Trabalho Ordenação

Carlos Mello[1] **, Arthur Przygocki**[2]

[1]PUCPR - Pontifícia Universidade Católica do Paraná.

Resumo. *Esse seria o relatorio referente ao TDE 3 de Resolução em problemas estruturados em Computação.*

1. Algoritmos

Escolhemos usar os algoritmos **Insert-Sort**, **Bubble-Sort**, **Merge-Sort**, **Shell-Sort**.

1.1. Insert-Sort

O algoritmo Insert-Sort ordena um vetor através de percorrer a lista a partir do segundo elemento(tudo que está a esquerda é considerado como ordenado), e então compara o número atual com os elementos até ele encontrar um número maior que ele ou até encontrar a sua posição.

Insert Sort-50 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
31000	617	49
28700	609	49
28500	630	49
26500	560	49
26200	533	49
Média Total		
28180.0	589.8	49.0

Insert Sort-500 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
2828300	60832	499
471400	60650	499
367100	65263	499
381600	65019	499
176700	58475	499
Média Total		
845020.0	62047.8	499

Insert Sort-1000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
2507000	248859	999
265800	250615	999
256200	247244	999
401100	257399	999
98700	250413	999
Média Total		
705760.0	250906.0	999

Insert Sort-5000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
5246400	6232306	4999
8999200	6229319	4999
3654800	6243128	4999
3760900	6197534	4999
3647000	6221187	4999
Média Total		
5061660.0	6224694.8	4999

Insert Sort-10000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
10983200	24796214	9999
9087300	25080854	9999
9248100	25070469	9999
9255000	25117785	9999
8985900	24794269	9999
Média Total		
9511900.0	2.49719182E7	9999

1.2. Bubble-Sort

O algoritmo Bubble-Sort tem como objetovo ordenar um vetor através de comparação e troca. Esse algoritmo vai pegar um valor ”x”e comparar com o elemento seguinte do vetor, caso ”x”seja maior do que o elemeno seguinte, será feita a troca com esse elemento, caso ”x”seja menor do que o elemento seguinte, os valores não mudam de lugar.

Bubbles Sort-50 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
88499	617	1274
76500	609	1274
75599	630	1274
86000	560	1274
70701	533	1274
Média Total		
79459.8	589.8	1274

Bubbles Sort-500 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
4390400	60832	125249
1040200	60650	125249
1280100	65263	125249
1147800	65019	125249
737100	58475	125249
Média Total		
1719120.0	62047.8	125249

Bubble Sort-1000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
8692100	248859	500499
1427700	250615	500499
1417400	247244	500499
1857000	257399	500499
1521800	250413	500499
Média Total		
2983200.0	250906.0	500499.0

Bubble Sort-5000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
28734700	6232306	12502499
27627900	6229319	12502499
32211000	6243128	12502499
37166000	6197534	12502499
30462900	6221187	12502499
Média Total		
31240500	6224694.8	12502499

Bubble Sort-10000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
105696401	24796214	50004999
137713700	25080854	50004999
148289100	25070469	50004999
144159700	25117785	50004999
158343900	24794269	50004999
Média Total		
138840560,2	24971918,2	50004999

1.3. Merge-Sort

O algoritmo Merge-Sort procura ordenar um vetor atraves de ir dividindo ele da maneira mais uniformimente possivel e então comparando os valores que sobraram do vetor para ver qual é menor. Depois que o menor é encontrado ele junta os dois valores com o menor na primeira posição e vai juntando todos os outros valores juntos.

Merge Sort-50 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
157600	219	98
60000	220	98
68600	216	98
59100	221	98
74000	220	98
Média Total		
83860.0	219.2	98

Merge Sort-500 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
828700	3858	998
212000	3856	998
156900	3867	998
167400	3849	998
174800	3862	998
Média Total		
307960.0	3858.4	998

Merge Sort-1000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
532300	8706	1998
195800	8715	1998
139500	8723	1998
143000	8658	1998
144600	8725	1998
Média Total		
231040.0	8705.4	1998

Merge Sort-5000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
1620300	55233	9998
1552100	55256	9998
1757800	55278	9998
1341800	55302	9998
1163400	55219	9998
Média Total		
1487080.0	55257.6	9998

Merge Sort-10000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
2067600	120522	19998
1347000	120520	19998
1236700	120514	19998
1299000	120466	19998
1070400	120568	19998
Média Total		
1404140.0	120518.0	19998

1.4. Shell-Sort

Shell Sort é um algoritmo de ordenação que divide o vetor em subgrupos, aplica ordenação interna em cada subgrupo e gradualmente, mescla esses subgrupos à medida que os incrementos diminuem.

Shell Sort-50 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
227100	617	142
55500	609	142
53299	630	142
48601	560	142
48601	533	142
Média Total		
64500	533	142

Shell Sort-500 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
2858100	60832	2457
311100	60650	2457
245900	65263	2457
270200	65019	2457
245400	58475	2457
Média Total		
786140.0	62047.8	2457.0

Shell Sort-1000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
4356300	248859	5457
719601	250615	5457
669900	247244	5457
646001	257399	5457
832099	250413	5457
Média Total		
1444780.2	250906	5457

Shell Sort-5000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
16251500	6232306	35084
8423000	6229319	35084
12368700	6243128	35084
11105600	6197534	35084
12402400	6221187	35084
Média Total		
12110240	6224694.8	35084.0

Shell Sort-10000 Elementos		
Tempo(Nano)	Número de Trocas	Iterações
27553801	24796214	75243
30904299	25080854	75243
32723901	25070469	75243
24964599	25117785	75243
25965901	24794269	75243
Média Total		
28.422.500,2	24.971.918,2	75243.0

2. Análise

2.1. Insert

O algoritmo Insert-Sort seria um mais tradicional, fazendo troca de elementos e comparações com um determinado elemento, sendo assim ele acaba se tornando um dos algoritmos mais lentos para a ordenação de um vetor realizando um grande número de trocas.

2.2. Bubble

O algoritmo Bubble-Sort é um algorítmo bem simples,trocando de lugar dois valores conforme percorre o array. Conforme o número de elementos foram aumentando, o tempo de execução foi o maior dentre todos os outros algoritmos e o número de iterações sempre foi maior, no último teste com 10000 elementos, comparados com os outros algoritmos, teve mais de 500 vezes mais execuções.

2.3. Merge

O algoritmo Merge-Sort utiliza a recursividade para dividir a lista em 2 e assim deixar o trabalho de comparação mais leve, por causa disso ele utiliza bastante as iterações(sendo o dobro do tamanho do vetor menos dois) e por ele dividir o seu trabalho ele acaba sendo um algoritmo rápido e eficiente embora ele acaba sendo um pouco mais díficil de implementar e entender.

2.4. Shell

O algoritmo Shell-Sort é um algoritmo que funciona dividindo o array em vários subarrays e ordenando-os com o algoritmo de inserção. Comparado com o Insert-Sort, com poucos elementos é mais eficiente. Algo importante para destacar é que a média de número de trocas entre Shell-Sort e Insert-Sort é muito parecida e as vezes igual.