INSTITUTO DE INFORMÁTICA
UFG

Relatório - Automação de Build com o Maven

Fernando Maltez Bezerra - 202302537 - Engenharia de Computação

Pedro Cézar Silva Ferreira - 202302579 - Engenharia de Computação

Link do projeto: https://github.com/ISO53/Sorting-Algorithms/tree/main

- **Dependências, plugins e configurações específicas**

A partir do arquivo pom.xml, observa-se que existe uma única dependência listada no seguinte trecho, a biblioteca flatlaf na versão 3.1.1:

```
<dependency>
    <groupId>com.formdev</groupId>
    <artifactId>flatlaf</artifactId>
    <version>3.1.1</version>
</dependency>
```

Além disso, o único plugin configurado é o maven-jar-plugin, na versão 3.3.0. Esse plugin é usado para criar um arquivo JAR para o projeto. A configuração específica para este plugin está relacionada à inclusão de informações no manifesto do JAR, como a classe principal a ser executada, veja:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.3.0</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>

<mainClass>com.mycompany.sortingalgorithms.SortingAlgorithms</mainC
lass>
            </manifest>
        </archive>
    </configuration>
</plugin>
```

Por fim, tratando de configurações específicas, nota-se primeiro que o projeto está configurado para ser compilado e executado usando o Java na versão 17 (maven.compiler.source e maven.compiler.target) nesse trecho do código:

```
<properties>
```

```
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>
```

E nesse outro trecho, temos o bloco <build> que contém configurações relacionadas à construção do projeto. Aqui, especificamente, há a definição de recursos, indicando que os arquivos dentro do diretório src/main/java/resources serão considerados recursos do projeto durante o processo de construção.

```
<build>
    <resources>
        <resource>
            <directory>
                src/main/java/resources
            </directory>
        </resource>
    </resources>
    <!-- Plugins também estão configurados dentro do bloco de build
-->
</build>
```

- **Análise parcial do código**

O projeto em análise é uma aplicação que visa visualizar as notações de complexidade de tempo (Big O) dos algoritmos de ordenação mais usados. Ele utiliza a interface gráfica Java Swing para criar uma representação visual das performances desses algoritmos.

A estrutura geral do projeto consiste em basicamente uma pasta resources com 2 imagens em png e outra pasta com as classes de cada algoritmo de ordenação, um arquivo principal, que de fato executa o projeto, além de um arquivo form que é usado de referência para o arquivo principal. Aqui analisaremos dois arquivos de exemplo: BubbleSort.java e InsertionSorte.java.

BubbleSort.java

```
package com.mycompany.sortingalgorithms;
/**
 *
 * @author Yusuf Ihsan Simsek
 */
public class BubbleSort {
```

```
    public static int[] bubbleSort(int array[], int length) {
        for (int i = 0; i < length - 1; i++) {
            for (int j = 0; j < length - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
        return array;
    }
}
```

A classe BubbleSort implementa o algoritmo de ordenação conhecido como Bubble Sort em Java. Este método de ordenação percorre iterativamente uma lista, comparando elementos adjacentes e trocando-os se estiverem fora de ordem. O processo é repetido até que a lista esteja completamente ordenada.

O algoritmo é implementado com dois loops aninhados. O loop externo (for (int i = 0; i < length - 1; i++)) representa as passagens pela lista. Cada passagem coloca o maior elemento não ordenado na posição correta no final da lista. O loop interno (for (int j = 0; j < length - i - 1; j++)) percorre a lista até o último elemento não ordenado. Dentro deste loop, os elementos adjacentes são comparados, e se estiverem fora de ordem, são trocados.

O processo continua até que toda a lista esteja ordenada. A complexidade de tempo do Bubble Sort é O(n^2) no caso médio e pior caso, tornando-o menos eficiente para grandes conjuntos de dados em comparação com algoritmos mais avançados, como Merge Sort ou Quick Sort.

Insertion Sort

```
package com.mycompany.sortingalgorithms;
/**
 *
 * @author Yusuf Ihsan Simsek
 */

public class InsertionSort {
```

```
    public static void insertionSort(int array[], int length) {
        for (int i = 1; i < length; ++i) {
            int key = array[i];
            int j = i - 1;
            while (j >= 0 && array[j] > key) {
                array[j + 1] = array[j];
                j = j - 1;
            }
            array[j + 1] = key;
        }
    }
}
```

A classe InsertionSort em Java implementa o algoritmo de ordenação conhecido como Insertion Sort. Este algoritmo percorre iterativamente uma lista, considerando um elemento por vez e inserindo-o na posição correta em relação aos elementos já ordenados.

O loop externo for (for (int i = 1; i < length; ++i)) percorre a lista a partir do segundo elemento até o último. Durante cada iteração desse loop, um elemento é marcado como a "chave" (key). Dentro do loop, há um segundo loop while que compara a "chave" com os elementos à sua esquerda. Se o elemento à esquerda for maior que a "chave", ele é deslocado para a direita. Esse processo continua até encontrar a posição correta para a "chave". Após encontrar a posição correta, a "chave" é inserida na posição apropriada na lista ordenada. Este processo é repetido até que todos os elementos da lista estejam ordenados.

A complexidade de tempo do Insertion Sort é O(n^2) no caso médio e pior caso, tornando-o adequado para listas pequenas ou quase ordenadas. No entanto, para conjuntos de dados maiores, outros algoritmos como Merge Sort ou Quick Sort são geralmente preferidos devido à sua eficiência superior.