Relação entre Algoritmos de Ordenação

Arthur Alexsander Martins Teodoro
Sulo Ricardo Dias Fernandes

[1]Instituto Federal de Minas Gerais - Campus Formiga

Resumo

Neste trabalho será mostrado uma relação entre quatro algoritmos de ordenação implementados, sendo estes dois com $O(n^2)$ e os outros $O(n * log(n))$. Será mostrado feita comparações entre três métricas colhidas: tempo para ordenação, trocas realizadas e comparações feitas.

1. Introdução

Segundo [Ziviani 1999], ordenar corresponde ao processo de rearranjar um conjunto de objetos em uma ordem crescente ou decrescente. O objetivo principal da ordenação é facilitar a recuperação posterior de itens do conjunto ordenado. Imagine como seria difícil utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética. A atividade de colocar as coisas em ordem está presente na maioria das aplicações onde os objetos armazenados têm que ser pesquisados e recuperados, tais como dicionários, índices de livros, tabelas e arquivos.

O problema de alguns métodos de ordenação é a complexidade, ou seja, o custo que o algoritmo leva para resolver o problema. Logo, para a resolução deste problema existem vários métodos, cada um usando de uma estratégia diferente.

Na terceira parte do Trabalho Multidisciplinar das disciplinas Algoritmos e Estruturas de Dados e Programação, ambas ofertadas pelo Instituto Federal de Minas Gerais - Campus Formiga, o problema de ordenação é tratado e foi implementado quatro métodos diferentes: *Bubble Sort*, *Insertion Sort*, *Quick Sort* e o *Quick Sort* Turbinado, uma mistura do método de inserção (*Insertion Sort*) e *Quick Sort*.

2. Métodos Implementados

2.1. Bubble Sort

O *BubbleSort* é um algoritmo que percorre o vetor inteiro comparando elementos adjacentes (dois a dois). Os elementos que estão fora de ordem são trocados. O resultado da ordenação se dá repetindo os dois passos acima com os primeiros n-1 elementos, depois com os primeiros n-2 elementos, até que reste apenas um elemento. Esta comparação com os elementos adjacentes faz com que o algoritmo possui ordem de complexidade de $O(n^2)$.

2.2. Insertion Sort

O método de ordenação por Inserção Direta é o mais rápido entre os outros métodos considerados básicos - sendo neste trabalho, implementado somente o BubbleSort. A principal característica deste método consiste em ordenarmos o arranjo utilizando um sub-arranjo ordenado localizado em seu inicio, e a cada novo passo, acrescentamos a este

sub-arranjo mais um elemento, até que atingimos o último elemento do arranjo fazendo assim com que ele se torne ordenado. Sua complexidade é, no melhor caso, $O(n)$, já no pior e médio caso, $O(n^2)$.

2.3. Quick Sort

Este método de classificação foi inventado por Hoare e seu desempenho é o melhor na maioria das vezes. O primeiro elemento do vetor a ser classificada é escolhido como o pivô. Depois da primeira fase da classificação, o pivô ocupa a posição que ocupará quando o vetor estiver completamente classificado. Os registros com valores de chaves menores do que o valor de chave do registro pivô o precedem no vetor e os registros com valores de chaves maiores do que o valor de chave do registro pivô o sucedem no vetor. Cada registro é comparado com o registro pivô e suas posições são permutadas se o valor de chave do registro for maior e o registro preceder o registro pivô, ou se o valor de chave do registro for menor e o registro suceder o registro pivô. A posição do registro pivô no final de uma fase divide o vetor original em dois sub-vetores (partições), cada uma delas precisando ser ordenada. A complexidade de tal algoritmo é, no melhor e médio caso $O(n * log(n))$ e no pior caso $O(n^2)$.

2.4. Quick Sort Turbinado

Este método é a junção de dois métodos, o de inserção e o Quick Sort. O método do Quick Sort não é muito bom com partições pequenas, então, quando a partição tem tamanho menor que 40, o método da inserção é chamada, ordenando a partição. A escolha do método de inserção se dá uma vez que este método é bom para vetores pequenos e quase ordenados, situação que se encontra as partições com tamanho inferior a 40 geradas pelo método do Quick Sort. Este método possui complexidade igual ao QuickSort.

3. Comparação

Os testes mostrados nesta seção foram realizados em uma máquina usando como sistema operacional Ubuntu 14.04, processador i5 1.60GHz x 4 e 8 Gb de RAM.

O algoritmo foi escrito em linguagem C padrão ANSI, compilado com GCC 4.8.4 via terminal.

3.1. BubbleSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	124750	124750	124750	124750	0
1000	499500	499500	499500	499500	0
10000	49995000	49995000	49995000	49995000	0
100000	4999950000	4999950000	4999950000	4999950000	0
300000	44999850000	44999850000	44999850000	44999850000	0

Figura 1. Tabela da comparações realizados pelo algoritmo BubbleSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	62253	61755	61755	61921	287,5204340564
1000	260948	259950	259950	260282,666667	576,1955686512
10000	24934980	25119061	24827211	24960417,3333	147578,450562178
100000	2495458910	2497972044	2495964137	2496465030,33	1329334,88031509
300000	22475861296	22485208025	22485582650	22482217324	5507667,54145803

Figura 2. Tabela da trocas realizados pelo algoritmo BubbleSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	0,001498	0,001285	0,001283	0,0013553333	0,000123557
1000	0,006063	0,005894	0,0054	0,0057856667	0,0003445204
10000	0,63916	0,652654	0,675291	0,6557016667	0,0182572861
100000	69,471099	69,884974	69,647866	69,6679796667	0,2076693251
300000	650,688344	649,966744	651,194943	650,616677	0,617227928

Figura 3. Tabela de tempo levado pelo algoritmo BubbleSort

Como pode ser visto nas imagens, o algoritmo da bolha é um dos mais lentos, gastando aproximadamente 651 segundos (10.85 minutos) para ordenar um vetor de 300000 elementos. Na contagem de tempo também deve ser levado em consideração as chamadas das funções de trocas e comparações, que levam um pouco mais de tempo. Porém, como o algoritmo faz todas as comparações possíveis, o algoritmo se torna lento.

3.2. InsertionSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	62750	62252	62252	62418	287,5204340564
1000	261944	260946	260946	261278,666667	576,1955686512
10000	24944974	25129052	24837198	24970408	147579,997276054
100000	2495558897	2498072030	2496064129	2496565018,67	1329333,37148826
300000	22476161281	22485508017	22485882637	22482517312	5507670,05300838

Figura 4. Tabela da comparações realizados pelo algoritmo InsertionSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	62253	61755	61755	61921	287,5204340564
1000	260948	259950	259950	260282,666667	576,1955686512
10000	24934980	25119061	24827211	24960417,3333	147578,450562178
100000	2495458910	2497972044	2495964137	2496465030,33	1329334,88031509
300000	22475861296	22485208025	22485582650	22482217324	5507667,54145803

Figura 5. Tabela da trocas realizados pelo algoritmo InsertionSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	0,000792	0,000809	0,000784	0,000795	1,276714533480E-005
1000	0,003449	0,003442	0,003308	0,0033996667	7,946277954699E-005
10000	0,313443	0,315917	0,315121	0,314827	0,0012629315
100000	33,163838	33,541754	33,144296	33,283296	0,2240443612
300000	309,493404	310,608375	310,047587	310,049788667	0,5574887606

Figura 6. Tabela de tempo levado pelo algoritmo InsertionSort

Pode ser visto que o algoritmo de inserção é um mais rápido que o algoritmo da bolha, uma vez que este realiza uma menor quantidade de comparações e para quando o

vetor já se encontra ordenado, diferente do BubbleSort. Nos teste realizados, o algoritmo de Inserção acabou realizando a mesma quantidade de trocas que o da Bolha, isso se dá porque foi contabilizada cada operação de mover um elemento para trás na hora da inserção do valor.

3.3. QuickSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	3707	3581	3581	3623	72,7461339179
1000	8001	8204	8204	8136,33333333	117,2021046455
10000	105348	109615	104521	106494,666667	2733,7414532712
100000	1324484	1340717	1338156	1334452,33333	8727,2809243964
300000	4552012	4474470	4388470	4471650,66667	81807,4441682011

Figura 7. Tabela da comparações realizados pelo algoritmo QuickSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	3510	3338	3338	3395,33333333	99,3042463006
1000	7518	7566	7566	7550	27,7128129211
10000	101490	106194	101626	103103,333333	2677,4594923796
100000	1285880	1309924	1274589	1290131	18046,9888624114
300000	4427404	4474470	4293547	4398473,66667	93866,9596947368

Figura 8. Tabela da trocas realizados pelo algoritmo QuickSort

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	0,000753	0,000761	0,000805	0,000773	0,000028
1000	0,001549	0,001721	0,001602	0,001624	8,808518604169E-005
10000	0,015724	0,014966	0,01501	0,0152333333	0,0004254989
100000	0,170873	0,162163	0,176148	0,169728	0,0070624588
300000	0,523315	0,536791	0,547767	0,5359576667	0,0122472817

Figura 9. Tabela de tempo levado pelo algoritmo QuickSort

Percebemos que o QuickSort é dos um dos métodos mais rápidos de todos, uma vez que a estratégia ”dividir para conquistar”aumenta muito a velocidade do algoritmo. Um dos únicos contras do algoritmo QuickSort é que sua implementação é recursiva, criando uma certa estranheza para programadores menos experientes.

3.4. QuickSort Turbinado

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	5618	5119	5119	5285,33333333	288,0977843256
1000	11146	11169	11169	11161,3333333	13,2790561914
10000	135973	140517	135598	137362,666667	2738,1600269768
100000	1633871	1654240	1662449	1650186,66667	14713,8592603482
300000	5372195	5444042	5292963	5369733,33333	75569,5766319577

Figura 10. Tabela da comparações realizados pelo algoritmo QuickSort Turbinado

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	5396	4801	4801	4999,33333333	343,5234101678
1000	10710	10489	10489	10562,6666667	127,5944094909
10000	131656	137712	132695	134021	3238,4395933845
100000	1618374	1625074	1598071	1613839,66667	14060,961429907
300000	5262913	5279397	5156755	5233021,66667	66561,1290268827

Figura 11. Tabela da trocas realizados pelo algoritmo QuickSort Turbinado

Tamanho	Execução 1	Execução 2	Execução 3	Média	Desvio Padrão
500	0,000117	0,000112	0,000112	0,0001136667	2,886751345948E-006
1000	0,000243	0,000268	0,000258	0,0002563333	1,258305739212E-005
10000	0,003249	0,00294	0,002925	0,003038	0,0001828852
100000	0,038304	0,038446	0,03833	0,03836	7,560423268575E-005
300000	0,10825	0,122211	0,121558	0,1173396667	0,0078786504

Figura 12. Tabela de tempo levado pelo algoritmo QuickSort Turbinado

Pelas Tabelas apresentadas com os valores colhidos do algoritmo QuickSort Turbinado percebemos que, mesmo realizando mais trocas e comparações que o QuickSort, este algoritmo tem seu tempo de execução menor em alguns casos. Isso se dá ao fato de quando a partição se encontra em um tamanho pequeno e quase ordenada, o algoritmo de inserção é chamado para ordenar tal partição, diminuindo o tempo, uma vez que a partição se encontra em um ótimo estado para o algoritmo de inserção.

4. Avaliação

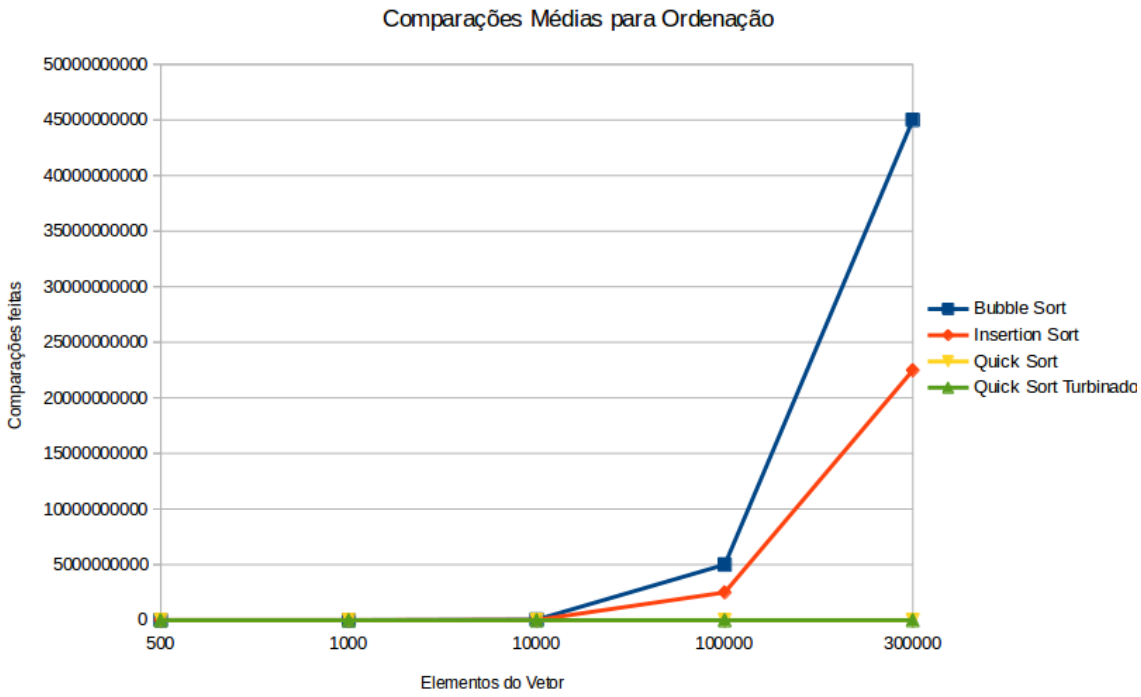

Figura 13. Gráfico da média de comparações de cada algoritmo

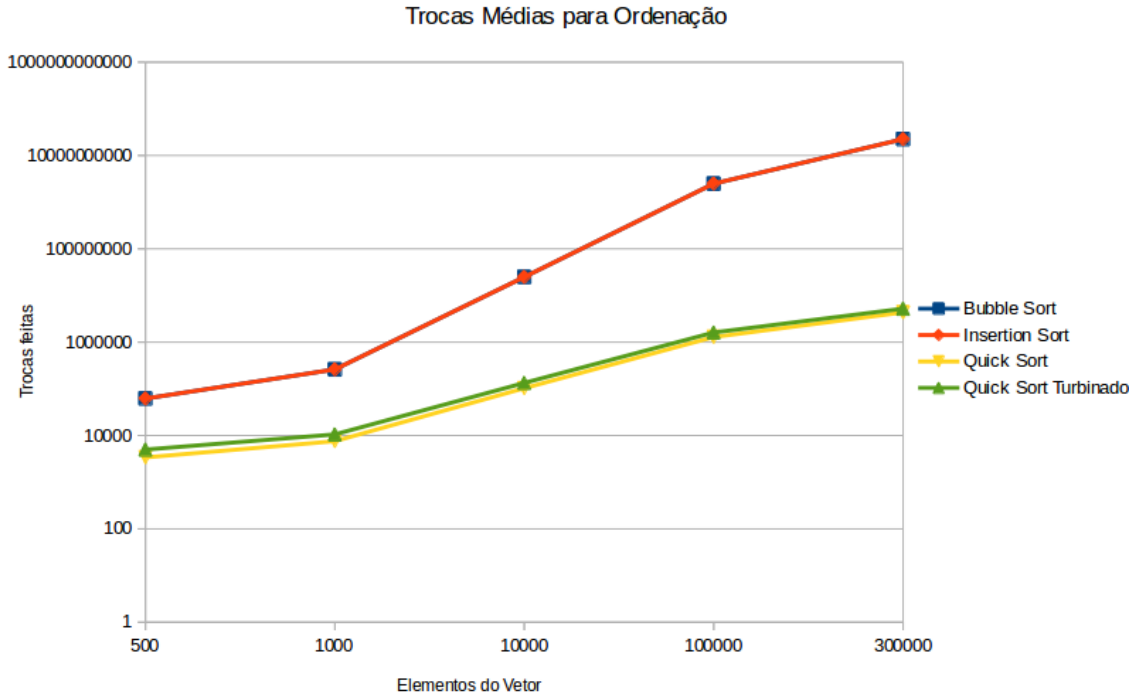

Figura 14. Gráfico da média de trocas de cada algoritmo

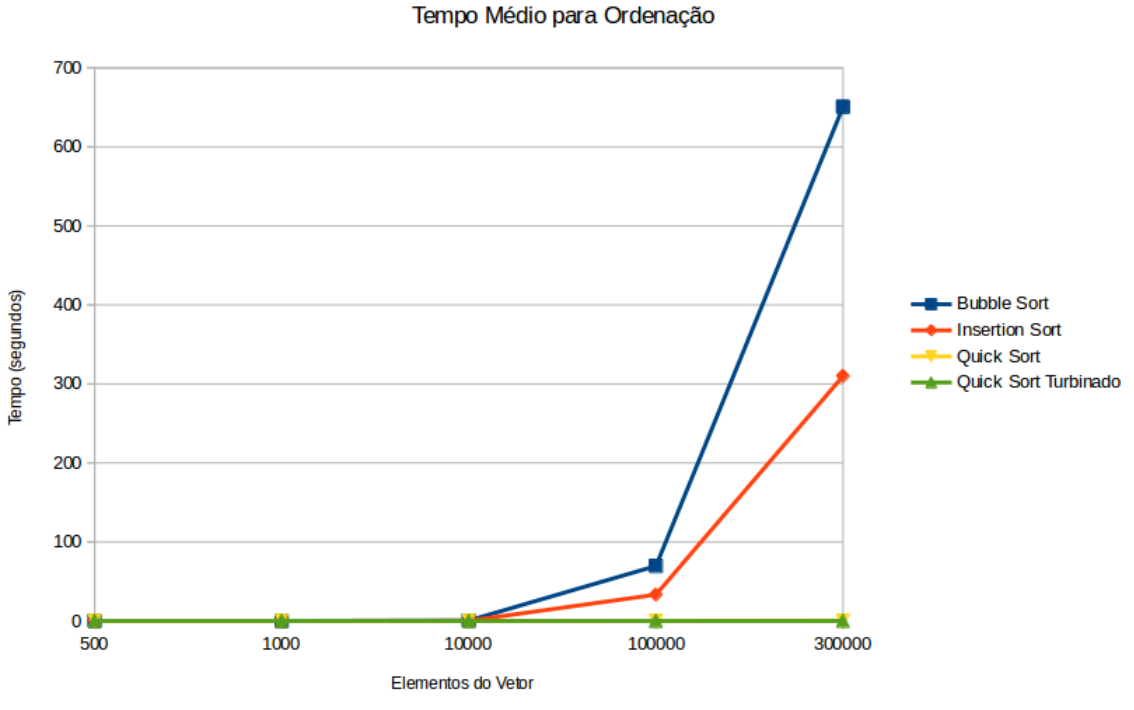

Figura 15. Gráfico da média de tempo de cada algoritmo

Como previsto, os algoritmos QuickSort e QuickSort Turbinado foram os que possuíram uma melhor performance nas três métricas colhidas. Os algoritmos BubbleSort e InsertionSort também atuaram como previsto, com o BubbleSort sendo o pior de todos

e o InsertionSort, o meio termo. A diferença entre os algoritmos de mesma ordem de complexidade se dá por causa da constante que multiplica tal ordem, causando assim a diferença. Porém, todos os algoritmos seguem sua ordem de complexidade, sendo visível pelo gráfico o arco gerado pelos algoritmos $O(n^2)$ e uma leve curva dos algoritmos $O(n * log(n))$.

5. Conclusão

Foi concluído que, em casos de uma pequena massa de dados, qualquer um dos algoritmos pode ser escolhidos, porém, quando a massa de dados se torna grande ou é necessário velocidade, é preferível escolher os algoritmos QuickSort ou o QuickSort Turbinado. Também é preciso notar que, mesmo com uma massa de dados pequena porém com elementos grandes, é preferível o algoritmo QuickSort, uma vez que este faz uma quantidade de trocas menor.

Com a implementação deste trabalho multidisciplinar foi aprimorado os conhecimentos, tanto de programação com estruturação de dados. Também foi um teste empírico de quais algoritmos de ordenação utilizar dependendo de cada situação, como tamanho de massa de dados.

Referências

Ziviani, N. (1999). *Projeto de algotimos com implementação em Pascal e C*. Pioneira, 4th edition.