CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
UNIDADE CONTAGEM

IAGO GONÇALVES MOYSÉS
PEDRO RANGEL FERREIRA DE AGUIAR
SAMUEL HENRIQUE DA CUNHA MACHADO

TRABALHO PRÁTICO 1

Métodos de Ordenação em Java

CONTAGEM - MG
Maio - 2022

IAGO GONÇALVES MOYSÉS
PEDRO RANGEL FERREIRA DE AGUIAR
SAMUEL HENRIQUE DA CUNHA MACHADO

TRABALHO PRÁTICO 1

Métodos de Ordenação em Java

Trabalho formalizado na disciplina de Linguagens e Técnicas de Programação II, que possui como objetivos ampliar o conhecimento da linguagem de programação Java por meio de técnicas de programação, aprender a lidar com coleções de objetos e reconhecer as características e implementação de alguns dos algoritmos de ordenação conhecidos na literatura acadêmica.

Docente: Professor Alisson Rodrigo dos Santos.

CONTAGEM - MG
2022

RESUMO

Este trabalho consiste na realização de um projeto programado na linguagem de programação Java. Estimulando os discentes apresentados a estudarem e implantarem métodos de ordenação, o presente trabalho estimula o crescimento acadêmico na área em questão, uma vez que é algo comumente utilizado no mercado de trabalho. Neste relatório serão apresentados três métodos de ordenação que foram explorados, a saber, *InsertionSort, QuickSort e SelectionSort.* Entretanto, este trabalho não se restringe apenas à implantação dos algoritmos propriamente ditos. Foi realizada uma análise que verificava a quantidade de movimentações apresentadas por cada algoritmo sob circunstâncias diferentes. Estes resultados serão apresentados em uma tabela mais adiante.

PALAVRAS-CHAVE: Métodos de Ordenação, Orientação a Objetos, Programação em Java

LISTA DE FIGURAS

Figura 1 - Exemplo InsertionSort - página 09
Figura 2 - Exemplo QuickSort - página 10
Figura 3 - Exemplo SelectionSort - página 11

LISTA DE TABELAS

Tabela 1: Números de movimentações de cada método de ordenação de acordo com os arquivos - página 12
Tabela 2: Vantagens e desvantagens de cada método de ordenação - página 14

SUMÁRIO

1 **INTRODUÇÃO** ……...……...……..……..07
2 **DESENVOLVIMENTO** ……..……...……..….08
2.1 - INÍCIO……..08
2.2 - MODELO DE APLICAÇÃO……..……..08
2.3 - MÉTODOS DE ORDENAÇÃO……...09
3 **CONSIDERAÇÕES FINAIS** ……..……..……..13
4 **REFERÊNCIAS** ……..……..15

1. INTRODUÇÃO

"Ordenação de Elementos" é uma tarefa que pode ser considerada trivial no mundo da programação, por se tratar de algo que se aplica em diversos casos e em diversos cenários de prestação de serviços.

Contudo, apenas citar "Ordenação de Elementos" não abrange tudo que é englobado nessa tarefa. Assim como a maioria dos problemas da vida cotidiana, acadêmica ou profissional, a ordenação não possui uma solução unânime e benéfica para todos os casos. A abordagem mais correta é discutir acerca dos "Métodos de Ordenação". Como são aplicados, qual a complexidade, casos de uso, suas vantagens e desvantagens. Ambas são perguntas pertinentes que este trabalho busca esclarecer.

A fim de se alcançar este objetivo, foram escolhidos três métodos de ordenação para serem explorados: *InsertionSort, QuickSort e SelectionSort.* Cada um desses algoritmos foi implantado na linguagem de programação Java. Possuindo um "contador" em cada um dos algoritmos, a quantidade de movimentações realizadas é o objeto de estudo do presente trabalho, cujos resultados estarão postos em uma tabela comparativa.

Em uma referência rápida ao processo de codificação, em paralelo com o estudo dos métodos de ordenação, a "Programação Orientada a Objetos" também foi explorada, sendo crucial para a implementação desses métodos.

2. DESENVOLVIMENTO

2.1 - INÍCIO

Inicialmente, foram realizadas pesquisas individuais e coletivas acerca dos métodos de ordenação selecionados, a fim de entender o seu funcionamento e aplicá-los no contexto deste trabalho. Cada integrante ficou responsável pela aplicação de um dos três algoritmos delegados.

2.2 - MODELO DE APLICAÇÃO

O funcionamento do projeto não é de difícil entendimento, e este será pormenorizado no decorrer desta seção.

Primordialmente à implementação de quaisquer dos algoritmos de ordenação, duas classes foram estabelecidas, a saber, a classe Read e a classe Write, que por sua vez herda propriedades da classe Read por meio do mecanismo de herança, característico da Programação Orientada a Objetos.

A classe Read é responsável por ler o arquivo de texto (cujo tamanho variava entre 100, 1000 e 10000 elementos dispostos de diferentes formas) escolhido pelo usuário e transformar seu conteúdo em um ArrayList de strings, retornando o Array ao término dessa operação. O Array retornado será manipulado pelo método de ordenação que também foi definido pelo usuário. O algoritmo será executado em um arquivo java dedicado para cada um, que além de ordenar, deverá registrar a quantidade de movimentações. Ao final, a classe Write irá guardar o ArrayList ordenado em um arquivo csv, que possuirá também o número de movimentos realizados pelo método de ordenação.

É possível perceber que todas essas operações são complicadas de serem gerenciadas individualmente. Para isso, foi criado um arquivo nomeado “index.java”. Este é o único arquivo executável, possuindo o método main e as funções que chamam as outras classes, como verificar o método de ordenação que o usuário definiu e chamar o mesmo, bem como instanciar objetos nas classes Read e Write em funções responsáveis por tratar os arquivos. Tudo isso é disposto no console com menus que auxiliam o usuário na escolha.

Conforme dito anteriormente, em paralelo com a ordenação, os algoritmos também foram responsáveis por verificar o número de movimentações, e estes resultados estão dispostos em uma tabela comparativa.

2.3 - MÉTODOS DE ORDENAÇÃO

2.3.1 - INSERTION SORT

O método Insertion Sort é um método que percorre todo o vetor, da esquerda para a direita, e ordena os elementos à esquerda a medida em que avança. Dado um array, para cada posição, ele irá verificar se o item da esquerda é maior que o próximo. Enquanto o item da esquerda for maior, ele irá fazer a troca das posições dos dois elementos. Possui complexidade $C(n) = O(n)$ no melhor caso e $C(n) = O(n^2)$ no pior. Devido ao fato de o Insertion Sort percorrer todo o vetor para fazer a comparação e ordenação, ele acaba não sendo tão eficiente para ordenação de arrays com muitas entradas, porém é o mais indicado para arrays com poucas entradas, sendo o mais eficiente.

REPRESENTAÇÃO:

Figura 1(GIF): Exemplo InsertionSort
https://upload.wikimedia.org/wikipedia/commons/9/9c/Insertion-sort-example.gif

2.3.2 - QUICKSORT

Apesar de ser das mesmas classes de complexidade do MergeSort e do HeapSort, o QuickSort é um método de ordenação com a prática mais veloz, pois suas constantes são menores. O QuickSort se baseia na eficiente ordenação de divisão por conquista dividindo o problema de ordenação de n elementos em dois problemas menores, portanto utiliza a ideia da recursividade, ou seja, a definição de uma função ou de outro objeto que refere-se ao próprio

objeto sendo definido. Vale mencionar que o pior caso de tempo de execução do QuickSort é tão ruim quanto o tempo os outros métodos de ordenação, como SelectionSort e InsertionSort: ***O(n²)***. Apesar disso, o QuickSort tem desempenho melhor que todos outros métodos de ordenação e é, portanto, o melhor ordenador do que o de inserção e seleção.

O QuickSort se baseia na ideia de particionamento por meio da escolha de alguma posição do array, sendo chamada de pivot e a realização de comparações entre os elementos auxiliares do array (mencionados mais a frente) e o pivot para no final chegar a um array ordenado. No final das operações os elementos menores que o pivô ficarão à esquerda e elementos maiores que o pivot na direita. São utilizados também dois elementos que ficarão responsáveis por apontar a uma posição do array, um que começa primeiramente na primeira posição e um outro que ficará na última posição. O elemento i, por exemplo, que começará na primeira posição, realizará comparações com o pivot para verificar se o elemento presente no lugar onde ele indica é menor que o pivot. Se essa questão for verídica haverá um incremento no elemento i e essa verificação acontecerá por meio da repetição(while) até que seja negada. A mesma utilização acontece com o elemento que ficará na última posição primeiramente. Chamaremos ele de j. Quando a condição dele ser maior que o pivot se mostrar verdadeira, o j será decrementado até que a condição seja negada. Após isso será preciso verificar se o i é maior ou igual ao j, se isso for verdade, o programa retorna o elemento j que será o pivot. Se não, os elementos presentes na posição i e j serão trocados. Dessa maneira o problema será dividido em dois. O primeiro problema consiste na ordenação dos elementos à esquerda do pivot e a segunda consiste na ordenação dos elementos à direita do pivot. E isso acontecerá até que todo o array seja ordenado.

REPRESENTAÇÃO:

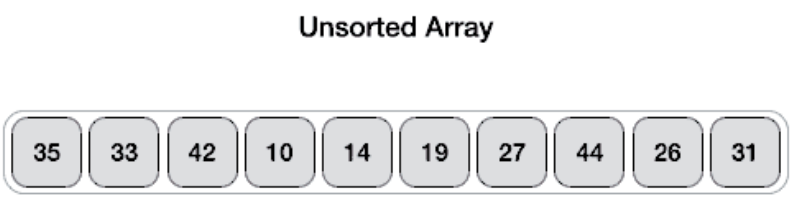

Figura 2(GIF): Exemplo QuickSort

https://shre.ink/6nS

2.3.3 - SELECTION SORT

Em uma abordagem simples, o Selection Sort é provavelmente o modo como a maioria das pessoas ordenariam uma lista de elementos desordenados. Isso se deve ao fato de sua rotina ser extremamente básica, isto é, selecionar o menor elemento da sequência e colocar ele na primeira posição do array. A ideia é executar várias vezes esses dois passos para ordenar um array. Para ser exato, se executarmos N vezes esses dois passos em um array, controlando os índices em que os passos são executados, o resultado é a ordenação completa do mesmo. Ou seja, esse algoritmo é de complexidade quadrática $O(N^2)$. É composto por dois laços, um laço externo e outro interno. O laço externo serve para controlar o índice inicial e o interno percorre todo o vetor.

REPRESENTAÇÃO:

8	5	2	6	9	3	1	4	0	7

Figura 3(GIF): Representação Selection Sort
https://codingconnect.net/wp-content/uploads/2016/09/Selection-Sort.gif

Tabela 1: Números de movimentações de cada método de ordenação de acordo com os arquivos

	Movimentações		
Arquivos	InsertionSort	QuickSort	SelectionSort
100 nomes aleatórios	2410	146	92
100 nomes crescentes	0	0	0
100 nomes decrescentes	5049	50	50
1000 nomes aleatórios	246101	2304	994
1000 nomes crescentes	0	1	1
1000 nomes decrescentes	500498	501	821
10000 nomes aleatórios	25071989	30701	10021
10000 nomes crescentes	0	18	18
10000 nomes decrescentes	50004981	5018	51799

3. CONSIDERAÇÕES FINAIS

Um dos objetivos deste trabalho é entender a diversidade de formas existentes para se realizar uma tarefa trivial, que nesse caso, se trata da ordenação de elementos. Durante a implementação, se tornou claro as qualidades e defeitos de cada método, e estas estão listadas na tabela abaixo.

Entendendo os casos de uso para cada método, os integrantes estão mais aptos a aplicarem os mesmos futuramente, seja durante estudos ou até em uma aplicação existente no mercado.

Durante o desenvolvimento, foram diagnosticados problemas devido a condições não ponderadas no momento da codificação, mas com as devidas correções, o programa se comporta da maneira que deveria.

Pensando em uma possível melhoria futura, um algoritmo capaz de tratar diversos tipos de entrada seria o ideal. Não só referente ao tipo de dado, mas o próprio arquivo recebido. Isso permitiria o projeto se tornar em um projeto escalável, extremamente importante no nicho da programação.

Tabela 2: Vantagens e desvantagens de cada método de ordenação

	PRÓS	CONTRAS
InsertionSort	• Simplicidade de implementação. • Bom desempenho em listas pequenas. • Requerimento de espaço mínimo, pois é local. • Duas vezes mais rápido que BubbleSort e normalmente mais rápido que SelectionSort.	• Não possui um desempenho bom com listas grandes. • É considerado um algoritmo lento. • $O(n^2)$ comparações, ordenação quadrática.
QuickSort	• Considerado um dos melhores métodos de ordenação. • Grande eficiência. • O pior caso de desempenho é raro. • É eficiente em listas de qualquer tamanho. • Tempo de execução baixo. • Bastante eficiente em listas desordenadas	• Seu pior desempenho é similar ao desempenho dos outros métodos mais simples: $O(n^2)$. • Implementação delicada e difícil. • Um pequeno engano pode levar a efeitos indesejados na ordenação.
SelectionSort	• Simplicidade de implementação. • Eficiência em listas pequenas. • Adequado para listas pequenas em ordem aleatória. • Possui menor quantidade de movimentações.	• $O(n^2)$ comparações, ordenação quadrática. • SelectionSort é mais lento que o InsertionSort. • Baixa eficiência em listas grandes.

4. REFERÊNCIAS

BRUNET, João. **Ordenação por Comparação:** InsertionSort. Campo Grande, 27 out. 2019. Disponível em: <https://joaoarthurbm.github.io/eda/posts/insertion-sort/>. Acesso em: 15 maio 2022.

BRUNET, João. **Ordenação por Comparação:** QuickSort. Campo Grande, 27 out. 2019. Disponível em: <https://joaoarthurbm.github.io/eda/posts/quick-sort/>. Acesso em: 15 maio 2022.

BRUNET, João. **Ordenação por Comparação:** SelectionSort. Campo Grande, 27 out. 2019. Disponível em: <https://joaoarthurbm.github.io/eda/posts/selection-sort/>. Acesso em: 15 maio 2022.

Bruno. **Algoritmos de Ordenação**: Análise e Comparação. [*S. l.*], 7 jun. 2013. Disponível em: <https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>. Acesso em: 15 maio 2022.

CANAL DO CÓDIGO. **QuickSort simples e sem complicações - Canal do Código**. 2018. (12m13s). Disponível em: <https://www.youtube.com/watch?v=PrR3nfq9wSY>. Acesso em: 03 mai. 2022.

FORTES, Reinaldo. **Ordenação**: Bubble, Selection e InsertionSort. Ouro Preto, MG. BCC202. jan. 2016. PDF. 38 slides. color. Disponível em: <http://www.decom.ufop.br/anascimento/site_media/uploads/bcc202/aula_12_-_bubblesort-selectionsort-insertionsort.pdf>. Acesso em: 14 mai. 2022.

GUARINO, Leandro. **Aula 13 - Selection Sort - Estruturas de Dados com Java**. 2020. (15m07s). Disponível em: <https://www.youtube.com/watch?v=5SccjA4afcM>. Acesso em: 10 mai. 2022

GUARINO, Leandro. **Aula 17 - Algoritmo Quick Sort - Estruturas de Dados com Java**. 2020. (13m20s). Disponível em: <https://www.youtube.com/watch?v=BW2qE_7HNG8>. Acesso em: 05 mai. 2022

SOUZA, Jairo Francisco de. **Método InsertionSort:** Estrutura de Dados II. Juiz de Fora, MG. UFJF. 2 dez. 2009. PDF. 14 slides. color. Disponível em: <https://www.ufjf.br/jairo_souza/files/2009/12/2-Ordenação-InsertionSort.pdf>. Acesso em: 7 mai. 2022.

WANDY, Joe. As vantagens e desvantagens dos algoritmos de ordenação. eHowBrasil, nov. 2021. Disponível em: <https://www.ehow.com.br/vantagens-desvantagens-algoritmos-ordenacao-info_16277/>. Acesso em: 11 mai. 2021