UNIVERSIDADE ESTADUAL DE SANTA CRUZ

DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CIÊNCIA DA COMPUTAÇÃO

IGOR LIMA ROCHA

ISABELLE SILVA DOS SANTOS DA CRUZ

COMPARAÇÃO ENTRE ALGORITMOS DE ORDENAÇÃO

.

ILHÉUS

2021

RESUMO

Os algoritmos de ordenação são excelentes ferramentas da Ciência da Computação quando o assunto é tratamento de dados. Este estudo tem como objetivo analisar e comparar os algoritmos de ordenação: Bubble Sort, Quick Sort e Merge Sort, a fim de avaliar sua eficiência baseada em seus tempos de execução. Foram executados quatro testes com diferentes tamanhos de vetores ordenados de forma crescente, decrescente e aleatória, em cada algoritmo e os resultados foram coletados e comparados.

Palavras-Chave: ordenação, complexidade, algoritmo.

SUMÁRIO

1. INTRODUÇÃO

Em computação complexidade define-se como o esforço necessário (em tempo e espaço) para executar certa tarefa. A complexidade de algoritmos pode ser dividida em pior, melhor caso e caso médio, que representam os diferentes desempenhos de acordo com o seu custo e podem ser definidos em uma função $f(n)$.

A complexidade pode ser estudada de forma assintótica, que representa o limite de $f(n)$ quando n tende ao infinito, de forma que é possível desconsiderar as constantes e operações que são praticamente nulas. Assim, a complexidade assintótica descreve o comportamento assintótico das operações elementares.

O estudo a seguir comparou, com experimentação, o desempenho dos algoritmos de ordenação Bubble Sort, Merge Sort e Quick Sort. Para isso, foi realizada uma série de testes com arquivos de tamanho e ordenações diferentes:

- Tamanho: 100, 1000, 10000 e 100000.
- Ordenação: aleatória, crescente e decrescente.

Os algoritmos foram comparados quanto a sua eficiência em tempo de execução bem como quanto ao número de comparações e movimentações que são feitas durante o desempenho da aplicação. Os resultados foram coletados e analisados.

2. ALGORITMOS DE ORDENAÇÃO

Algoritmos de ordenação objetivam a manipulação de uma estrutura de dados a fim de ordená-la de forma parcial ou completa. Apesar de existirem um vasto número de opções de algoritmos de ordenação, o presente trabalho comparou apenas três deles.

2.1 Bubble Sort

Considerado um dos algoritmos de ordenação com implementação mais simples, o bubble sort também é o menos eficiente deles. Sua ordenação é feita através de comparações consecutivas de pares de elementos pertencentes à estrutura de dados e trocá-los de posição para que o primeiro seja maior que o segundo, movimentação essa que repete-se até que esteja completamente ordenado.

Quanto à sua complexidade, o Bubble possui classe assintótica de ordem quadrática, sendo n^2 no pior e no caso médio, e n no melhor caso, quando a estrutura já

está ordenada. Sua classe assintótica é $O(n^2)$ para tempo, devido ao *for* aninhado, e $O(1)$ para espaço pois é realizado *in place*.

2.2 Merge Sort

Este algoritmo busca ordenar os elementos pertencentes à estrutura de dados a partir da quebra e mesclagem, usando o método caracterizado na computação como "divisão e conquista", ao tornar um problema tão pequeno que seja simples resolvê-lo.

O Merge Sort é implementado de forma recursiva, que tem grande custo computacional. Por isso, a sua complexidade é de $O(n * log(n))$ para todos os casos.

2.3 Quick Sort

Também adotando uma estratégia de divisão e conquista, o Quick Sort realiza a ordenação a partir da seleção de um pivô, e movimenta os elementos da estrutura de forma que aqueles à esquerda sejam menores que o pivô e os a direita sejam maiores. Esse passo é repetido até que a ordenação seja concluída.

No pior caso a sua complexidade é $O(n^2)$ que ocorre quando o pivô escolhido divide a lista de forma totalmente desbalanceada. O melhor caso, que ocorre quando o pivô divide a lista exatamente em $\frac{n}{2}$, tem complexidade igual à $O(n * log(n))$.

3. RESULTADOS E DISCUSSÕES

Foram utilizadas as linguagens de programação C e C++ e a IDE Visual Studio Code como ambiente de desenvolvimento dos códigos executados nos testes deste estudo. Em relação ao hardware, a máquina operada possui a seguinte especificação:

Versão do driver: 496.76
Processador: Intel(R) Core(™) i9
Frequência: 3.60GHz
Memória RAM: 15.93GB
Sistema operacional: Windows 10.

3.1 Análise do tempo de execução dos algoritmos.

A tabela 1 mostra os valores dos tempos de execução dos três algoritmos para ordenação aleatória, crescente e decrescente dos arquivos com 100, 1000, 10000 e

100000, além do tempo de execução do algoritmo Merge implementado na biblioteca STL do C++.

Tabela 1 - Tempo de execução em segundos

Algoritmos/ Conjunto de Dados	Bubble Sort	Quick Sort	Merge Sort	Merge - STL
A_100	0.000000	0.000000	0.000039	
A_1000	0.002000	0.000000	0.000273	
A_10000	0.184000	0.001000	0.002931	
A_100000	21.395000	0.013000	0.032021	
C_100	0.000000	0.000000	0.000019	
C_1000	0.001000	0.002000	0.000229	
C_10000	0.090000	0.295000	0.001674	
C_100000	8.887000	55.796015	0.019691	
D_100	0.000000	0.000000	0.000029	
D_1000	0.001000	0.002000	0.000136	
D_10000	0.157000	0.193000	0.001354	
D_100000	15.981000	36.906332	0.018984	

Podemos observar na primeira parte da tabela 1, que o algoritmo Bubble Sort teve o pior desempenho quando as chaves estavam ordenadas de maneira aleatória, e que o Quicksort teve o melhor desempenho nesse quesito. O fenômeno observado pode ser justificado pela complexidade de cada um dos algoritmos, visto que quando se trata de ordenação de chaves aleatórias o Quicksort é o mais eficiente.

Mas, ao observarmos a segunda e terceira parte da tabela 1, vemos esse desempenho do quicksort cair drasticamente, tendo o pior deles entre os três. Isso se deve ao fato de que a execução do quicksort é mais lenta quando o algoritmo não é otimizado para a escolha de um pivô aleatório, o que justifica o seu desempenho

piorado quando os testes foram em arquivos grandes ordenados ou completamente desordenados.

Dessa forma, na segunda e terceira parte da tabela, o Merge Sort exibiu o melhor desempenho entre os três algoritmos, pois é o único que tem a mesma complexidade para qualquer um dos casos, mantendo assim certa estabilidade nos resultados.

Em meio a diversas tentativas e persistência da falha, o algoritmo da biblioteca STL não foi possível de ser implementado.

3.2 Análise de comparações e movimentações de chaves.

A tabela 2 mostra a quantidade de comparações e movimentações das chaves dos arquivos ordenados pelos três algoritmos quando a ordenação era aleatória, crescente e decrescente dos arquivos com 100, 1000, 10000 e 100000.

Tabela 2 - Total de comparação de chaves + movimentações de itens.

Algoritmos/ Conjunto de Dados	**Bubble Sort**	**Quick Sort**	**Merge Sort**
A_100	7407	1.134	1.217
A_1000	756.790	16.656	18.709
A_10000	74.878.340	259.605	254.153
A_100000	7.502.640.779	2.978.227	3.205.257
C_100	4.950	6.605	1.028
C_1000	499.500	999.000	15.020
C_10000	49.995.000	99.990.000	202.624
C_100000	4.999.950.000	9.999.900.000	2.522.832
D_100	9.858	5.836	1.017
D_1000	999.000	749.000	14.908
D_10000	99.990.000	74.990.000	198.224

D_100000	9.999.900.000	7.499.900.000	2.483.952

A primeira coisa a ser observada ao analisar a Tabela 2, é que o algoritmo Bubble Sort sempre faz o mesmo número de comparações, independente do arquivo estar ou não ordenado, o que muda é apenas o número de trocas que são realizadas. Quando o arquivo está ordenado nenhuma troca é feita, e quando está completamente desordenado o número de trocas é igual ao número de comparações.

O quicksort segue esta mesma lógica quando se trata de arquivos completamente ordenados, e o motivo é o mesmo que justificava seu baixo desempenho na seção 3.1, pois arrays ordenados classificam o pior caso para esse algoritmo.

Já o mergesort mostra número de trocas e movimentações menor que dos outros dois, mas equivalentes ao seu desempenho, pois o merge faz comparações e movimentações desnecessárias independentemente da ordenação prévia do arquivo, já que o divide em arquivos menores para depois mesclá-los.

Na terceira parte da Tabela 2, podemos observar que em D_100 o número é menor do que esperado, pois existem muitos números repetidos que não são trocados em nenhum dos algoritmos.

4. CONCLUSÃO

Apesar do Bubble Sort ter a implementação mais simples, seus resultados são insatisfatórios. Sua ordenação é lenta, e a maioria dos autores recomendam o uso de um Bubble sort modificado e mais eficiente. Quanto a sua utilidade, este algoritmo só é bem usado para fins educacionais.

O QuickSort apesar de muito eficiente para o uso geral, tem uma implementação mais difícil, e também precisa de otimização para ordenar melhor uma grande quantidade de dados. O uso de um pivô aleatório pode ajudar a evitar os cenários de piores casos como o de um arquivo perfeitamente ordenado, assim como visto nos experimentos das Tabelas 1 e 2.

Quando falamos do MergeSort podemos observar que dentre os três algoritmos foi o que mostrou um desempenho mais estável, mesmo sendo de implementação recursiva. Comparando os diferentes resultados nas tabelas é possível notar também que suas melhores performances são quando as chaves estão ordenadas de forma crescente. Seu ponto negativo é a necessidade de O(n) espaço de memória, o que o faz precisar de um espaço extra para sua implementação.

5. BIBLIOGRAFIA

1. ALLAIN, Alex. Sorting Algorithm Comparison. [*S. l.*], 2019. Disponível em: https://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html. Acesso em: 1 dez. 2021.

2. ANALYSIS of Algorithms: Analysis of different sorting techniques. [*S. l.*], 28 jun. 2021. Disponível em: https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/ Acesso em: 1 dez. 2021.

3. COMPARISON of Sorting Algorithms. [*S. l.*], 19 jul. 2021. Disponível em: https://www.codeproject.com/Articles/5308420/Comparison-of-Sorting-Algorithms. Acesso em: 1 dez. 2021.

4. OLIVEIRA, Alexandre César. ESTRUTURAS DE DADOS: Complexidade. [*S. l.*], 2005. Disponível em: http://www.deinf.ufma.br/~acmo/grad/ED_complexidade_2005.pdf. Acesso em: 1 dez. 2021.

5. SORTING and Searching Algorithms: Quicksort Algorithm. Disponível em: https://www.programiz.com/dsa/quick-sort. Acesso em: 28 nov. 2021.

6. SORTING and Searching Algorithms: Merge Sort Algorithm. [S. l.]. Disponível em: https://www.programiz.com/dsa/merge-sort. Acesso em: 26 nov. 2021.