Relatório da atividade de prática 02

Disciplina: Estrutura de Dados

Aluno: Vinícius Alves de Moura

Introdução

Existem vários algoritmos de ordenação disponíveis para organizar os mais diversos tipos de dados, cada um deles apresenta peculiaridades e é mais indicado para um tipo específico de aplicação. Cabe aos desenvolvedores conhecer, testar e aplicar o melhor algoritmo para aquela tarefa, reduzindo assim o esforço computacional necessário. Um dos aspectos mais importantes para se conhecer essas técnicas são as suas complexidades que acabam por se manifestar no tempo necessário para ordenar um certo conjunto de dados. Na presente atividade foi explorado o tempo de execução de dois algoritmos de ordenação de dados: *Binary Insertion Sort* e *Selection Sort.*

O algoritmo BinaryInsertion é uma mistura de *binary search* com *insertion sort*. Ele funciona comparando um elemento com os outros anteriores, mas diferente do insertion sort, não é necessário fazer comparações com todos esses elementos anteriores. Nessa etapa é usada a mesma lógica do binary sort, onde o conjunto de dados que serão comparados com o elemento que se deseja ordenar é dividido em dois, e o elemento é comparado apenas com aquele que ocupa a metade do conjunto de dados já ordenado. Dessa forma a complexidade é quadrática.

Já o algoritmo Selection sort consiste em ordenar por seleção, procurando o menor elemento do conjunto de dados e colocando ele na primeira posição, depois procurando o segundo menor elemento e o colocando na segunda posição, e assim sucessivamente até ordenar todos os dados. A complexidade desse algoritmo é quadrática e em todos os casos tem que realizar o mesmo número de comparações.

Objetivo

O objetivo principal da atividade desenvolvida foi medir o tempo de execução de dois algoritmos de ordenação em diferentes casos.

Metodologia

Para avaliar o tempo de execução dos algoritmos foi desenvolvido um código em java que importa uma série de listas de nomes desordenadas e mede o tempo necessário para realizar dez ordenações sucessivas usando cada um dos algoritmos escolhidos (*Binary Insertion Sort* e *Selection Sort*). As listas de nomes foram fornecidas pelo professor da disciplina e a implementação dos algoritmos foi obtida no site algs4.cs.princeton.edu.

Para a leitura dos arquivos, foi implementado o método *getNomes* que recebe o nome do arquivo e retorna um array do tipo String onde cada elemento corresponde a uma linha do arquivo.

```java
// Método que recebe o nome do arquivo a ser lido e retorna um array de String
// onde cada elemento corresponde a uma linha do arquivo
public static String[] getNomes(String nomeArquivo){
  ArrayList<String> listaNomes = new ArrayList<String>(); // Cria um arraylist para guardar os dados
  try {
    File arquivo = new File(nomeArquivo);
    Scanner nomes = new Scanner(arquivo);
    while (nomes.hasNextLine()) { // Continua lendo enquanto ainda estiver linhas para serem lidas
      String nome = nomes.nextLine();
      listaNomes.add(nome);  // Adiciona a linha lida ao ArrayList
    }
    nomes.close();
  } catch (FileNotFoundException e) {
    System.out.println("Erro ao efetuar a leitura");
    e.printStackTrace();
  }
  return listaNomes.toArray(new String[0]); // Retorna o arraylist convertido para array
}
```

Dentro da função main foi feita a leitura das três listas, atribuindo suas linhas à arrays

```java
Run | Debug
public static void main(String[] args) {
  // Efetua a leitura das três listas atribuindo cada uma a seu respectivo array
  String[] nomes5k = getNomes("nomes5k.txt");
  String[] nomes10k = getNomes("nomes10k.txt");
  String[] nomes50k = getNomes("nomes50k.txt");
  double comeco; // Declarava a variável necessária para marcar o tempo
```

Foi implementado múltiplas chamadas dos métodos de ordenação. Para cada tamanho de lista, cada método é executado dez vezes e no começo de cada execução, o array a ser ordenado recebe os dados desordenados originais.

```
//////////////////////////Listas desordenadas
// 5k
System.out.print("5k\t");
String[] nomes5kTemp = new String[nomes5k.length]; // Declara uma variável que recebe os dados da lista e é enviada para ordenação

// BinaryInsertion
comeco = System.nanoTime();
for(int i = 0; i < 10; i++){ //Faz 10 repetições
  System.arraycopy(nomes5k, 0, nomes5kTemp, 0, nomes5k.length); // A cada repetição o array recebe os dados desordenados
  BinaryInsertion.sort(nomes5kTemp); // Executa a ordenação
}
System.out.print(((System.nanoTime() - comeco)/1000000.0) + " ms\t"); // Imprime o tempo de execução

// Selection
comeco = System.nanoTime();
for(int i = 0; i < 10; i++){
  System.arraycopy(nomes5k, 0, nomes5kTemp, 0, nomes5k.length);
  Selection.sort(nomes5kTemp);
}
System.out.println(((System.nanoTime() - comeco)/1000000.0) + " ms");
```

Posteriormente o mesmo teste é realizado, mas dessa vez o array é previamente ordenado.

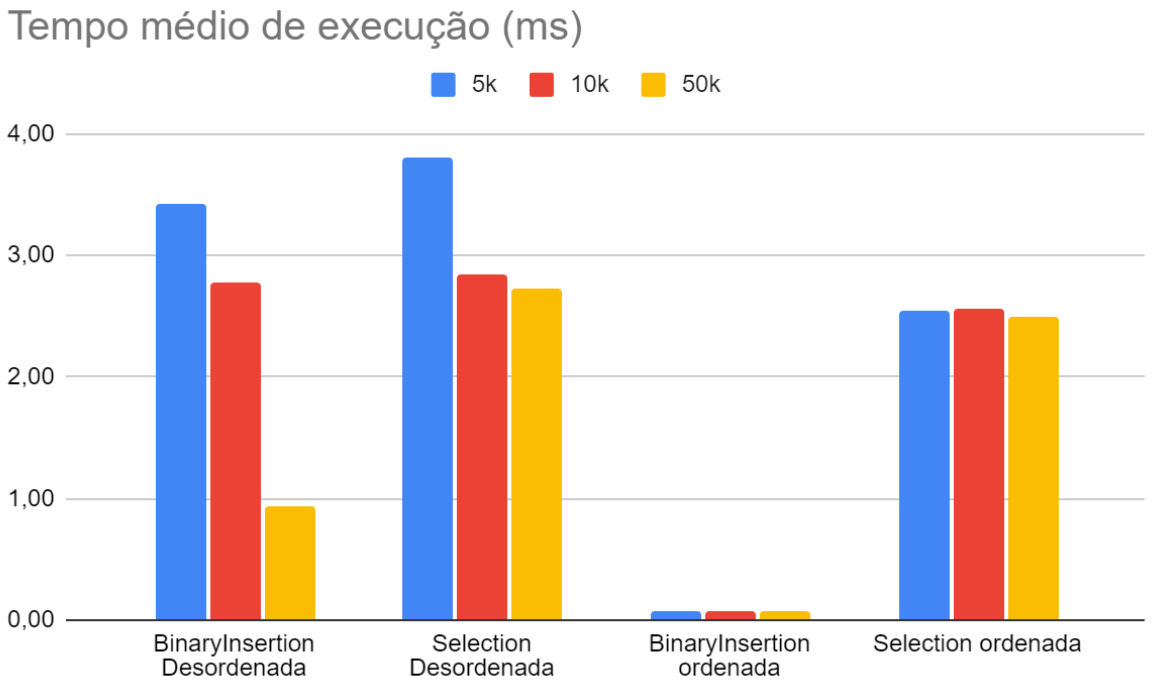
```
System.out.println("\tBinaryInsertion\tSelection");
// 5k
System.out.print("5k\t");

// BinaryInsertion
System.arraycopy(nomes5k, 0, nomes5kTemp, 0, nomes5k.length);
BinaryInsertion.sort(nomes5kTemp);
comeco = System.nanoTime();
for(int i = 0; i < 10; i++){
  BinaryInsertion.sort(nomes5kTemp);
}
System.out.print(((System.nanoTime() - comeco)/1000000.0) + " ms\t");

// Selection
System.arraycopy(nomes5k, 0, nomes5kTemp, 0, nomes5k.length);
Selection.sort(nomes5kTemp);
comeco = System.nanoTime();
for(int i = 0; i < 10; i++){
  Selection.sort(nomes5kTemp);
}
System.out.println(((System.nanoTime() - comeco)/1000000.0) + " ms");
```

Resultados

O algoritmo obteve os seguintes resultados:

	BinaryInsertion Desordenada	Selection Desordenada	BinaryInsertion ordenada	Selection ordenada
5k	3,43	3,80	0,07	2,55
10k	2,79	2,84	0,07	2,57
50k	0,93	2,73	0,08	2,50

*tempos em ms



Analisando os dados é possível observar que números de dados maiores são mais eficientes usando o BinarayInsertion que o Seletion Sort, e ainda que quando a lista já está ordenada, o tempo de execução do BinaryInsertion é muito baixo, já para o Selection Sort continua o mesmo.
Isso acontece pois o algoritmo Seletion faz o mesmo número de comparações em todos os tipos de caso, e por isso seu tempo de execução tende a não ser alterado quando a lista já está ordenada.