Relatório Projeto de Algoritmos

Ana Clara Pereira Machado 4820
Leslye Esteves Gomes 3976
Paulo Gabriel Pimenta Gomes 3687

Outubro 04, 2018

1 Introdução e Objetivos

Neste relatório são reportadas análises de tempo de execução dos algoritmos Insertion Sort, Selection Sort, Shell Sort, Merge Sort, Heap Sort e Quick Sort a fim de, compará-los computacionalmente, analisando seus comportamentos perante as diferentes entradas de dados que lhes foram aplicadas.

2 Materiais e Métodos

Os algoritmos estudados neste trabalho foram implementados sobre a linguagem de programação C e avaliados em um hardware composto com um processador intel Core i7-7500U, 1TB HDD Samsung ST1000LM035-1RK172 5400RPM, 8GB de RAM, e Placa de Video Nvidia GeForce 940MX 2GB.

Para avaliar os algoritmos, foram utilizados vários casos de teste (entradas de tamanhos diversos) com 10, 1000, 100000, 100000 e1000000 elementos. Usamos essas entradas em 3 diferentes ordens (ordem aleatória, ordem crescente e ordem decrescente). As entradas foram geradas de forma aleatória de acordo com os arquivos disponibilizados pelo professor Joelson Santos na aula prática 1.

3 Resultados

A unidade de medida de tempo utilizada nas tabelas é a de milissegundos.

Abaixo temos a tabela com os valores de entrada aleatórios:

Algoritmos	**Tamanho da Entrada (n)**				
*****	**10**	**1000**	**10000**	**100000**	**1000000**
Insertion Sort	0,1886501	1,7708427	125,3825013	10021,4288888	1009374,7458513
Selection Sort	0,0007052	1,1763304	115,9884313	11543,6211934	1295695,2927039
Shell Sort	0,0003526	0,1558567	1,9566718	35,5712995	624,7044188
Merge Sort	0,0042314	0,3755371	3,6167928	37,4916518	497,6829536
Quick Sort	0,0003526	0,0003526	1,1100384	14,1819038	159,0546056
Heap Sort	0,0010579	0,1748980	1,8254983	26,5108052	441,8569799

Abaixo temos a tabela com os valores de entrada em ordem crescente:

Algoritmos	Tamanho da Entrada (n)				
*	10	1000	10000	100000	1000000
Insertion Sort	0,0000000	0,0063471	0,0317355	0,3215867	3,3100159
Selection Sort	0,0003526	1,1276692	113,5430323	11424,2053247	1434714,9271246
Shell Sort	0,0003526	0,0384353	0,3413333	4,5607485	91,4656808
Merge Sort	0,0056419	0,2708099	1,9785341	19,8361177	332,8852746
Quick Sort	0,0007052	0,0246832	0,3170027	3,8107321	46,7221075
Heap Sort	0,0007052	0,1583250	1,4450245	17,4686471	248,7106559

Abaixo temos a tabela com os valores de entrada em ordem decrescente:

Algoritmos	Tamanho da Entrada (n)				
*	10	1000	10000	100000	1000000
Insertion Sort	0,0003526	1,0874709	108,4195071	10729,1770836	1157065,2359114
Selection Sort	0,0007052	1,0959337	108,8669781	10802,2527291	1167915,5207201
Shell Sort	0,0007052	0,0444297	0,4365399	5,6443406	63,2757134
Merge Sort	0,0049366	0,1900606	1,7909418	16,4033910	165,3530507
Quick Sort	0,0003526	0,0232727	0,3138291	3,8107321	46,2376117
Heap Sort	0,0007052	0,0878016	1,0765398	12,8786313	147,7052742

4 Conclusões

Com base nos resultados podemos analisar que para entradas aleatórias o quick sort foi o algoritmo com melhor desempenho, com destaque para o shell sort que também obteve ótimos resultados. Observamos também que o insertion sort foi o pior algoritmo até a entrada de tamanho 10000, a partir da entrada com 100000 números o pior algoritmo passou a ser o selection sort.

Com as entradas ordenadas o algoritmo com o melhor custo de execução foi o insertion sort, seu funcionamento condiz perfeitamente com seu comportamento, já que o número de comparações dele é menor que os demais e com o vetor ordenado ele não movimenta nada. Os algoritmos com os piores desempenhos é o selection sort, seguido pelo heap sort. O heap sort tem esse tipo de resultado graças a seu comportamento que mesmo o vetor ordenado ele bagunça tudo para o ordenar novamente.

Já para os algoritmos em ordem decrescente o quick sort obteve os melhores resultados referente a tempo, seguido pelo shell sort, novamente observamos a vantagem de ordenar grandes blocos primeiro, o merge sort obteve um resultado mediano, pois a pilha de elementos fica muito grande, heap sort tem um custo bem parecido em todos os casos, pois ele basicamente faz o mesmo independente da ordem das entradas, o que faz ele variar mesmo é o tamanho das entradas. Insertion e Selection sort obtiveram resultados muito pouco atrativos computacionalmente.

Abaixo podemos ver detalhadamente a media de cada algoritmo em segundos

Insertion Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media**
10	0,19	0	0	0,06
1000	1,77	0,01	1,09	0,95
10000	125,38	0,03	108,42	77,94
100000	10021,43	0,32	108,42	3376,72
1000000	1009374,75	3,31	10729,18	340035,75
MEDIA	203904,7	0,73	2189,42	68698,29

Selection Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media**
10	0	0	0	0
1000	1,18	1,13	1,1	1,13
10000	113,54	113,54	108,87	111,98
100000	11543,62	11424,21	10802,25	11256,69
1000000	1295695,29	1434714,93	1167915,52	1299441,91
MEDIA	261470,73	1446253,8	1178827,74	1310811,72

Shell Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media total**
10	0	0	0	0
1000	0,16	0,04	0,04	0,08
10000	1,96	0,34	0,44	0,91
100000	35,57	4,56	5,64	15,26
1000000	624,7	63,28	63,28	250,42
MEDIA	132,48	68,22	69,4	266,67

Merge Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media total**
10	0	0,01	0	0
1000	0,38	0,27	0,19	0,28
10000	3,62	1,98	1,79	2,46
100000	37,49	19,84	16,4	24,58
1000000	497,68	332,89	165,35	331,97
MEDIA	107,83	354,98	183,74	359,3

Quick Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media total**
10	0	0	0	0
1000	0	0,02	0,02	0,02
10000	0,31	0,32	0,31	0,31
100000	3,81	3,81	3,81	3,81
1000000	46,72	46,72	46,24	46,56
MEDIA	10,17	10,18	10,08	10,14

Heap Sort				
Entradas	**Aleatório**	**Crescente**	**Decrescente**	**Media total**
10	0	0	0	0
1000	0,17	0,16	0,09	0,14
10000	1,83	1,45	1,08	1,45
100000	26,51	17,47	12,88	18,95
1000000	441,86	248,71	12,88	234,48
MEDIA	94,07	267,78	26,92	255,02

O que podemos concluir com esse trabalho é que cada algoritmo terá uma particularidade que pode ser bom o ruim e que dependerá muito da experiência de quem os aplica para aproveitar ao máximo essa peculiaridade de cada um.

Em relação ao tempo alguns os mais demorados foram o selection e o insertion sort (apesar de quando a entrada está ordenada o insertion tem o menor tempo).

Já com relação a memória usada, nenhum algoritmo chegou a usar mais que 20% da memória total da máquina, para o tamanho da entrada, ou seja,
a memória para entradas de dados desse tamanho nada afetou.