Trabalho 1: Algoritmos de ordenação

Caio Cesar Aguiar Alarcon n° 7241109

24 de setembro de 2018

1. Introdução

O trabalho atual tem como propósito comparar diversos algoritmos de ordenação em diversos cenários diferentes variando-se o tamanho dos vetores a serem ordenados e seu nível de aleatoriedade.

2. Procedimento

Primeiramente um programa principal chamado sort foi criado na linguagem C. Ele tem por objetivo permitir, por meio de seus argumentos de chamada, que o usuário possa efetuar testes com cada algoritmo em cada cenário diferente. Além disso o programa permite que o nome para um arquivo de saída seja fornecido como parâmetro para que os dados de saída sejam nele anexados. Permite ainda que o usuário escolha o tamanho, o nível de aleatoriedade e a seed a ser usada pelo algoritmo pseudoaleatório da função rand.

Em seguida, cinco testes foram feitos para cada um dos quatro níveis de aleatoriedade, para cada um dos cinco tamanhos diferentes de vetor, para cada um dos oito algoritmos de ordenação diferentes totalizando oitocentos testes ao todo. Para ser possível automatizar o processo, um programa, também na linguagem C, chamado criascript foi criado com o objetivo de gerar um script na linguagem shell script chamado testatudo.sh que fosse capaz de executar todos os testes paralelamente. Como todos os testes seriam executados ao mesmo tempo usar o time como seed seria inútil, por este motivo o programa criascript também cumpriu o papel de gerar os seeds para cada teste garantindo que não haveria repetição de vetores. Além disso o arquivo testatudo.sh foi salvo e com isso é possível replicar os testes obtendo exatamente os mesmos dados que foram usados neste trabalho. Os testes foram executados em um servidor de computação em nuvem executando Debian e duraram aproximadamente dezesseis horas rodando em uma máquina de oito CPUs. O arquivo com os resultados foi salvo como saída.txt.

Os dados brutos contidos no arquivo saída.txt foram tratados utilizando o Excel e com a ajuda de uma macro personalizada os quatro gráficos, um para cada cenário, foram criados.

Notou-se, ao tratar os dados, que faltavam cinco linhas no arquivo saída.txt, havia apenas setecentas e noventa e cinco em vez das oitocentas linhas esperadas. Ao investigar o problema notou-se que a execução do algoritmo QuickSort havia falhado ao

tentar ordenar o vetor de um milhão de elementos. O algoritmo apresentava o erro de estouro da pilha. Por ser um algoritmo recursivo, é provável que haja um limite para o tamanho do vetor devido à limitação do tamanho da pilha fornecido pelo sistema. Uma possível solução para isso é usar a versão iterativa do QuickSort em vez da recursiva.

3. Discussão dos resultados

O cenário aleatório revela que o HeapSort, intercalação e QuickSort são os mais eficientes, sendo o menos eficiente o Bolha Coquetel:

	100		1000		10000		100000		1000000	
Alg	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib
Bolha	4819,00	4312,00	498363,60	491346,40	49988751,20	49982377,60	4999859338,00	4997819020,40	499998380153,60	500119709482,40
BolhaCo	5266,80	4612,00	507691,80	494626,00	49867012,80	49860167,20	5013049869,00	5000537897,20	500241099758,40	500296689652,40
BolhaSe	4750,00	4749,20	495330,40	493949,60	49955738,20	50007160,40	4999310184,00	4999440699,60	499993347561,80	499780312305,60
heapSort	1184,00	1084,00	19073,60	18073,60	258378,80	248378,80	3249712,00	3149712,00	39097247,60	38097247,60
Insercao	2197,80	4395,60	247239,20	494478,40	25009287,00	50018574,00	2495909647,00	4991819294,00	249990116906,40	499980233812,80
Intercala	294,40	1344,00	4427,20	19952,00	61297,80	267232,00	776626,80	3337856,00	9394872,80	39902848,00
quickSort	827,20	1446,00	13032,40	19842,80	177306,60	239942,80	2215445,80	2887353,60	27233234,80	33988472,80
Selecao	4950,00	198,00	499500,00	1998,00	49995000,00	19998,00	4999950000,00	199998,00	499999500000,00	1999998,00

Tabela 1: Cenário aleatório

No cenário quase inversamente ordenado o HeapSort, o Seleção e o Intercalação se destacam, enquanto o Bolha Coquetel revela eficiência muito baixa especialmente em vetores maiores:

	100		1000		10000		100000		1000000	
Alg	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib
Bolha	4942,20	7345,60	499488,00	966712,80	49994997,40	99657158,40	4999949993,20	9996567052,40	499999499998,60	999965668816,80
BolhaCo	7880,40	7489,20	980818,20	966504,00	99748024,20	99656848,80	9997700022,00	9996564960,40	999978200020,80	999965677659,60
BolhaSe	4915,00	7480,40	499491,00	966464,40	49994971,00	99656160,80	4999949968,80	9996570506,40	499999499961,40	999965672625,20
heapSort	1192,80	1092,80	17785,60	16785,60	242979,20	232979,20	3092464,00	2992464,00	37612523,60	36612523,60
Insercao	3765,20	7530,40	483497,20	966994,40	49828695,40	99657390,80	4998283268,60	9996566537,20	499982836124,20	999965672248,40
Intercala	233,20	1344,00	2691,00	19952,00	28274,20	267232,00	279134,40	3337856,00	2765165,60	39902848,00
quickSort	676,20	832,00	35892,20	41165,60	3367279,00	3835436,00	343294936,00	389718670,00	34334608865,60	39062146202,40
Selecao	4950,00	198,00	499500,00	1998,00	49995000,00	19998,00	4999950000,00	199998,00	499999500000,00	1999998,00

Tabela 2: Cenário quase inversamente ordenado

Já no cenário quase ordenado o péssimo desempenho do QuickSort é o que se destaca, tendo ele falhado em decorrência de um estouro de pilha para o vetor de um milhão de elementos. O algoritmo de seleção também apresenta baixa eficiência nesse caso. Importante notar também que o HeapSort não obteve um bom desempenho nesse caso. O algoritmo mais eficiente para este cenário é o de inserção enquanto o restante apresenta desempenho semelhante.

	100		1000		10000		100000		1000000	
Alg	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib
Bolha	726,60	277,20	8954,80	2778,00	87956,80	28115,60	939951,00	280324,40	9999945,00	2804308,40
BolhaCo	613,80	280,80	6993,00	2831,20	81991,80	28112,00	899991,00	280285,20	9399990,60	2804416,40
BolhaSe	583,80	266,00	7555,60	2830,00	85126,20	28125,60	929271,40	280449,20	9891622,20	2804176,00
heapSort	1364,00	1264,00	20340,00	19340,00	272554,80	262554,80	3398664,00	3298664,00	40552854,80	39552854,80
Insercao	139,80	279,60	1390,20	2780,40	14072,80	28145,60	140383,20	280766,40	1401635,20	2803270,40
Intercala	314,20	1344,00	4730,60	19952,00	65212,40	267232,00	814209,00	3337856,00	9753642,80	39902848,00
quickSort	2123,60	4209,20	209291,80	418307,20	20856867,20	41711141,20	2092407530,00	4184788182,80		
Selecao	4950,00	198,00	499500,00	1998,00	49995000,00	19998,00	4999950000,00	199998,00	499999500000,00	1999998,00

Tabela 3: Cenário quase ordenado

Por último o cenário dos repetidos. Nesse caso os mais eficientes são o Heapsort, o de intercalação e o QuickSort. Algo interessante de observar também nesse caso é o desempenho do algoritmo de Seleção, que apesar de fazer pouquíssimas atribuições, faz um número bem alto de comparações, empatando com os demais.

	100		1000		10000		100000		1000000	
Alg	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib	Comp	Atrib
Bolha	4819,00	4312,00	498363,60	491346,40	49988751,20	49982377,60	4999859338,00	4997819020,40	499998380153,60	500119709482,40
BolhaCo	5266,80	4612,00	507691,80	494626,00	49867012,80	49860167,20	5013049869,00	5000537897,20	500241099758,40	500296689652,40
BolhaSe	4750,00	4749,20	495330,40	493949,60	49955738,20	50007160,40	4999310184,00	4999440699,60	499993347561,80	499780312305,60
heapSort	1184,00	1084,00	19073,60	18073,60	258378,80	248378,80	3249712,00	3149712,00	39097247,60	38097247,60
Insercao	2197,80	4395,60	247239,20	494478,40	25009287,00	50018574,00	2495909647,00	4991819294,00	249990116906,40	499980233812,80
Intercala	294,40	1344,00	4427,20	19952,00	61297,80	267232,00	776626,80	3337856,00	9394872,80	39902848,00
quickSort	827,20	1446,00	13032,40	19842,80	177306,60	239942,80	2215445,80	2887353,60	27233234,80	33988472,80
Selecao	4950,00	198,00	499500,00	1998,00	49995000,00	19998,00	4999950000,00	199998,00	499999500000,00	1999998,00

Tabela 4: Cenário com muitos elementos repetidos

4. Conclusão

Apesar do resultado de todos os algoritmos ser o mesmo, vários fatores influenciam no desempenho de cada maneira diferente de se ordenar um vetor. Por isso é importante entender de forma abrangente o problema a ser resolvido para que o melhor algoritmo possa ser escolhido dependendo da situação que o ele será submetido.