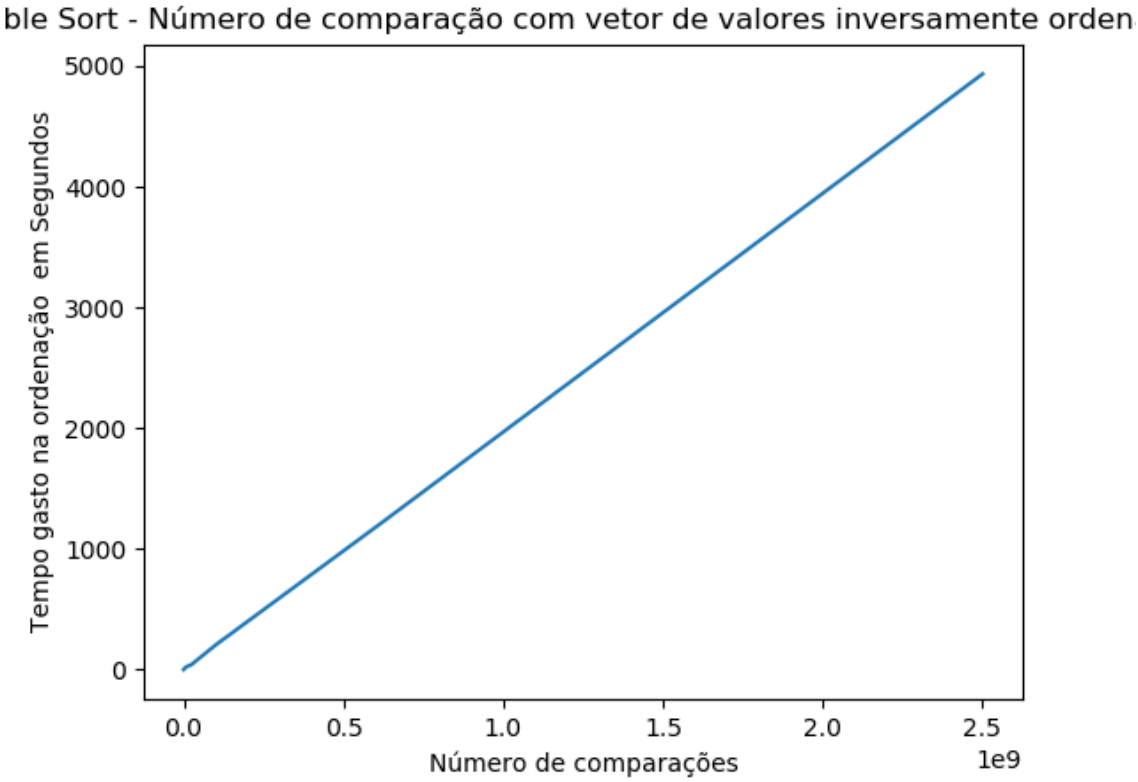
A figura 34 apresenta o gráfico gerado com vetores de valores já ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

Figura 35: buble sort com vetor de valores inversamente ordenadas n x t



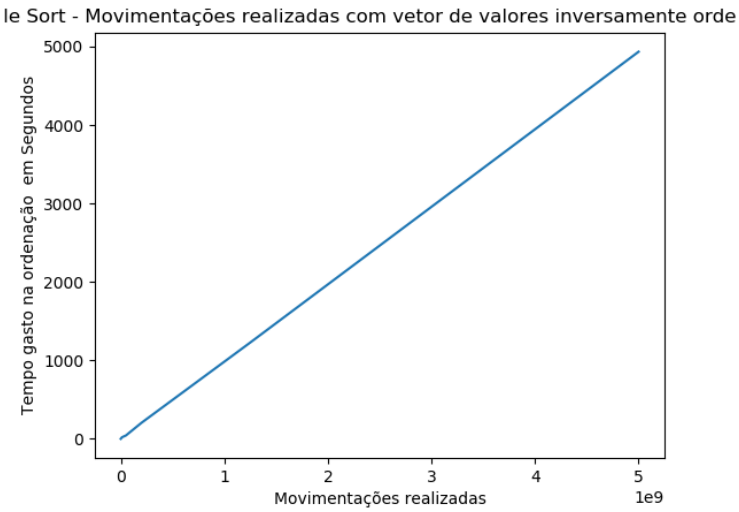
A figura 35 apresenta o gráfico gerado com vetores de valores inversamente ordenadas. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 36: buble sort número de comparação com vetor de valores inversamente ordenadas do tipo c x t



A figura 36 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

Figura 37: buble sort movimentações realizadas com vetor de valores inversamente ordenadas do tipo m x t



A figura 37 apresenta o gráfico gerado com vetores de valores inversamente ordenados. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

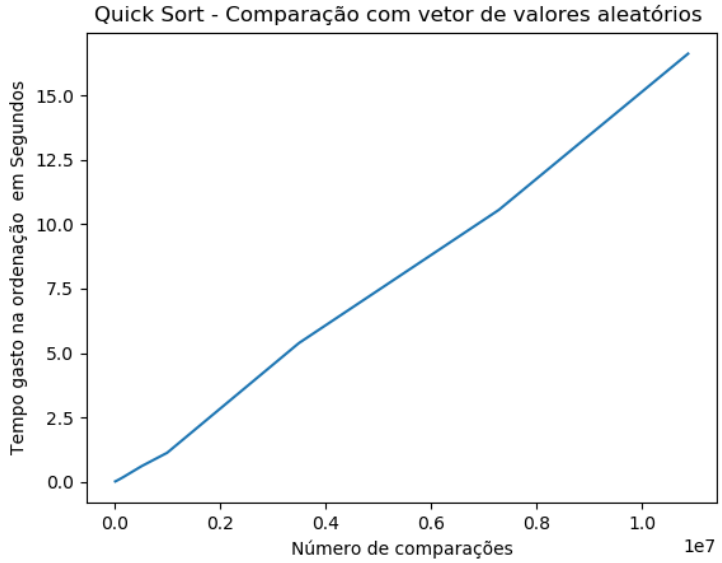
Quick Sort

O quicksort adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o Quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada.

Figura 38: quick sort com vetor de valores aleatórios tipo n x t

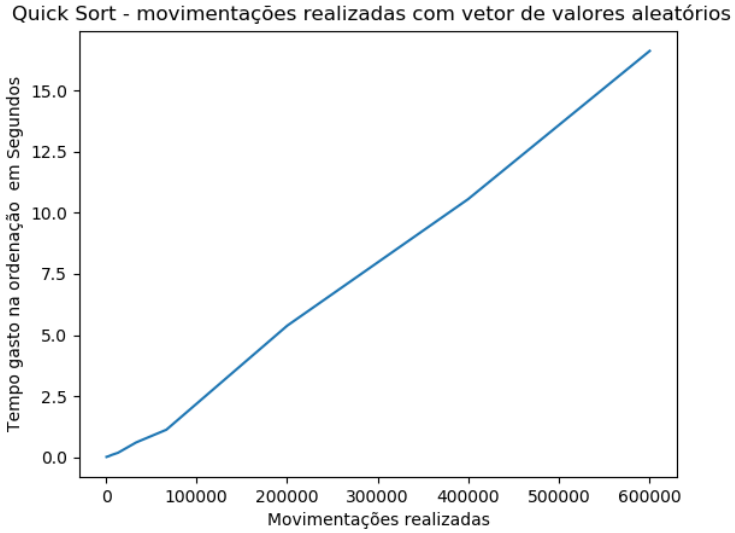
A figura 38 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico apresenta uma relação do tipo n x t (onde n é o número de elementos a ser ordenado e t, o tempo gasto na ordenação).

Figura 39: quick sort comparação com vetor de valores aleatórios do tipo c x t



A figura 39 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo c x t, foi traçado utilizando número de comparações na estrutura do algoritmo e o tempo gasto na ordenação.

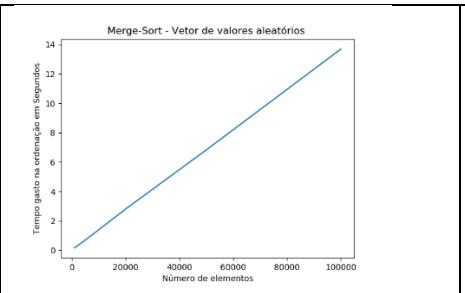
Figura 40: quick sort realizadas com vetor de valores aleatórios tipo m x t

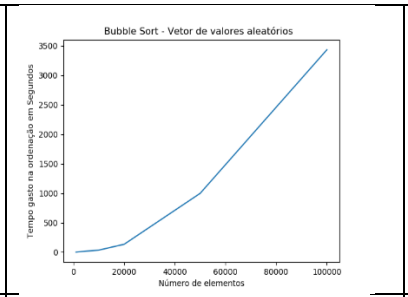


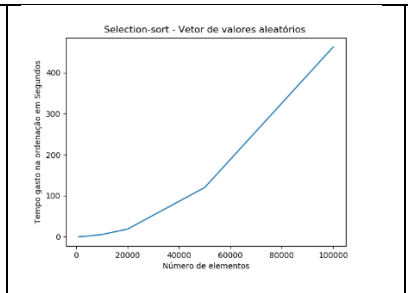
A figura 40 apresenta o gráfico gerado com vetores de valores aleatórios. O gráfico foi do tipo m x t, foi traçado utilizando número de movimentações realizadas e o tempo gasto na ordenação.

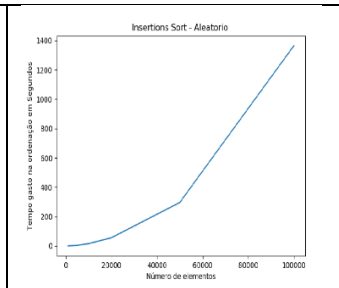
TABELA PARA ANALISE E COMPARAÇÃO

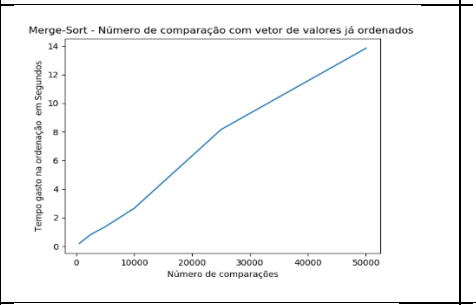
A tabela a seguir, podemos observar algumas diferenças relacionadas aos algoritmos.

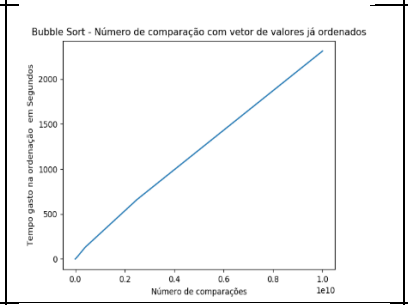


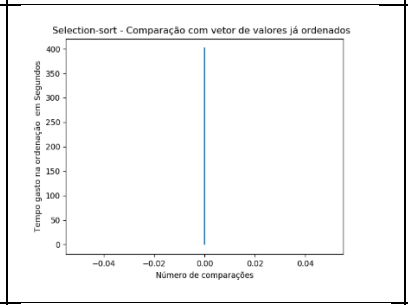


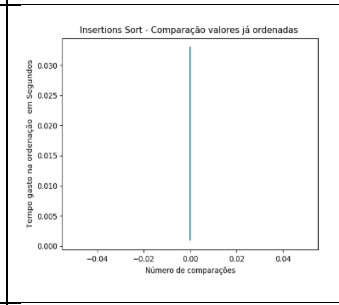


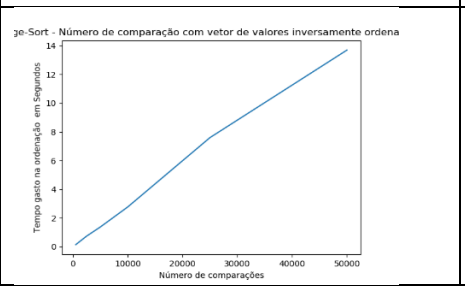


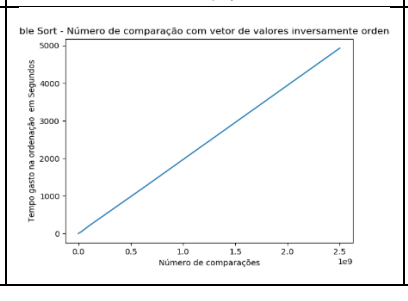


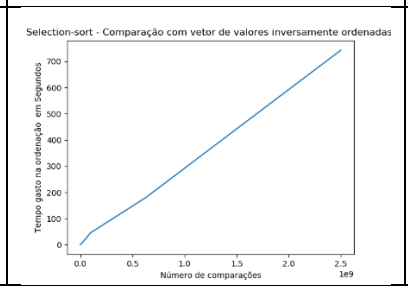












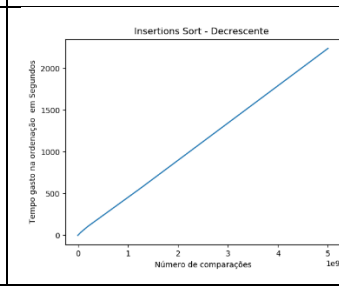
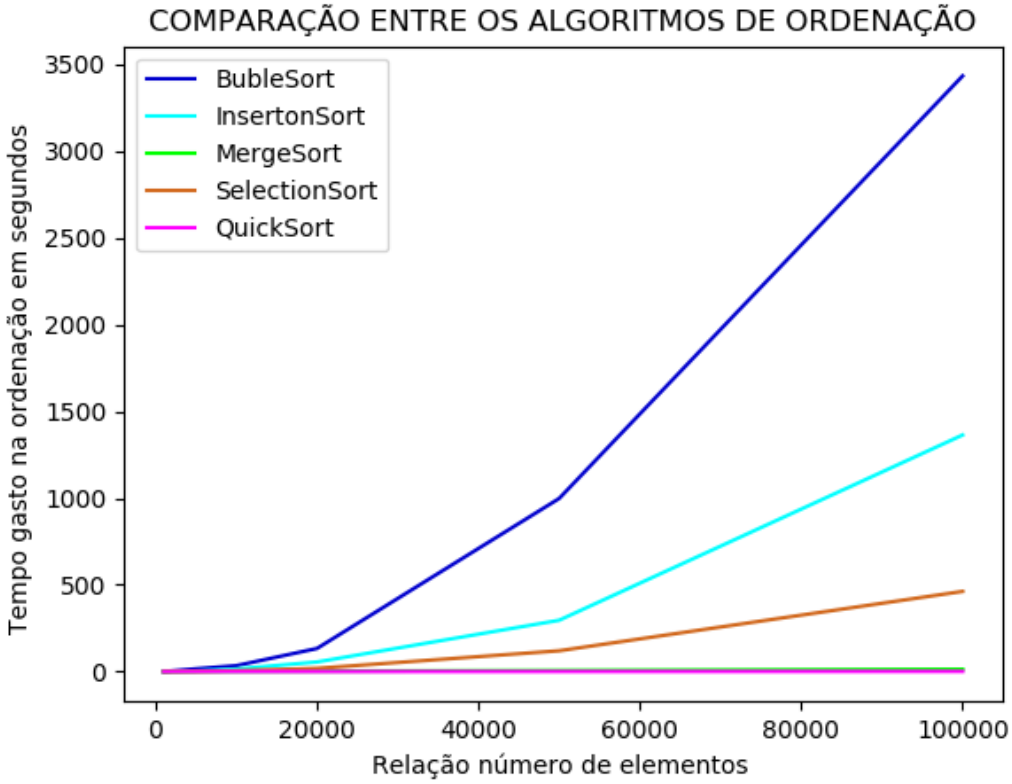


GRAFICO APRESENTANDO AS COMPARAÇÃO

O gráfico a seguir será apresenta a relação número de elementos a ser ordenado e t, o tempo gasto na ordenação de todos os algoritmos utilizados para análise, apenas vetores de valores aleatórios.



Nessa comparação o algoritmo com pior desempenho foi Bublesort, devido a sua grande quantidade de comparações. Os algoritmos QuickSort e MergeSort com os algoritmos com melhor desempenho.

GITHUB

https://github.com/David15117/A-an-lise-emp-rica---Algoritmos-de-ordena-o.git