UNIVERSIDADE FEDERAL DE MATO GROSSO
Campus Universitário do Araguaia - CUA
Instituto de Ciências Exatas e da Terra - ICET
Curso de Bacharelado em Ciência da Computação

Relatório de análise dos métodos de ordenação da disciplina de Estrutura de Dados 1

Discentes: Fernanda Lima de Souza, Guilherme da Silva Ferraz

Docente: Dr. Ivairton Monteiro Santos

Barra do Garças - MT

2024

Análise dos métodos de ordenação

Método de ordenação: **Bubble Sort**		
Entrada	Número de Trocas	Tempo (s)
1000 – arq1.txt	251939	0.058000
1000 – arq2.txt	250570	0.034000
1000 – arq3.txt	243858	0.031000
1000 – arq4.txt	255245	0.064000
1000 – arq5.txt	257926	0.037000
Média:	**251907.6**	**0.0448**
10 000 – arq1.txt	24987215	2.523000
10 000 – arq2.txt	24771353	2.636000
10 000 – arq3.txt	25085143	2.555000
10 000 – arq4.txt	24887834	2.437000
10 000 – arq5.txt	24890590	2.805000
Média:	**69512862.4**	**2,5898**
100 000 – arq1.txt	2567996030	261.075000
100 000 – arq2.txt	2574667053	271.403000
100 000 – arq3.txt	2571454263	248.009000
100 000 – arq4.txt	2574721612	261.187000
100 000 – arq5.txt	2573497598	266.023000
Média:	**2572467311.2**	**261.5394**

O Bubble Sort é um algoritmo de ordenação simples que percorre repetidamente a lista, comparando elementos adjacentes e trocando-os se estiverem fora de ordem. O processo continua até que nenhuma troca seja necessária, indicando que a lista está ordenada.

Ao analisar os resultados dos testes do Bubble Sort, torna-se evidente sua baixa eficiência, especialmente com conjuntos de dados maiores. Em média, levou 0,0448 segundos para ordenar cinco arquivos com 1000 valores, 2,5898 segundos para ordenar 10000 valores e 261,5394 segundos (aproximadamente 4,35 minutos) para ordenar 100000 valores.

Além disso, o número de trocas é consideravelmente alto, pois o Bubble Sort compara cada par de elementos adjacentes da lista em cada iteração até que todos estejam em suas posições corretas. Essa alta complexidade de trocas torna o Bubble Sort menos eficiente em comparação com algoritmos de ordenação mais avançados. Em média, foram realizadas 251907,6 trocas ao ordenar 1000 valores, 69512862.4 trocas ao ordenar 10000 valores e 2572467311.2 trocas ao ordenar 100000 valores

Método de ordenação: **Shell Sort**		
Entrada	Número de Trocas	Tempo (s)
1000 – arq1.txt	4307	0.001581
1000 – arq2.txt	4298	0.001595
1000 – arq3.txt	4371	0.001100
1000 – arq4.txt	4303	0.001298
1000 – arq5.txt	4263	0.001145
Média:	**4308**	**0.0013438**
10 000 – arq1.txt	77417	0.136810
10 000 – arq2.txt	77406	0.143279
10 000 – arq3.txt	77452	0.139098
10 000 – arq4.txt	77913	0.157556
10 000 – arq5.txt	77377	0.130927
Média:	**77513 trocas**	**0.141534 segundos**
100 000 – arq1.txt	1062173	15.768346
100 000 – arq2.txt	1063871	15.219912
100 000 – arq3.txt	1062567	15.258156
100 000 – arq4.txt	1062924	16.047577
100 000 – arq5.txt	1062131	16.064180
Média:	**1062733.6**	**15.6716342**

O Shell Sort é um método de ordenação seleciona um intervalo inicial para dividir o conjunto de dados em subgrupos menores. Em seguida, aplica o algoritmo de ordenação por inserção a cada subgrupo. Após a conclusão de uma passagem de ordenação, o intervalo é reduzido, e o processo é repetido até que o intervalo seja igual a 1, tornando o último passo equivalente a uma ordenação por inserção convencional.

Em média, o Shell Sort teve 4308 trocas ao ordenar 1000 valores, 77513 trocas ao ordenar 10000 e 1062733,6 ao ordenar 100000 valores. Já em relação ao tempo de execução deste método, ele obteve em média 0,0013438 segundos ao ordenar 1000 valores, 0,141534 segundos ao ordenar 10000 valores e 156716342 segundos ao ordenar 100000 valores. Ao analisar os resultados obtidos pelo Shell Sort é perceptível a diferença entre ele e o Bubble Sort, visto que o número de trocas e o tempo de execução do Shell Sort é consideravelmente menor do que o método Bubble Sort. Todavia, ele ainda é obsoleto ao lidar com grandes volumes de dados.

Método de ordenação: **Insert Sort**		
Entrada	Número de Trocas	Tempo (s)
1000 – arq1.txt	252939	0.045000
1000 – arq2.txt	251570	0.072000

1000 – arq3.txt	244858	0.035000
1000 – arq4.txt	256245	0.038000
1000 – arq5.txt	258926	0.061000
Média:	**252907.6**	**0.798**
10 000 – arq1.txt	24997215	2.274000
10 000 – arq2.txt	24781353	2.365000
10 000 – arq3.txt	25095143	2.126000
10 000 – arq4.txt	24897834	2.101000
10 000 – arq5.txt	24900590	3.310000
Média:	**24934427**	**2,4352**
100 000 – arq1.txt	2568096030	228.960000
100 000 – arq2.txt	2574767053	216.139000
100 000 – arq3.txt	2571554263	219.684000
100 000 – arq4.txt	2574821612	229.854000
100 000 – arq5.txt	2573597598	214.039000
Média:	**2109099511.2**	**221.7352**

O Insertion Sort é um algoritmo de ordenação simples e eficiente que percorre uma lista de elementos e, para cada elemento, o insere na posição correta, garantindo que os elementos à esquerda estejam ordenados. O Insertion Sort é eficiente para conjuntos de dados pequenos. Sua complexidade de tempo é quadrática no pior caso, tornando-o menos adequado para grandes conjuntos de dados.

Em média, o Insertion Sort teve 252907,6 trocas ao ordenar 1000 valores, 24934427 trocas ao ordenar 10000 e 2109099511,2 ao ordenar 100000 valores. O aumento significativo no número de trocas conforme o tamanho do conjunto de dados cresce é esperado. O Insertion Sort realiza trocas sempre que encontra um elemento que precisa ser movido para a posição correta, e essa operação torna-se mais frequente à medida que o número de elementos aumenta. Já em relação ao tempo de execução deste método, ele obteve em média 0,798 segundos ao ordenar 1000 valores, 2,4352 segundos ao ordenar 10000 valores e 221,7352 segundos ao ordenar 100000 valores. A complexidade de tempo do Insertion Sort é quadrática, o que significa que o tempo de execução aumenta quadráticamente com o tamanho do conjunto de dados. Essa característica torna o Insertion Sort menos eficiente para conjuntos de dados maiores, como evidenciado pelo aumento substancial no tempo de execução ao lidar com 100000 valores.

O Insertion Sort é uma escolha razoável para conjuntos de dados pequenos, como indicado pelo número relativamente baixo de trocas e o tempo de execução razoável ao ordenar 1000 valores. No entanto, à medida que o tamanho do conjunto de dados cresce, o aumento nas trocas e no tempo de execução sugere que algoritmos mais eficientes podem ser preferíveis em cenários com grandes conjuntos de dados.

Método de ordenação: **Merge Sort**		
Entrada	Número de Trocas	Tempo (s)
1000 – arq1.txt	8713	0.000177
1000 – arq2.txt	8709	0.000161
1000 – arq3.txt	8702	0.000130
1000 – arq4.txt	8742	0.000163
1000 – arq5.txt	8712	0.000133
Média:	**8715.6 trocas**	**0.0001528 segundos**
10 000 – arq1.txt	120487	0.001623
10 000 – arq2.txt	120424	0.001894
10 000 – arq3.txt	120400	0.001835
10 000 – arq4.txt	120512	0.001563
10 000 – arq5.txt	120405	0.001829
Média:	**120445.6 trocas**	**0.0017488 segundos**
100 000 – arq1.txt	1536627	0.019709
100 000 – arq2.txt	1536475	0.019481
100 000 – arq3.txt	1536489	0.018737
100 000 – arq4.txt	1536322	0.019723
100 000 – arq5.txt	1536275	0.018384
Média:	**1536437.6 trocas**	**0.0192068 segundos**

O Merge Sort é um método eficiente de ordenação que utiliza o conceito de divisão e conquista. Ele divide a lista de elementos em sub-listas menores, ordena essas sub-listas e, em seguinda, combina-as para obter a lista final ordenada. O Merge Sort possui uma complexidade de tempo de O(n log n), tornando-o eficiente para conjuntos de dados grandes. Ele supera outros algoritmos de ordenação em cenários onde a estabilidade e o desempenho consistente são essenciais. No entanto, seu principal inconveniente é o uso de espaço adicional devido à criação de arrays temporários durante a operação de combinação.

Em média, o Merge Sort teve 8715,6 trocas ao ordenar 1000 valores, 120445,6 trocas ao ordenar 10000 e 1536437,6 trocas ao ordenar 100000 valores. O número de trocas realizado pelo Merge Sort é significativamente menor em comparação com algoritmos como o Bubble Sort, indicando uma eficiência superior na organização dos elementos. O comportamento linear no aumento do número de trocas à medida que o tamanho do conjunto de dados cresce sugere uma consistência na abordagem do algoritmo, independentemente do tamanho do conjunto. Já em relação ao tempo de execução deste método, ele obteve em média 0,0001528 segundos ao ordenar 1000 valores, 0,0017488 segundos ao ordenar 10000 valores e 0,0192068 segundos ao ordenar 100000 valores. O tempo de execução extremamente baixo, na ordem de milissegundos e segundos, mesmo para conjuntos de dados maiores, demonstra a eficiência do

Merge Sort. A relação quase linear entre o aumento do tamanho do conjunto de dados e o tempo de execução destaca a escalabilidade do algoritmo.

O Merge Sort é eficaz em minimizar o número de trocas, tornando-o uma escolha eficiente para conjuntos de dados de diferentes tamanhos. A eficiência do algoritmo em termos de tempo de execução o torna uma opção viável para lidar com grandes quantidades de dados em um tempo aceitável. O desempenho consistente ao longo dos diferentes tamanhos de conjuntos de dados sugere que o Merge Sort é uma escolha robusta em diversas situações.

Método de ordenação: **Quick Sort**		
Entrada	Número de Trocas	Tempo (s)
1000 – arq1.txt	2603	0.000118
1000 – arq2.txt	2538	0.000118
1000 – arq3.txt	2626	0.000116
1000 – arq4.txt	2591	0.000117
1000 – arq5.txt	2534	0.000095
Média:	**2470.4 trocas**	**5,000564 segundos**
10 000 – arq1.txt	34917	0.001343
10 000 – arq2.txt	34719	0.001307
10 000 – arq3.txt	35579	0.001331
10 000 – arq4.txt	35584	0,001219
10 000 – arq5.txt	35626	0.001048
Média:	**35285 trocas**	**0.0012496 segundos**
100 000 – arq1.txt	447002	0.012786
100 000 – arq2.txt	454177	0.012594
100 000 – arq3.txt	445751	0.013071
100 000 – arq4.txt	450586	0.012675
100 000 – arq5.txt	452038	0.012879
Média:	**449910.9 trocas**	**0.012801 segundos**

Assim como o Merge Sort, o Quick Sort adota o princípio da divisão e conquista, tornando-se, portanto, um algoritmo recursivo, sendo eficaz para lidar com conjuntos de dados de tamanhos variados. A escolha inteligente do pivô ajuda a minimizar o número de trocas. Contudo, o modo como o Quick Sort implementa a divisão e conquista difere um pouco da abordagem do Merge Sort.

Enquanto no Merge Sort, a fase de divisão tem um papel mais passivo, deixando a maior parte do trabalho para a etapa de combinação, no Quick Sort, a situação é inversa, com todo o processo crucial ocorrendo durante a fase de divisão. Surpreendentemente, no Quick Sort, o passo de combinação não desempenha nenhum papel significativo. O Quick Sort funciona a partir da posição e seu tempo de execução no pior caso é tão ruim quanto o Insert

Sort: $O(n^2)$. Mas o seu tempo de execução médio é tão bom quanto o do Merge Sort: $O(n \log_2 n)$.

- **Número de Trocas para os arquivos de 1000 valores:** A média de trocas é de 2470.4, o que indica um desempenho moderado na ordenação de conjuntos menores. A média de tempo é de 0.000564 segundos, mostrando uma eficiência notável na ordenação de conjuntos de 1000 elementos.
- **Número de Trocas para os arquivos de 10.000 valores:** A média de trocas é de 35285, um aumento considerável em relação ao conjunto de 1000 elementos, indicando um aumento na complexidade do algoritmo. A média de tempo é de 0.0012496 segundos, ainda assim mantendo uma eficiência notável mesmo para conjuntos maiores.

- **Número de Trocas para os arquivos de 100.000 valores:** A média de trocas é de 449910.9, indicando um aumento significativo na quantidade de trocas para conjuntos ainda maiores. A média de tempo é de 0.012801 segundos, ainda mostrando uma eficiência considerável em relação ao tamanho da entrada.

Em comparação com Bubble Sort**, o** Quick Sort apresenta uma taxa de comparação e troca mais eficiente, é claro. Observa-se ainda que o mesmo mantém uma eficiência notável ainda assim com o aumento do tamanho da entrada. O aumento gradual no número de trocas é esperado, mas o tempo de execução continua sendo relativamente baixo em comparação com outros métodos.

Com base nas observações o Quick Sort é uma ótima escolha para ordenação, especialmente quando se trata de conjuntos de dados de diferentes tamanhos. Sua eficiência média de tempo e capacidade de lidar bem com uma variedade de tamanhos de entrada o tornam uma opção versátil e eficaz.

CONCLUSÃO

Ao analisar os resultados obtidos pelos testes dos métodos de ordenação Bubble Sort, Shell Sort, Insertion Sort, Merge Sort e o Quick Sort, o método que teve opior desempenho foi do método Bubble Sort, visto que na prática ele tem execução lenta mesmo quando comparado a outros algoritmos quadráticos (n^2). Já os melhores foram o Quick sort e o Merge Sort, tendo ambos possuindo um tempo de execução consideravelmente menor ao lidar com grande volume de dados em comparação ao demais métodos de ordenação.

Logo, a escolha de qual método utilizar depende da quantidade de dados a serem processados e das prioridades de otimização, caso quantidade de dados for pequena, o Bubble Sort e o Insertion Sort conseguem lidar de maneira bem. Se a quantidade de dados for mediana, o Shell Sort é uma Sort opção e em relação a grandes quantidades de dados, o Quick Sort e o Merge são mais eficientes- visto que os demais são obsoletos ao lidar com grandes quantidades de dados. A eficiência dos algoritmos varia conforme as necessidades práticas, e a escolha deve ser feita considerando características específicas de cada aplicação.