

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

FACULDADE DE COMPUTAÇÃO

ALGORITOMOS E PROGRAMAÇÃO II

EDUARDO ATENE SILVA

Relatório de tempo de execução dos algoritmos de ordenação

Campo Grande

2020

Introdução

Esta dissertação ilustra os tempos de execução dos algoritmos de ordenação: Bubble Sort, Select Sort, Insertion Sort, Merge Sort e Quick Sort. Para isso, foi criado um algoritmo que irá testar cada algoritmo de ordenação e irá calcular o tempo de execução, gerando tabela para demonstrar os casos. Os testes serão feitos por meio de vetores alocados dinamicamente com os respectivos tamanhos.

Ao decorrer deste relatório, será apresentada uma discussão referente à saída produzida.

Relatório

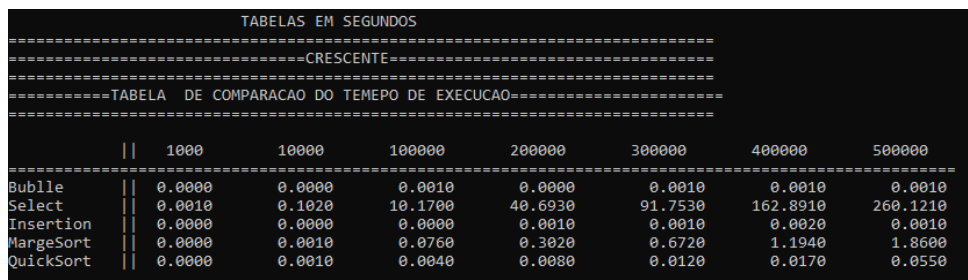
De acordo com os tempos de execução dos algoritmos de ordenação: Bubble Sort, Select Sort, Insertion Sort, Merge Sort e Quick Sort. Para isso, foi criado e executado um programa que irá testar cada algoritmo e irá calcular o tempo de execução. Para os testes foram gerados vetores alocados dinamicamente com os respectivos tamanhos: 1,000; 10,000; 100,000; 200,000; 300,000; 400,000; 500,000. Os vetores foram ordenados de forma crescente, decrescente e embaralhado. Para embaralhar o vetor, foi solicitado o algoritmo Fisher Yates. Esse algoritmo irá gerar uma permutação aleatória, ou seja, embaralhar o vetor. Para embaralhar o vetor, foi usada a função *rand()*, da biblioteca *stdlib.h*, que foi usada para sortear um número para embaralhar o vetor.

Os algoritmos de ordenação irão ordenar os vetores de forma crescente. Mas por que ordenar um vetor crescente que já está ordenado? Iremos ordenar este vetor para medir o tempo de execução de um vetor que já está ordenado.

Saída do tempo de execução dos algoritmos

Como já dito, os vetores foram gerados de forma crescente, decrescente e embaralhados. Neste tópico, vamos discutir sobre as saídas produzidas e debater qual é o algoritmo mais eficiente, de acordo com cada saída.

Vetores crescente:



```
=====
TABELAS EM SEGUNDOS
=====CRESCENTE=====
=====TABELA DE COMPARACAO DO TEMPO DE EXECUCAO=====
=====
```

	1000	10000	100000	200000	300000	400000	500000
Bubble	0.0000	0.0000	0.0010	0.0000	0.0010	0.0010	0.0010
Select	0.0010	0.1020	10.1700	40.6930	91.7530	162.8910	260.1210
Insertion	0.0000	0.0000	0.0000	0.0010	0.0010	0.0020	0.0010
MergeSort	0.0000	0.0010	0.0760	0.3020	0.6720	1.1940	1.8600
QuickSort	0.0000	0.0010	0.0040	0.0080	0.0120	0.0170	0.0550

Imagem 1: Tabela referente a saída produzida no algoritmo.

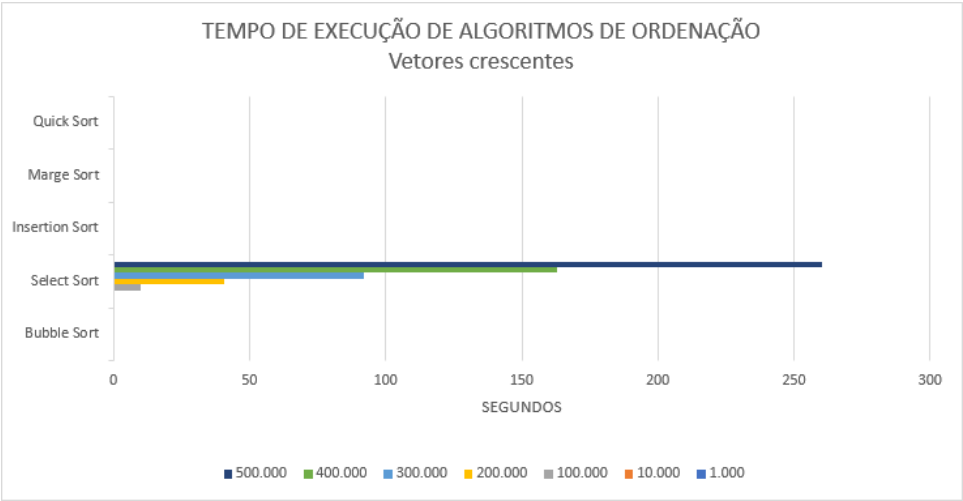


Imagem 2: Gráfico referente a saída produzida da Imagem 1.

Está nitidamente explicito que o algoritmo Select Sort demandou mais tempo em vetores já ordenados. Já o Marge Sorte, teve um tempo entre 0 e 2 segundos para verificar/ordenar os valores. Já os outros algoritmos, em vetor crescente, o tempo de execução nestes foi quase nulo.

Vetores decrescente:

TABELAS EM SEGUNDOS								
-----DECRESCENTE-----								
=====TABELA DE COMPARACAO DO TEMPO DE EXECUCAO=====								
		1000	10000	100000	200000	300000	400000	500000
Bublle		0.0000	0.0000	0.0000	0.0010	0.0000	0.0020	0.0010
Select		0.0010	0.1030	10.2720	41.8560	92.7610	165.1740	268.1560
Insertion		0.0000	0.0000	0.0000	0.0010	0.0010	0.0010	0.0020
MargeSort		0.0000	0.0010	0.0730	0.3010	0.6720	1.1900	1.8620
QuickSort		0.0000	0.0000	0.0040	0.0090	0.0130	0.0180	0.0220

Imagem 3: Tabela referente a saída produzida no algoritmo.

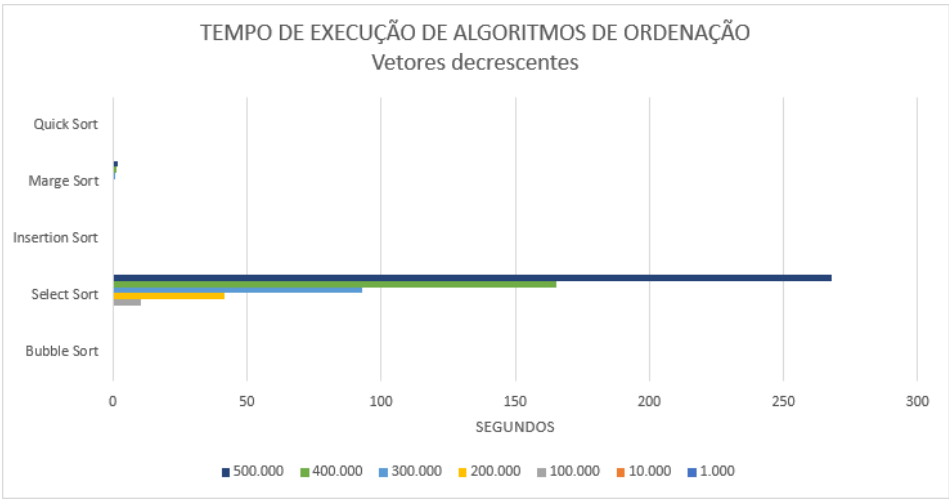


Imagem 4: Gráfico referente a saída produzida da Imagem 3.

Novamente, o algoritmo Select Sort demandou mais tempo de execução comparado com os outros algoritmos. Já o Marge Sorte, teve um tempo entre 0 e 1.860 segundos para verificar/ordenar os valores. Já os outros algoritmos, em vetor decrescente, o tempo de execução, novamente, foi quase nulo em todos os tamanhos dos vetores.

Vetores embaralhado:

TABELAS EM SEGUNDOS

=====EMBARALHADO=====

-----TABELA DE COMPARACAO DO TEMPO DE EXECUCAO-----

	1000	10000	100000	200000	300000	400000	500000
Bubble	0.0020	0.3130	37.2370	123.3340	233.6820	385.4000	538.3040
Select	0.0010	0.1040	10.1780	41.5620	93.3710	164.9950	257.8080
Insertion	0.0010	0.0690	7.0200	19.6790	32.6270	45.8080	59.3770
MergeSort	0.0010	0.0010	0.0000	0.3120	0.6930	1.2160	1.8910
QuickSort	0.0000	0.0010	0.0120	0.0260	0.0380	0.0510	0.0640

Imagem 5: Tabela referente a saída produzida no algoritmo.

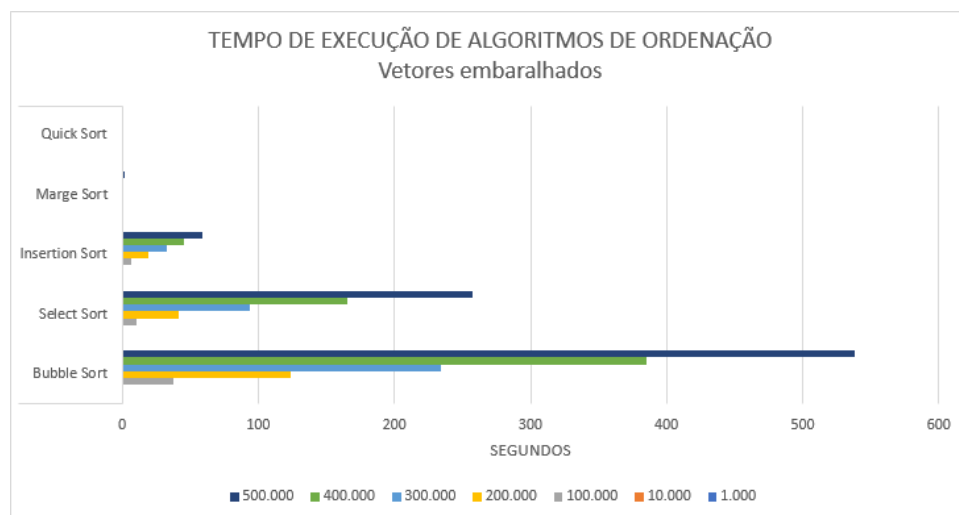


Imagem 6: Gráfico referente a saída produzida da Imagem 5.

Neste caso de teste, o algoritmo que demandou mais tempo foi o Bubble Sort, principalmente com vetores com grande quantidade de dados. Em segundo, foi o Select, que teve a metade do tempo do Bubble e também demonstro pouco eficientes para vetores grandes. Logo em seguida, vem o algoritmo Insertion, que demonstro mais eficiente que o Bubble e o Select. Os algoritmos que demonstro mais eficiente em vetores embaralhados foram os algoritmos Marge Sort e Quick Sort. O Quick Sort em especial, o tempo de execução neste foi quase nulo, mostrou muito eficiente em vetores grandes.

Observação/Discussão

O tempo de execução do Bubble Sort não se monstro eficientes em termo de velocidade de vetores com grandes quantidades. Demonstro eficientes em vetores já ordenados de forma crescente e decrescente.

O tempo de execução do Select Sort se manteu constante nos testes aplicados, ou seja, ele sempre faz as mesmas comparações, independentemente do se o vetor esteja ordenado ou não. Porém, comparado com os outros algoritmos, é bem simples de ser implementado e não necessita de um vetor auxiliar, ou seja, demanda menos tempo na memória.

O tempo de execução do Insertion Sort foi mais eficiente nos três casos de testes que os algoritmos de Bubble e Select. O algoritmo é de simples implementação e se mostrou eficiente em vetores pequenos e se mostrou estável.

O tempo de execução do Merge Sort se mostrou muito eficiente em vetores com grande quantidade de dados. Por conta de utilizar um vetor auxiliar, ocupa mais espaço na memória.

O tempo de execução do Quick Sort se mostrou o mais eficiente dentre estes algoritmos nos testes implementados.

Conclusão

De certeza, o algoritmo Quick Sort, dentre os testados, é o mais eficiente e recomendado, pois ele é o mais rápido e a relação se manteve constante. Sendo como o segundo recomendado o algoritmo Merge Sort. Os algoritmos que são menos recomendados e eficientes são os: Bubble Sort, Select Sort e Insertion Sort.

Apêndice

Tabela gerada pelo programa com a demonstração em minutos:

```
Obs: 1 para SIM | 0 para NAO
Gostaria de exibir a tabela em minutos? 1

TABELAS EM MINUTOS
=====
-----CRESCENTE-----
=====
-----TABELA DE COMPARACAO DO TEMPO DE EXECUCAO-----
=====
|| 1000 10000 100000 200000 300000 400000 500000
=====
Bubble || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0
Select || 0 : 0 0 : 0 0 : 10 0 : 42 1 : 35 2 : 53 4 : 25
Insertion || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0
MergeSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 1 0 : 1
QuickSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0

TABELAS EM MINUTOS
=====
-----DECRESCENTE-----
=====
-----TABELA DE COMPARACAO DO TEMPO DE EXECUCAO-----
=====
|| 1000 10000 100000 200000 300000 400000 500000
=====
Bubble || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0
Select || 0 : 0 0 : 0 0 : 11 0 : 43 1 : 35 2 : 50 4 : 28
Insertion || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0
MergeSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 1 0 : 1
QuickSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0

TABELAS EM MINUTOS
=====
-----EMBARALHADO-----
=====
-----TABELA DE COMPARACAO DO TEMPO DE EXECUCAO-----
=====
|| 1000 10000 100000 200000 300000 400000 500000
=====
Bubble || 0 : 0 0 : 0 0 : 38 2 : 6 3 : 57 6 : 12 8 : 55
Select || 0 : 0 0 : 0 0 : 10 0 : 41 1 : 33 2 : 44 4 : 22
Insertion || 0 : 0 0 : 0 0 : 7 0 : 20 0 : 34 0 : 46 1 : 2
MergeSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 1 0 : 1
QuickSort || 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0 0 : 0

Pressione qualquer tecla para continuar. . .
```