UNIVERSIDADE FEDERAL DO TOCANTINS
CAMPUS UNIVERSITÁRIO DE PALMAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO
RELATÓRIO DE PROJETO E ANÁLISE DE ALGORITMOS

PAA - ANÁLISE EMPÍRICA SOBRE ALGORITMOS DE ORDENAÇÃO

Fernando Barroso Noleto

Orientador: Dr. Warley Gramacho da Silva

Palmas
Abril de 2018

Resumo

Neste relatório será apresentado todos os testes e análises, além das conclusões feitas ao se executar 5 tipos de métodos de ordenação de vetores: Bubble sort, Insertion sort, Selection sort, Merge sort e Quick sort. Estes algoritmos de ordenação foram executados sobre vetores sobre 3 circunstâncias: vetores que já estavem ordenados, vetores inversamente ordenados e vetores ordenados aleatoriamente. Os resultado desta análise foram exibidos ao final do relatório.

Palavra-chave: LaTeX. UFTeX. Algoritmos de ordenação. Bubble sort. Insertion sort. Selection sort. Merge sort. Quick sort.

Abstract

In this report we will present all the tests and analyzes, as well as the conclusions made when executing 5 types of vector ordering methods: Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort. These ordering algorithms were executed on vectors over 3 circumstances: vectors that were already ordered, inverse ordered vectors and vectors randomly ordered. The results of this analysis were displayed at the end of the report.

Keywords: LaTeX. UFTeX. Sorting algorithms. Bubble sort. Insertion sort. Selection sort. Merge sort. Quick sort.

Sumário

1 Introdução

A análise empírica (prática), também chamada de análise de desempenho ou bateria de testes, leva em consideração o tipo de computador onde o programa foi executado, a linguagem em que foi implementado, tipo da chave (grande ou pequena), estruturas a serem ordenadas (leves ou pesadas), variação no tamanho das entradas, variação na forma das entradas (ascendente, descendente e aleatória).

Os resultados de uma avaliação empírica de algoritmos de ordenação consistem nos tempos coletados para se ordenar um conjunto de elementos, números de comparações realizadas, número de movimentações e gráficos ilustrativos. O objetivo deste tipo de análise é verificar, na prática, o comportamento assintótico de um algoritmo. No caso de algoritmos de ordenação, deve-se coletar dados quantitativos para o tempo em função do número de elementos a serem ordenados, e em seguida traçar gráficos com os valores obtidos.

A partir destes conceitos, foram implementados 5 algoritmos de ordenação: bubble sort, insertion sort, selection sort, merge sort e quick sort a fim de se obter o comportamento assintótico dos tais algoritmos através de testes de realizados observando o tempo que levou para ser executado, o número de trocas efetudas, etc. além de construir os gráficos destes testes para melhor análise. Foram utilizados para o tamanho dos vetores os valores de 100, 500, 1000, 5000, 10000, 50000, 100000, 200000 e 500000 afim de se obter o comportamento dependendo do tamanho do vetor.

Ao final, será apresentado todos os resultados (gráficos) obtidos através destes testes e as conclusões que podem ser obtidos através destes resultados.

2 Análise do ponto de vista do número de trocas efetuadas para se ordenar um vetor

Foram realizados testes para analisar o comportamento assintótico, o tempo necessário para se ordenar o vetor e a quantidade de trocas efetuadas para a ordenação de todos os algoritmos. Para isto foi implementado cada função que contém os respectivos algoritmo e passados como parêmetros um vetor que variava a quantidade de elementos: de 100 a 500000.

O vetor também variava a forma de ordenação em que se encontrava inicialmente: O vetor podia já estar ordenado em ordem crescente, ou ordenado em ordem decrescente ou poderia estar ordenado de forma aleatória (randômica).

Após executar todos estes testes, foi plotado um gráfico para cada situação. O gráfico obtido do ponto de vista do número de trocas do usando todos os algoritmos com o vetor ordenado em ordem crescete está apresentado na figura abaixo:

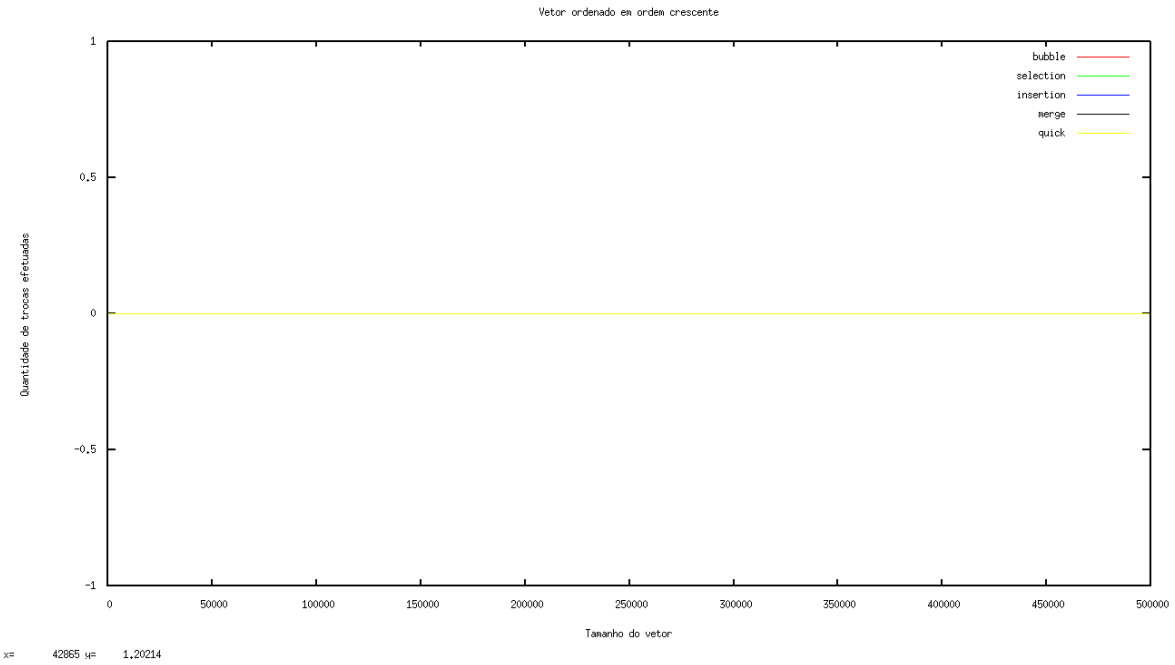

Figura 2.1: *Vetores ordenados em ordem crescente*

Analisando o gráfico, pode-se observar que todos os algoritmos não fizeram nenhuma troca, pois como o vetor já está ordenado então é necessário mover os elementos de posição. (1)

Agora vamos analisar o gráfico que representa a troca de posição dos elementos dentro do vetor num vetor que está ordenado de maneira aleatória:

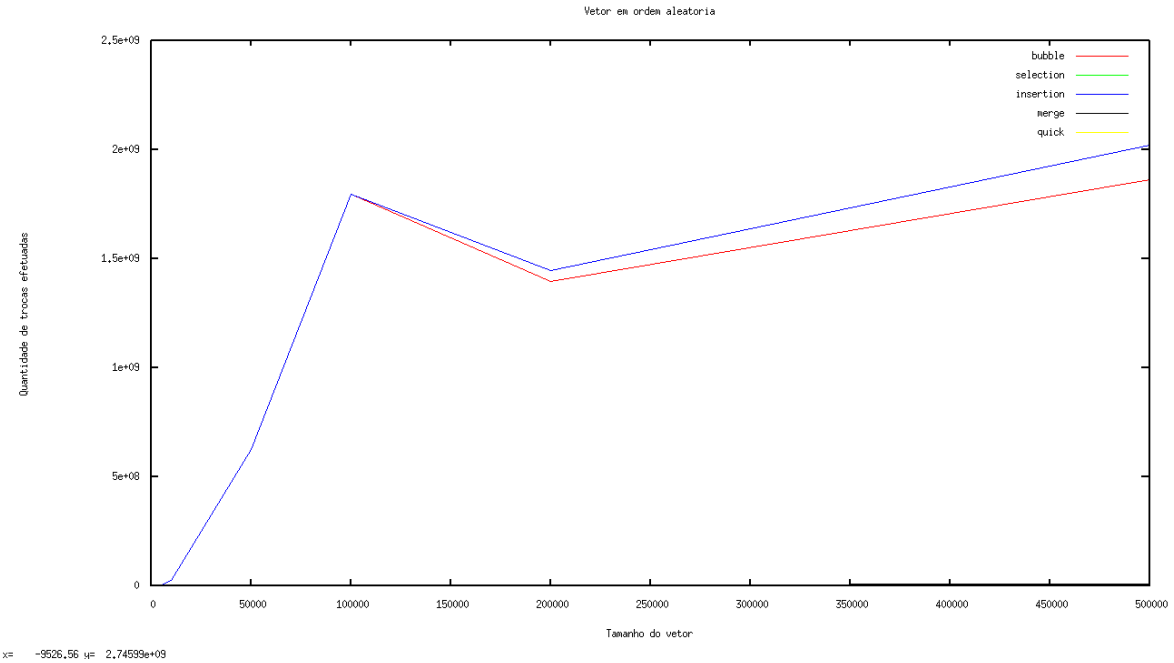

Figura 2.2: *Vetores ordenados em ordem aleatória*

A partir deste gráfico podemos observar que o algoritmo insertion sort juntamente com o bubble sort possui o maior número do trocas feitas para poder se ordenar o vetor corretamente.

Isso deve ao fato do algoritmo bubble sort e o insertion, para os piores casos, fazerem comparações e trocas dentro de 2 laços de repetição aninhados, ou seja ele tem comportamento quadrático.

Por fim, vamos analisar o gráfico que corresponde a troca de posição dos elementos dentro do vetor num vetor que está inversamente ordenado, ou seja, em ordem decrescente:

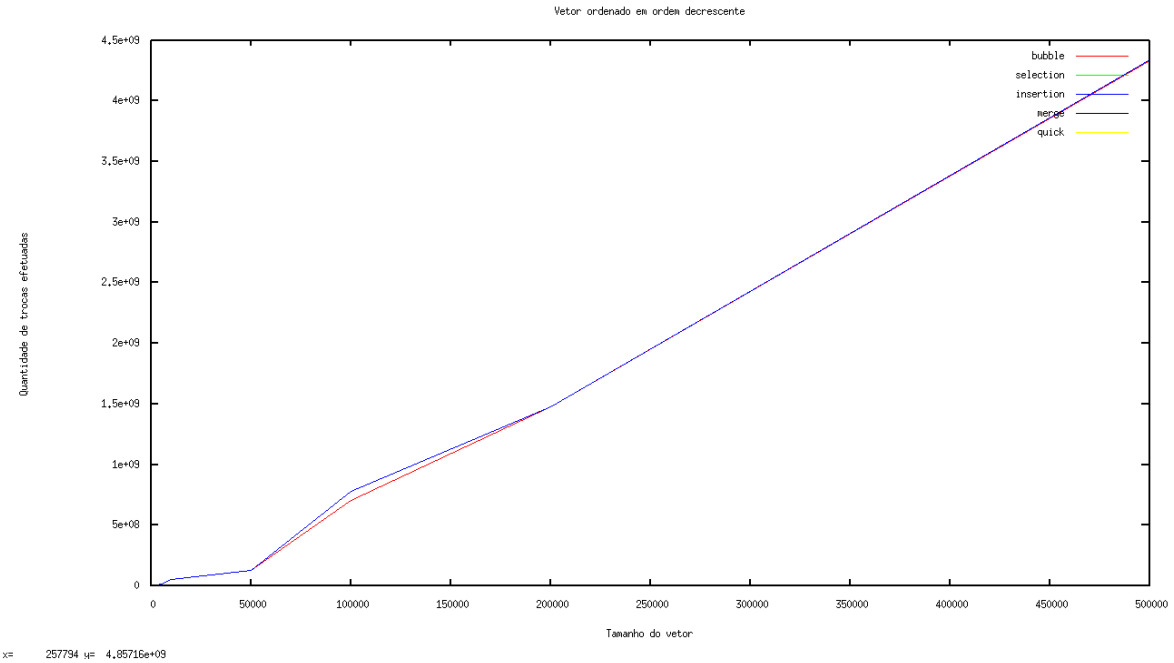

Figura 2.3: *Vetores ordenados em ordem decrescente*

Analisando o gráfico é possível chegar a conclusão de que o novamente, os algoritmos insertion sort juntamente com o bubble sort foram os métodos de ordenação que mais efetuaram trocas para conseguir ordenar corretamente os vetores ordenados inversamente. Isso ocorre devido a ambos os algoritmos possuírem o comportamento quadrático. (2)

3 Análise do ponto de vista do tempo necessário para se ordenar um vetor

Após realizar as análises do ponto de vista do número de trocas realizadas dentro do vetor para ordená-lo, vamos fazer as análises a partir do tempo necessário fazer a ordenação. Também será analisado os algoritmos bubble sort, insertion sort, selection sort, merge sort e quick sort.

Primeiramente vamos verificar o gráfico que corresponde ao tempo levado para se ordenar um vetor que já está ordenado em ordem crescente:

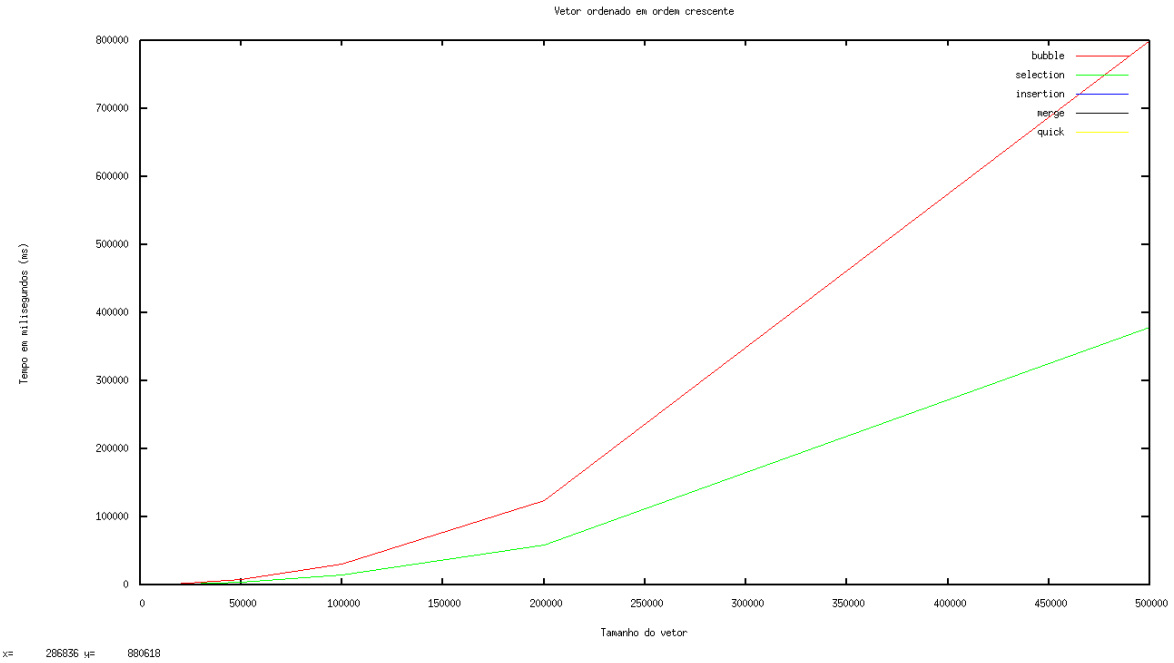

Figura 3.1: *Vetores ordenados em ordem crescente*

Podemos observar a partir do gráfico que os algoritmos que obtiveram os piores desempenhos foram os algoritmos bubble sort e selection sort respectivamente. Isso ocorre devido ao algorirmo bubble sort e selection sort serem os algoritmos que tem os piores comportamento em relação aos outros algoritmos. O algoritmo quick sort e merge sort utilizam a estratégia de divisão e conquista para poderem ordenar os vetores, economizando assim, tempo e preocessamento.

Assim como explicado acima, para todos os casos analisados, o comportamento dos algoritmos seguem o mesmo padrão: merge sort e quick sort, por utilizarem a estratégia

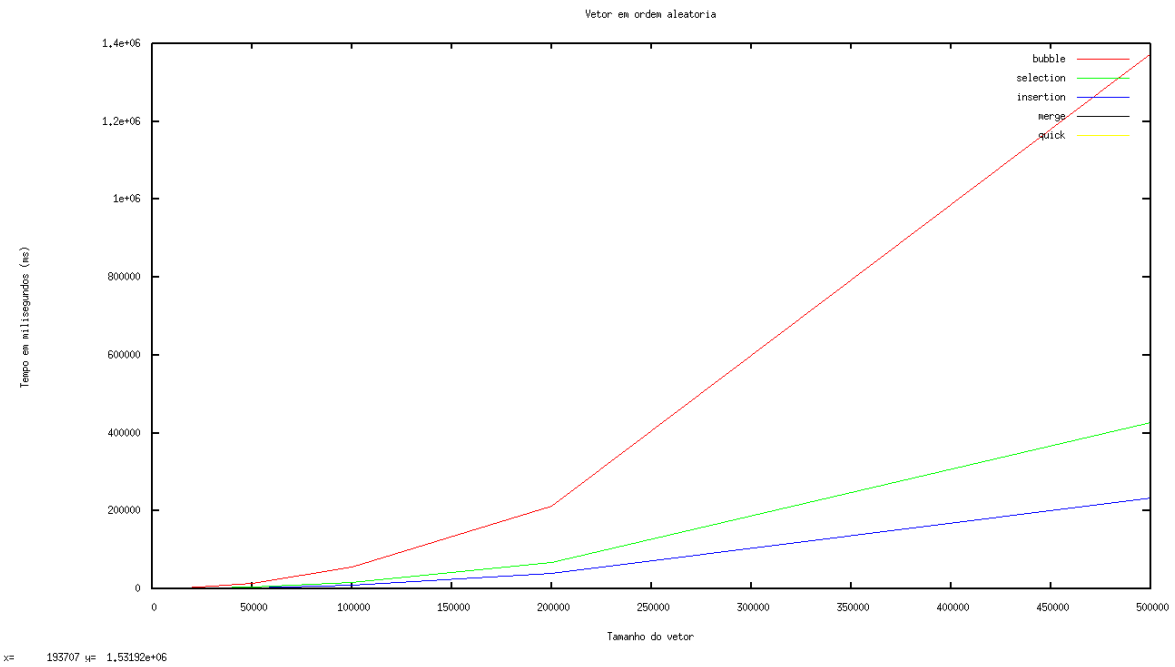

Figura 3.2: *Vetores ordenados em ordem aleatória*

de divisão e conquista, se sobressaem em relação ao outros algoritmos do ponto de vista do tempo gasto para realizar ordenação nos vetores.

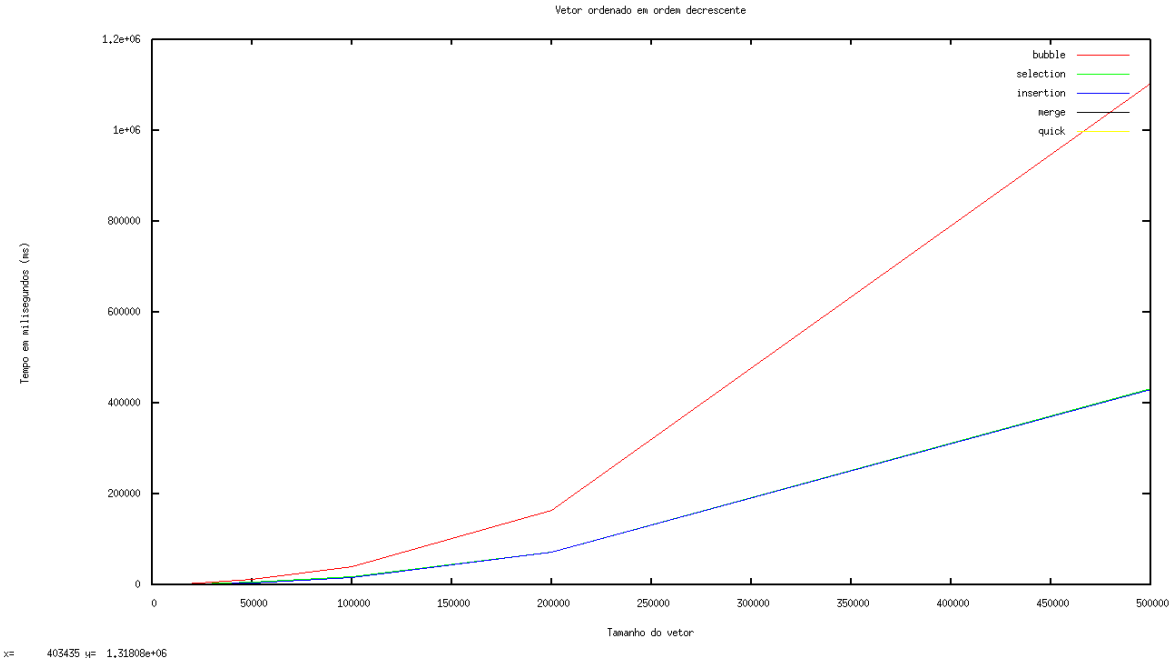

Figura 3.3: *Vetores ordenados em ordem decrescente*

4 Comportamento assintótico dos algoritmos de ordenação de vetores

Após analisar todos os gráficos apresentados, podemos elaborar uma tabela que contém o comportamento em qualquer caso para todos os algoritmos analisado:

Tabela 4.1: *Comportamento assintótico dos algoritmos.*

Algoritmo	Complexidade		
	Melhor caso	Médio caso	Pior caso
Bubble sort	O(n)	O(n^2)	O(n^2)
Insertion sort	O(n)	O(n^2)	O(n^2)
Selection sort	O(n^2)	O(n^2)	O(n^2)
Merge sort	O(n*log(n))	O(n*log(n))	O(n*log(n))
Quick sort	O(n*log(n))	O(n*log(n))	O(n^2)

(3)

5 Conclusão

Ao final deste relatório podemos concluir que o algoritmo de merge sort é o melhor algoritmo dentre os algoritmos analisado para se ordenar vetores. O merge sort tem o comportamento logarítmico para qualquer caso, sendo assim, ele se sobressai em relação aos outros algoritmos. Não podemos deixar de destacar o algoritmo quick sort, que também é um ótimo algoritmo de ordenação e é bastante popular devido a sua facilidade de implementação e eficiência.

A eficiência dos algoritmos merge sort e quick sort se deve, principalmente, ao fato de ambos os algoritmos se utilizarem da estratégia de divisão-e-conquista para chegar ao seu objetivo. Essa estratégia consiste em dividir o problema inicial em diversos problemas menores afim de se resolver cada pequeno problema separadamente, assim sendo possível melhorar o desempenho do método utilizado.

Por outro lado, estes tipos de algoritmosGasto extra de memória. O algoritmo cria uma cópia do vetor para cada nível da chamada recursiva, totalizando um uso adicional de memória igual a (n*log(n)).

(3)

Referências Bibliográficas

1 FEOFILOFF, P. *Análise do algoritmo Quicksort*. 2015. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/quick.html>.

2 HONORATO, B. de A. *Algoritmos de ordenação: análise e comparação*. 2017. Disponível em: <https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>.

3 MERGE sort. 2018. Disponível em: <https://pt.wikipedia.org/wiki/Merge_sort>.