UMA PROPOSTA PARA INTRODUÇÃO DOS CONCEITOS DE COMPLEXIDADE ALGORÍTMICA: um estudo de caso com alunos ingressantes no curso de Engenharia de Computação

CÂMARA, João Vítor de Oliveira[1,2], SILVA FILHO, Lúcio Rodrigues[1,2], SILVA, Gabriel da[1,3]

[1]Instituto Federal de Minas Gerais (IFMG) - Campus Bambuí,

[2] Aluno do curso de Engenharia de Computação do IFMG – Campus Bambuí, [3] Professor Orientador

joaovitoroliveiracamara@hotmail.com, lucioraiser@hotmail.com,

gabriel.silva@ifmg.edu.br

RESUMO

O presente resumo apresenta os resultados obtidos em um experimento realizado na oferta da disciplina Algoritmos e Estruturas de Dados I, do primeiro período do curso de Engenharia de Computação do IFMG – Campus Bambuí, no primeiro semestre de 2018, o qual abordou uma forma alternativa para a introdução aos Métodos de Ordenação em Memória e a Análise da Complexidade Algorítmica. Historicamente, os alunos costumam apresentar muita dificuldade na assimilação destes conteúdos, principalmente, quando ministrados utilizando apenas a notação assintótica. Assim, como proposta alternativa, estes conteúdos foram ministrados considerando a medição do tempo (em milissegundos) da execução dos algoritmos, antes da discussão na Notação *Big O*. Em seguida, os alunos realizaram um trabalho no qual foram implementados 4 métodos de ordenação (*Insertion Sort, Selection Sort, Bubble Sort, QuickSort*) e a observação dos resultados de sua execução considerando a medição do tempo (em milissegundos), antes da discussão em notação *O*. O professor e os alunos consideraram que os resultados foram interessantes e satisfatórios, os quais são apresentados neste artigo.

Palavras-chave: Algoritmos de ordenação em memória. Complexidade algorítmica. Abordagem alternativa para o ensino.

1 INTRODUÇÃO

Cormen (2002) define que analisar um algoritmo significa prever os recursos de que ele necessitará para sua execução. Esta previsão é feita em tempo de computação e está relacionada ao consumo de memória, ciclos de processador e demais *hardwares* envolvidos.

Ziviani (2004), com base em Knuth (1971), agrupa os algoritmos em 2 grupos quando o assunto é análise de complexidade: 1. Análise de um algoritmo em particular e 2. Análise de uma Classe de Algoritmos. O foco do presente trabalho foi a classe dos Algoritmos de Ordenação em Memória.

A ordenação consiste em reagrupar elementos e posicioná-los em uma ordem específica, com o objetivo de facilitar a recuperação de dados de uma lista. Com o uso do algoritmo de ordenação certo é possível realizar essa tarefa em menor tempo.

Apesar de se ter ciência de que a medição do tempo (relógio) apresenta algumas fragilidades quando comparada ao uso de modelos matemáticos ou modelos de computação genéricos

(ASCÊNCIO e ARAÚJO, 2010), os autores acreditam que, do ponto de vista do processo de ensino-aprendizagem, estas fragilidades não geram prejuízos na abordagem inicial destes assuntos.

Para esse trabalho foram selecionados quatro algoritmos de ordenação, sendo eles, três algoritmos de ordenação simples e um complexo. Os mesmos foram usados para ordenar conjuntos de dados com tamanhos e disposição dos dados distintos. Assim, foi possível abordar, ao mesmo tempo, a influência que tanto o volume, quanto a organização dos dados, causam no processo de ordenação. Os conceitos de pior caso e melhor caso foram apresentados com base nestes conjuntos.

O objetivo do trabalho foi propor uma abordagem diferente para o ensino destes conceitos, uma vez que, historicamente, os alunos ingressantes apresentam dificuldades em assimilá-los pelo ensino da forma tradicional, utilizando-se, diretamente, a notação *O*.

Os resultados obtidos foram considerados interessantes pelos autores e por isso compartilhados neste resumo expandido.

2 METODOLOGIA

Foram selecionados 4 métodos de ordenação para o ensino dos conceitos de Ordenação de dados em memória e Complexidade Algorítmica, sendo 3 deles apresentados aos alunos como Algoritmos Simples (*Insertion Sort, Selection Sort, Bubble Sort*) por não fazerem uso de nenhuma técnica mais complexa e 1 baseado na Técnica de Divisão e Conquista, o *QuickSort*, o qual utiliza recursividade, assunto também abordado na disciplina.

O professor elaborou e disponibilizou conjuntos de dados numéricos inteiros, tanto para utilização nas aulas, quanto para a realização do trabalho pelos alunos, nos quais os dados estavam organizados das seguintes maneiras: 1. Números distintos aleatórios, 2. Números aleatórios com algumas repetições, 3. Números aleatórios distintos ordenados, 4. Números aleatórios distintos quase ordenados e 5. Números aleatórios distintos invertidos.

Além da organização dos números nos conjuntos, as quantidades também variaram. Para cada um dos 5 grupos descritos acima, foram criadas instâncias chamadas de Conjuntos Pequenos, Conjuntos Médios e Conjunto Grande (apenas 1 vetor com 30 mil elementos). Assim, foram utilizados 11 vetores nos experimentos.

Assim, as execuções dos 4 algoritmos de ordenação foram realizadas com todas as combinações dos grupos de dados. Ressalta-se que os conjuntos disponibilizados para realização do trabalho foram diferentes dos usados nas aulas.

Após a apresentação dos conceitos aos alunos, foi disponibilizado o enunciado do trabalho, no qual foi enunciado o que se deveria fazer. O programa deveria oferecer ao usuário um menu para escolha de qual(is) algoritmos desejava executar, bem como sobre qual(is) conjunto(s) de dado(s).

Todo o desenvolvimento foi realizado utilizando-se a o IDE CodeBlocks e a linguagem de programação C++, no paradigma Imperativo, com modularização em procedimentos e funções.

Foi usada a biblioteca **`time.h`** para manipulação do dado de medição do tempo, de modo a conseguir exibir o valor em milissegundos.

A implementação está disponível e pode ser acessada em um repositório do Github pela URL: https://github.com/th3d3rkn3ss/Algoritmos-e-Estruturas-de-Dados.

Os experimentos foram realizados em um computador com processador Intel® Core™ i3-2370M com 2.40GHz e memória RAM 4GB DDR3.

Os resultados foram apresentados na forma de gráficos e tabelas, contendo as características das execuções com cada algoritmo de ordenação e respectivos conjuntos de dados.

3 RESULTADOS E DISCUSSÃO

A interface do programa foi simples, porém funcional, apresentando todas as operações possíveis, conforme definido na metodologia. Inicialmente, o usuário escolhe o vetor a ser usado. Em seguida, qual o algoritmo deseja usar para ordená-lo. Esta operação se repete quantas vezes forem solicitadas.

Para cada uma das combinações de vetores e algoritmos foram coletados os dados, gerando os dados apresentados na Tabela 1.

Tabela 1 – Medições dos tempos (milissegundos) de execução

Vetor	Qtde	Característica	*Insertion Sort*	*Selection Sort*	*Bubble Sort*	*Quick Sort*
1	100	Distintos aleatórios	29,64	52,55	53,22	12,8
2	100	Aleatórios com algumas repetições	33,01	51,2	37,73	16,84
3	100	Aleatórios distintos ordenados	19,83	22,23	39,75	9,43
4	100	Aleatórios distintos quase ordenados	19,83	51,88	30,32	12,8
5	100	Aleatórios distintos invertidos	17,35	49,8	50,53	0,76
6	5000	Distintos aleatórios	0,13	47,16	57,85	0,72
7	5000	Aleatórios com algumas repetições	33,3	54,49	56,93	0,9
8	5000	Aleatórios distintos ordenados	0,13	55,59	58,93	0,76
9	5000	Aleatórios distintos quase ordenados	4,85	55,5	56,24	0,79
10	5000	Aleatórios distintos invertidos	61,62	51,41	52,46	0,77
11	30000	Aleatórios e distintos.	905,08	1667,03	1766,77	4,94

Fonte: Os Autores (2018).

O Gráfico 1 mostra o comportamento das execuções usando os vetores de 1 a 10, os quais contém 100 ou 5000 elementos, com cinco variações (vide Tabela1). Optou-se por esta apresentação pois os tempos de execução variaram em um intervalo próximo de tempo, o que facilita a observação dos resultados. Outro resultado interessante de ser analisado é o apresentado pelo Gráfico 2, o qual descreve os resultados das execuções apenas para o vetor de 30 mil elementos, o que eleva o tempo de execução para uma outra escala maior. Como os valores são muito discrepantes, o gráfico é apresentado numa escala logarítmica, para melhor compreensão.

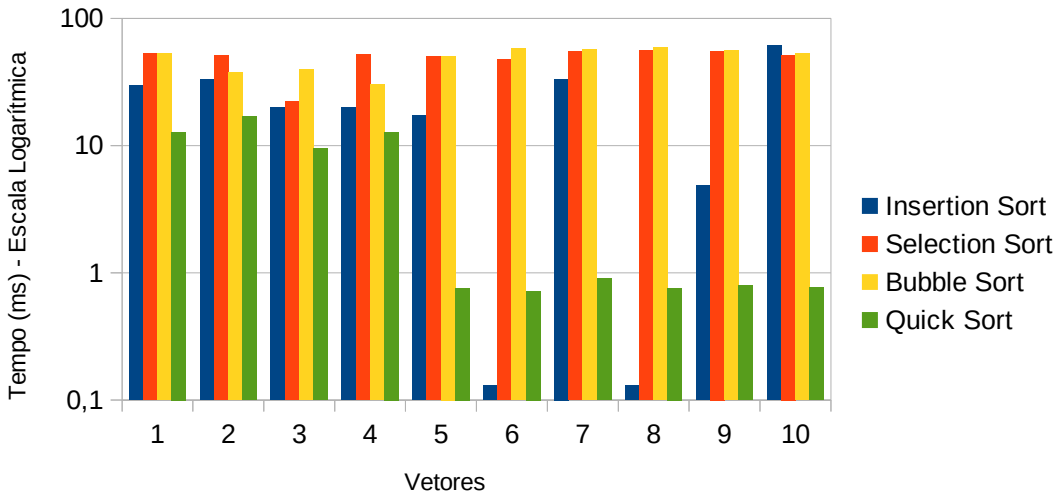

Gráfico 1 – Execuções com vetores de 1 a 10
Fonte: Os Autores (2018)

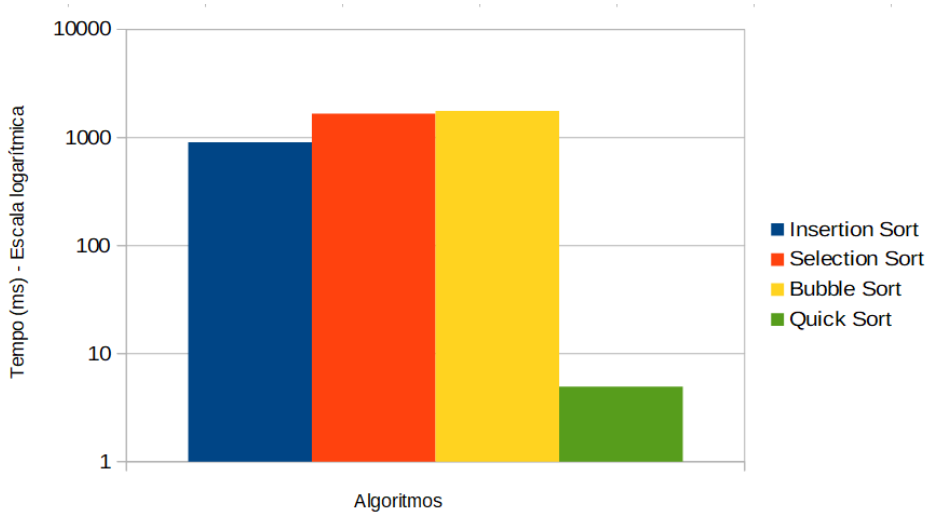

Gráfico 2 – Execuções com vetor de 30 mil
Fonte: Os Autores (2018)

Como mostrado, o comportamento das execuções acompanha o que é definido pela literatura como característica de cada um dos algoritmos usando-se a notação *big O*, conforme mostra a Tabela 2. Após todos os testes realizados, podemos observar que o método de ordenação mais eficiente neste caso foi o *Quick Sort*, exceto nos vetores 6 e 8 onde ele ficou em segundo atrás do método *Insertion Sort*.

Tabela 2 – Tempos Computacionais em *Big O*

Casos	Algoritmos			
	Insertion Sort	***Selection Sort***	***Bubble Sort***	***Quick Sort***
Pior	*O (n²)*	*O (n²)*	*O (n²)*	*O (n²)*
Melhor	*O (n-1)*	*O (n²)*	*O (n)*	*O (n log n)*

Fonte: Ascêncio e Araújo (2010)

Logo após o *Quick Sort* vem o *Insertion Sort*, seguido pelo *Selection Sort* que apresenta uma diferença de 101,89 ms para o *Bubble Sort*. Podemos observar que as características de como estão organizados os dados dentro dos vetores interferem no tempo de ordenação dos métodos, do caso contrário existiria apenas um método de ordenação que seria o mais rápido e eficiente. Essa diferença nas características dos vetores que fez, por exemplo, em dado momento o *Insertion Sort* ser mais eficiente que o *Quick Sort*.

4 CONCLUSÃO

Conforme demonstrado pelos resultados, foi possível analisar junto aos alunos os tempos de execução dos algoritmos tanto em milissegundos, quanto na notação *Big O*. No experimento em específico, as fragilidades nesta abordagem destacadas por alguns autores não comprometeram o estudo, pelo contrário, facilitaram a compreensão por parte dos alunos.

Nesta etapa do trabalho, não foram realizadas análises mais específicas sobre a eficiência da metodologia utilizada. Para o professor da disciplina, a qualidade das discussões apresentadas pelos alunos no trabalho final da disciplina demonstrou que os alunos, de certa maneira, assimilaram bem os conteúdos. Pretende-se, num outro momento, realizar um novo experimento com foco na avaliação da metodologia. Os autores consideram o resultado deste trabalho importante, pois a metodologia proposta pôde contribuir para facilitar o ensino e o aprendizado de um tema que se apresenta como difícil na maioria das ofertas da disciplina.

REFERÊNCIAS

ASCÊNCIO, Ana Fernanda Gomes; ARAÚJO, Graziela Santos de. **Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++**. São Paulo: Pearson Prentice Hall, 2010.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: teoria e prática**. Rio de Janeiro: Elsevier, 2002.

KNUTH, D.E. **Mathematical Analysis of Algorithms**. Procedings of IFIP Congress 71, vol. 1, North Holland, Amsterdam, Holanda, 135-143. 1971.

ZIVIANI, Nivio. **Projeto de algoritmos com implementações em pascal e C.** 2. ed. São Paulo: Pioneira Thomson Learning, 2004.