CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CURSO DE INFORMÁTICA
TURMA: INFORMÁTICA - 3

Artur Borges Corrêa
Caio César Nascimento Silva
Igor Richard Dias Silva

Trabalho Prático 1

Métodos de Ordenação (InsertionSort; ShellSort; MergeSort)

Contagem
Maio de 2022

Artur Borges Corrêa
Caio César Nascimento Silva
Igor Richard Dias Silva

Trabalho Prático 1

Relatório abordando os métodos de ordenação InsertionSort, ShellSort e MergeSort, aplicado no curso de Linguagem e Técnicas de Programação II.

Professor(a): Alisson Rodrigo dos Santos
Disciplina: Linguagem e Técnicas de Programação II
Turma: INF-3

Contagem
Maio de 2022

Resumo

O relatório tem por objetivo realizar uma análise a respeito dos métodos de ordenação InsertionSort, ShellSort e MergeSort, com foco no do tempo de execução de cada um em diversas situações. A realização do trabalho foi feita usando o VS Code para a construção do código, juntamente com pesquisas feitas para entender sobre os métodos de ordenação utilizados. Após o estudo, pode-se descobrir a grande eficácia de se utilizar métodos diferentes para situações específicas. O MergeSort é melhor aplicado para grandes quantidades de dados não ordenados, o ShellSort para quantidade menores de dados, e o método mais simples, InsertionSort, somente se sobressai em situações de verificação de material já ordenado. Após os testes, os alunos obtiveram novos conhecimentos sobre o melhor e mais eficiente método para situações diversas onde há a necessidade de uma ordenação.

Palavras-chave: Métodos de Ordenação; Java; ShellSort, MergeSort, InsertionSort.

Sumário

Lista de ilustrações

1 Introdução

Ordenação é o ato de se colocar os elementos de uma sequência de informações, ou dados, em uma ordem predefinida. O termo técnico em inglês para ordenação é “sorting”, cuja tradução literal é “classificação”.

Um método de ordenação é uma maneira pré-determinada de realizar essa ordenação, onde existem diversos meios, tendo cada um seus tipos de peculiaridades e situações onde são melhores ou piores. Os diversos tipos de ordenação podem ser utilizados em basicamente qualquer situação.

Há maneiras de se comparar matematicamente os métodos, através do cálculo de sua complexidade e capacidade, porém tais técnicas e resultados não serão abordados durante este trabalho.

Durante este trabalho, serão abordados os métodos InsertionSort, MergeSort e ShellSort, com o objetivo de avaliar os tempos de execução de cada um dos métodos e comparar em quais situações são melhores para se utilizar um método de ordenação.

2 Funções

2.1 Métodos de Ordenação

2.1.1 Teórico InsertionSort

O método de ordenação Insertion Sort é feito começando pelo segundo valor, que vai comparando da esquerda para direita, deixando-os ordenados à medida que vai passando pelos valores. Ele é melhor utilizado para ordenar um pequeno número de valores.

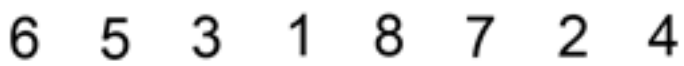

Figura 1 – Insertion Sort (GIF)
https://lamfo-unb.github.io/img/Sorting-algorithms/Insertion-sort-example-300px.gif

2.1.2 Teórico ShellSort

O ShellSort melhora a ordenação por inserção, dividindo a lista original em pequenas sub listas e ordenando-as. O ponto principal desse método é como essas sub listas são escolhidas. Em vez de quebrar a lista em sub listas de itens contíguos, o ShellSort usa um incremento, às vezes chamado de gap, para criar uma sub-lista escolhendo todos os itens que estão afastados i itens uns dos outros. Isso pode ser visto na Figura 1. Essa lista tem nove itens. Se usarmos um incremento de três, serão três sublistas, cada uma sendo submetida à ordenação por inserção. Depois de realizado esse processo, ficamos com a lista mostrada em Figura 2. Embora essa lista não esteja completamente ordenada, algo muito interessante aconteceu. Ao ordenarmos as sublistas, os itens ficaram mais próximos de onde eles pertencem de fato.

Figura 2 – ShellSort com Incremento de Três

https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OShellSort.html

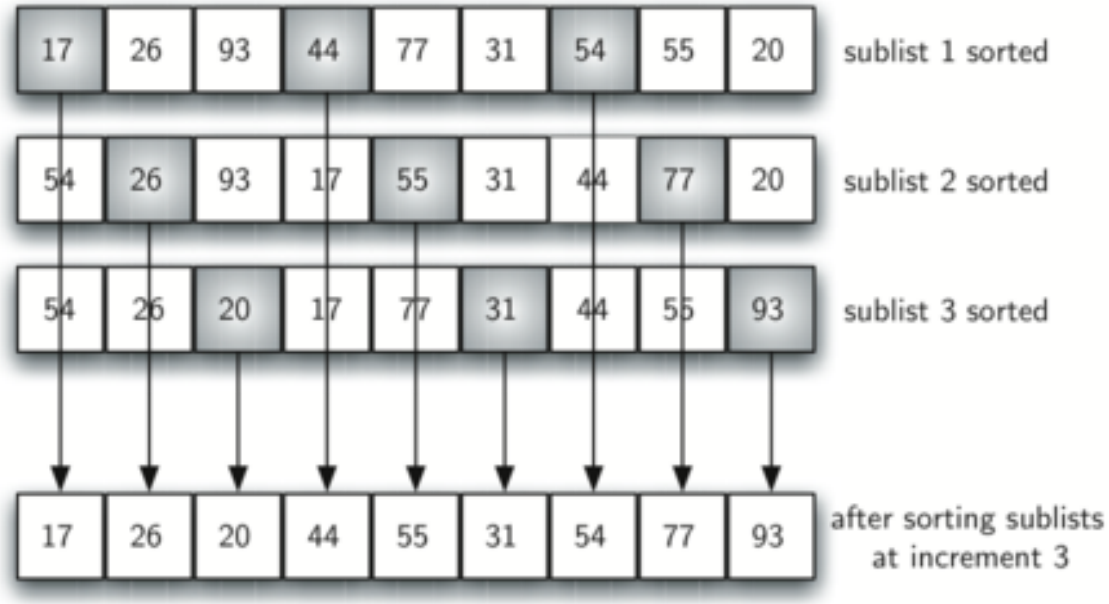

Figura 3 – ShellSort depois de Ordenar Cada Sublista

https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OShellSort.html

A Figura 3 mostra uma ordenação por inserção final usando um incremento de um; em outras palavras, uma ordenação por inserção convencional. Observe que ao realizar as ordenações de sub listas anteriores, reduzimos agora o número total de operações de deslocamento necessárias para colocar a lista na sua ordem final. Nesse caso, precisamos de apenas mais quatro deslocamentos para completar o processo.

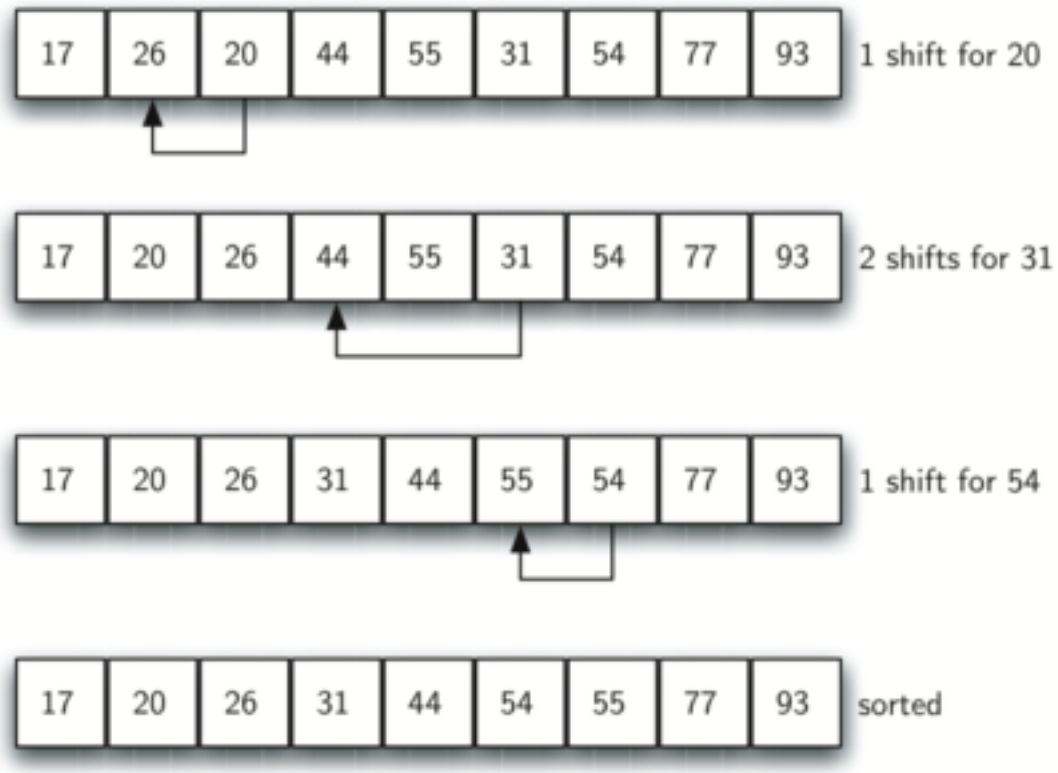

Figura 4 – ShellSort: Uma Inserção Final Com Incremento de 1

https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OShellSort.html

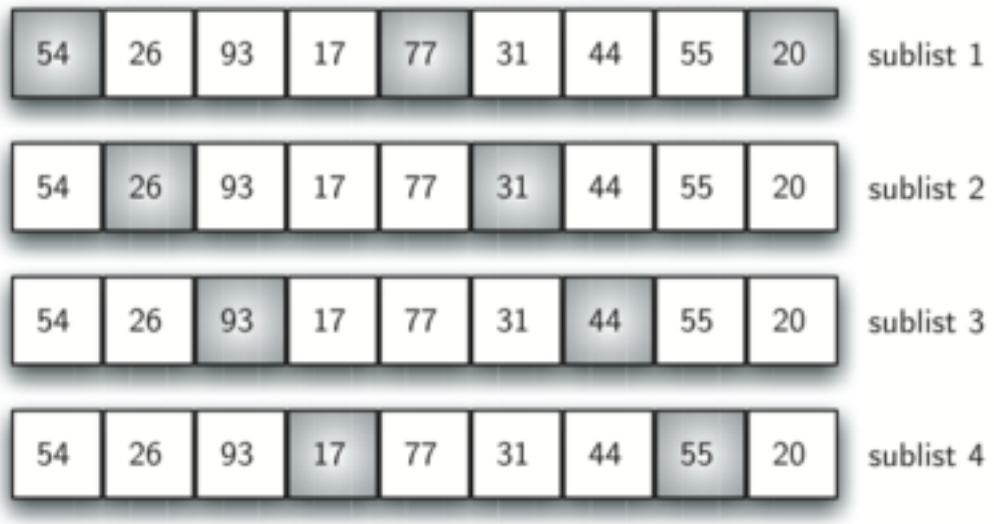

Figura 5 – Sub Listas Iniciais para um ShellSort

https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/OShellSort.html

À primeira vista, você pode pensar que o ShellSort não tem como ser melhor que o InsertionSort, já que ele a utiliza na lista toda como um último passo. Acontece que essa

ordenação por inserção final não precisa realizar muitas comparações (ou deslocamentos), já que a lista foi preordenada por ordenação por inserção incrementais anteriores. Em outras palavras, cada passagem produz uma lista que está “mais ordenada” que a anterior. Isso faz com que a passagem final seja muito eficiente.

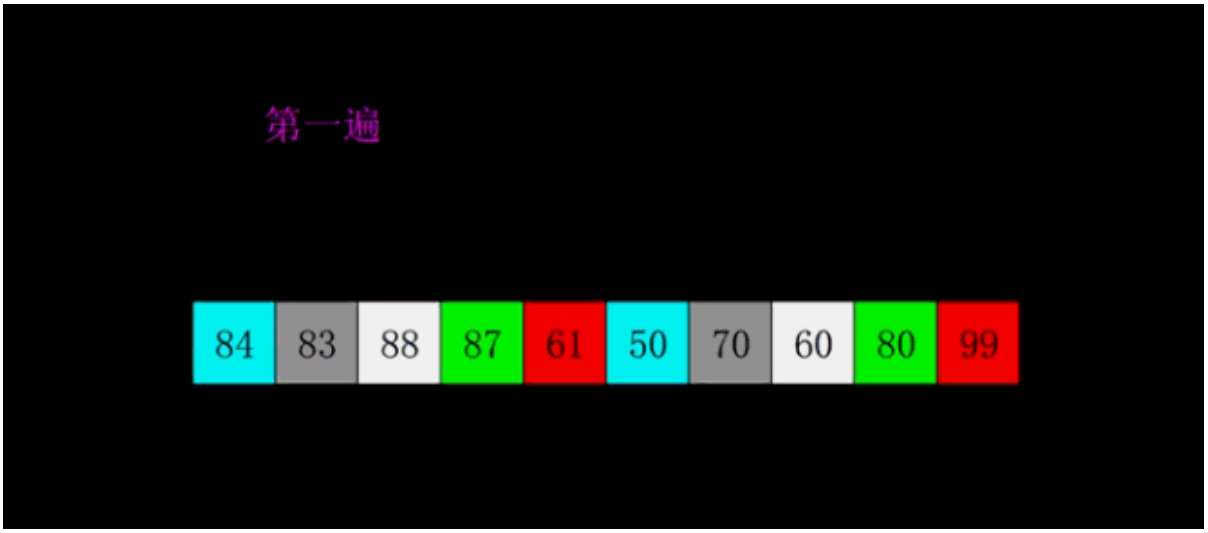

Figura 6 – Como funciona o método Shell Sort
https://blog.csdn.net/ytx2014214081/article/details/105841184

2.1.3 Teórico MergeSort

O método MergeSort se baseia em um princípio conhecido como “Dividir para Conquistar”, e pode ser observado na seguinte Figura 4:

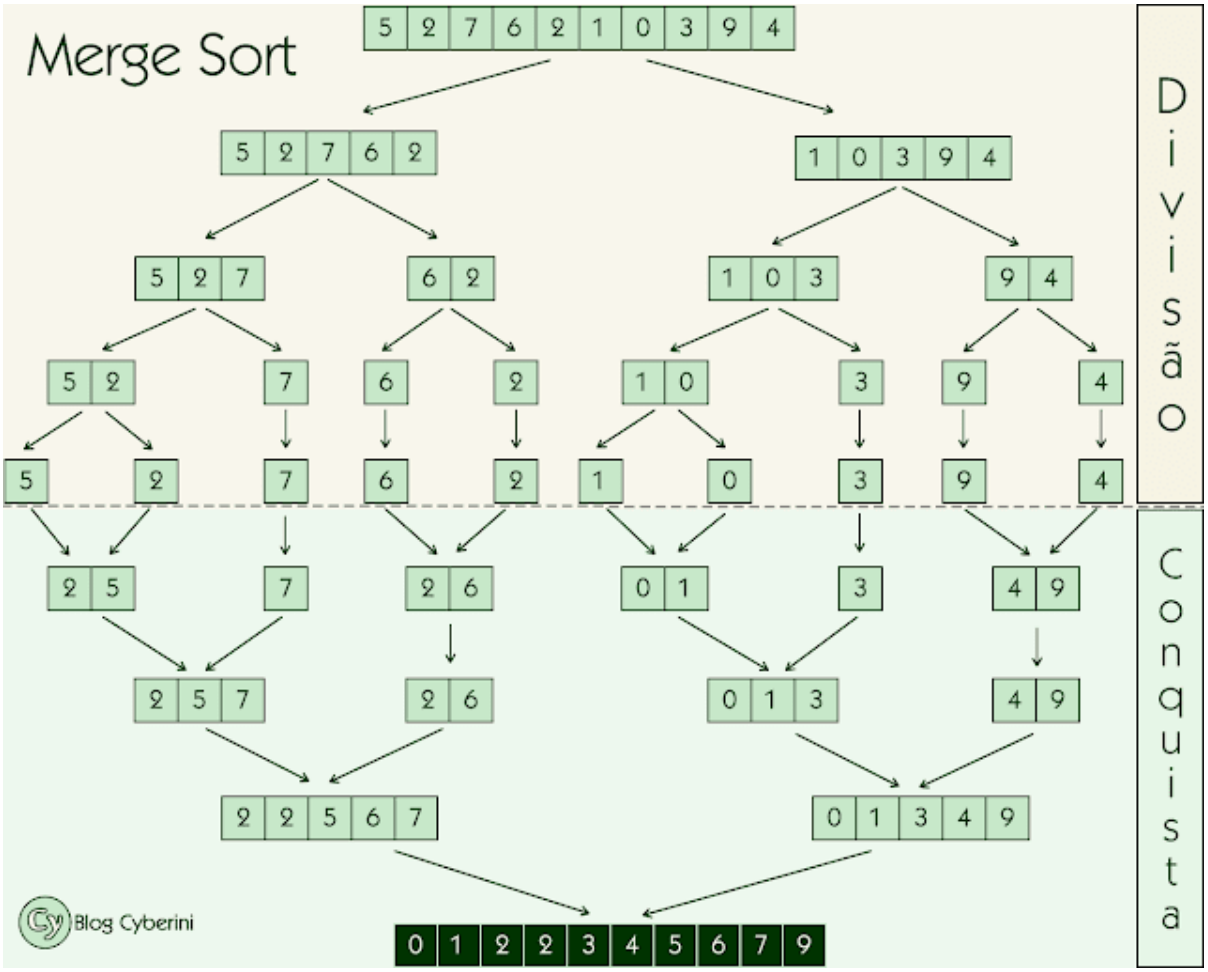

Figura 7 – Divisão X Conquista

https://www.blogcyberini.com/2018/07/merge-sort.html

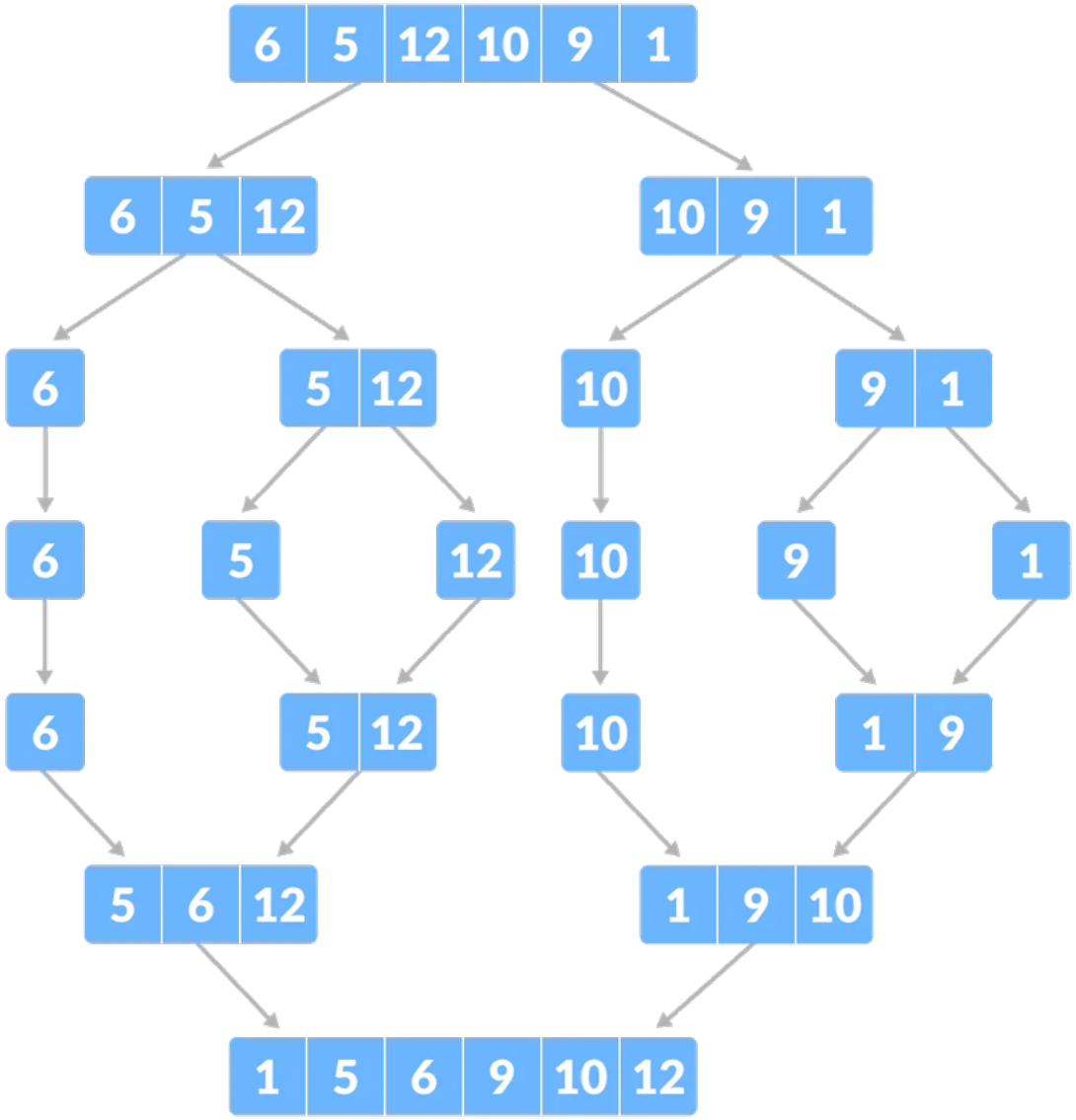

Figura 8 - Método de Funcionamento do MergeSort

https://www.programiz.com/dsa/merge-sort

Ele se baseia em uma chamada recursiva contínua, que trabalha repartindo o vetor (que é considerado um problema) em vetores cada vez menores (ou seja, problemas menores) até chegar em uma separação unitária. Após isso, sua funcionalidade se baseia em unir vetores que antes eram um só, porém realizando uma ordenação naquela parte em cada união (ou seja, resolvendo os problemas a cada junção).

Figura 9 – Animação de MergeSort (GIF)
https://upload.wikimedia.org/wikipedia/commons/c/c5/Merge_sort_animation2.gif

Sua principal desvantagem se trata do gasto extra de memória, visto que uma nova cópia de parte do vetor é gerada a cada repartição, durante a ordenação.

2.1 Outras Funções

2.1.1 Leitura

Na parte responsável pela leitura, a classe OpenTXTFile, é feita a leitura do arquivo usando o try catch juntamente com o scanner para ler o arquivo apresentado enquanto é pulada a primeira linha do mesmo. Enquanto o arquivo ainda tiver linhas os dados dele serão pegos e guardados num vetorDeStrings criado para isso. Depois disso o scanner é fechado, assim como é apresentado um erro se der errado alguma parte da leitura. A classe é finalizada retornando o vetorDeStrings.

2.1.2 Exportação

A classe ExportToCSVFile possui a função ExportarCSV(), do qual é responsável por exportar o vetor de string recebido no parâmetro em um arquivo csv com o nome recebido no parâmetro, através da biblioteca Print Writter. O arquivo recebe o vetor já ordenado e o tempo de execução de cada método de ordenação.

2.2.3 Main

No main, inicialmente é criado um objeto para cada uma das seguintes classes: OpenTXTFile; MergeSort; InsertionSort; ShellSort; ExportToCSVFile. Cada um dos objetos serve para manipular as funções presentes em suas devidas classes. Também são declaradas as variáveis para o cálculo do tempo total, além de um vetor, onde serão armazenados os dados, tanto antes quanto depois da ordenação.

Também existe um corpo padrão que é repetido durante o resto do main, para realizar as operações necessárias em cada arquivo:

```
vetorDeStrings = openTXTFile.LerTXT("entrada-100-cre.txt");
```

Recebe os valores iniciais do arquivo e armazena no vetor "vetorDeStrings" utilizando o objeto openTXTFile e o método LerTXT, passando por parâmetro o nome do arquivo a ser lido, que é modificado a cada chamada, para se adequar a necessidade.

```
startTime = System.currentTimeMillis();
```

Aqui, é obtido o valor em milissegundos do sistema e armazenado na variável startTime, para uso posterior.

```
vetorDeStrings = mergesort.OrdenarViaMergeSort(vetorDeStrings, vetorDeStrings.size());
```

o "vetorDeStrings" é chamado para ser ordenado, após o sinal de igual, se utiliza o objeto e método referente ao tipo de ordenação desejada, no caso do exemplo, o MergeSort, e é passado como parâmetros o próprio vetor.

```
endTime = System.currentTimeMillis();
duration = endTime - startTime;
```

O valor final de tempo é obtido (após a ordenação), e utilizando a variável 'duration' se calcula o tempo total gasto, que é a diferença entre os tempos obtidos.

```
exportar.ExportarCSV(vetorDeStrings,duration,"entrada-100-cre-ordenada-por-mergesort.csv");
```

E por fim, o arquivo é exportado, utilizando o objeto exportar e o método ExportarCSV, e por parâmetros são passados o vetor que foi ordenado, o tempo de duração, e o nome desejado para o arquivo de saída, onde mantemos o padrão de: "Nome anterior do arquivo + 'ordenada por' + método de ordenação".

3 Tabela Comparativa de Tempo

Tempo de Execução		Tipo de Dado Testado								
		100 Itens			1000 Itens			10000 Itens		
		Aleatório	Crescente	Decrescente	Aleatório	Crescente	Decrescente	Aleatório	Crescente	Decrescente
Método	InsertionSort	1 ms	0 ms	1 ms	9 ms	0 ms	16 ms	376 ms	1 ms	677 ms
	MergeSort	1 ms	1 ms	1 ms	2 ms	1 ms	0 ms	10 ms	8 ms	3 ms
	ShellSort	0 ms	0 ms	0 ms	2 ms	0 ms	1 ms	11 ms	4 ms	3 ms

Material derivado de testes próprios.

Dado X Método Por Tempo De Execução		Tipo de Dado Testado								
		100 Dados			1000 Dados			10000 Dados		
		Aleatório	Crescente	Decrescente	Aleatório	Crescente	Decrescente	Aleatório	Crescente	Decrescente
Método	InsertionSort	1 ms	0 ms	1 ms	9 ms	0 ms	16 ms	376 ms	1 ms	677 ms
	MergeSort	1 ms	1 ms	1 ms	2 ms	1 ms	0 ms	10 ms	8 ms	3 ms
	ShellSort	0 ms	0 ms	0 ms	2 ms	0 ms	1 ms	11 ms	4 ms	3 ms

Figura 10 – Tabela de comparação de Dados X Método de Ordenação (Em forma de imagem)

Tabela em forma de imagem (Via Excel)

*: Os resultados podem variar de acordo com a máquina utilizada.

**: Os testes foram realizados em uma máquina com a seguinte configuração:

CPU: Ryzen 5 5600X 4.6GHz

GPU: Radeon™ RX 480 G1 Gaming 8GB

RAM: 24 GB XPG DDR4 3200MHz

SSD: 1TB NVME XPG

4 Considerações Finais

Esse trabalho teve por objetivo realizar pesquisas e experiências dentro do campo dos métodos de ordenação, focando na análise do tempo de execução.

Para se obter um resultado eficiente do tempo de execução de cada método, foram realizados testes dentro de 3 tipos de arquivos, que se divergiam pela quantidade de dados (sendo estes 100, 1000 e 10000 dados por arquivo), e onde em cada grupo haviam-se três situações para testes, um arquivo já ordenado, um arquivo ordenado em ordem decrescente, e um último arquivo que estava em forma aleatória, completamente fora de ordem.

Após as análises, pode-se perceber algumas situações:

- O método InsertionSort somente se sobressai em situações extremamente específicas e pré-determinadas, no caso, em situações onde o conjunto de dados já está ordenado, e é necessário apenas realizar a verificação, ou inserir um elemento na lista na sua devida posição, mantendo a ordem.
- Os métodos ShellSort e MergeSort tiveram resultados extremamente promissores se comparados ao InsertionSort, reduzindo em algumas situações o tempo de execução em quase 226 vezes.
- No resultado em geral, ShellSort e MergeSort têm basicamente a mesma capacidade e qualidade para ordenação, divergindo em pouquíssimos milissegundos o tempo de execução entre si.

Com isso, pode-se perceber ao final que há métodos mais eficientes a depender da situação aplicada, e que o tipo de método utilizado em um programa deve ser escolhido com base em um estudo do caso específico para o qual o programa é proposto.

E por fim, para análises futuras, seria interessante a realização de testes que não levem em conta somente o tempo de execução, mas também a otimização por núcleo de processamento, a capacidade de trabalho com bases de dados realmente grandes, com milhões de informações, o gasto em memória, e diversas outras capacidades.

Referências

BAELDUNG. **How to Write to a CSV File in Java**. 21 de Dezembro de 2021. Disponível em: https://www.baeldung.com/java-csv. Acesso em: [Abril-Maio 2022].

BAELDUNG. **Merge Sort in Java**. 12 de Maio de 2022. Disponível em: https://www.baeldung.com/java-merge-sort. Acesso em: [Abril-Maio 2022].

DEVSUPERIOR. **Como ler arquivo texto CSV em Java (aplicação real) - Aulão #002**. 16 de Junho de 2020. Disponível em: https://www.youtube.com/watch?v=xLDViuYlqGM. Acesso em: [Abril-Maio 2022].

DEVSUPERIOR. **Como ler arquivo texto em Java (aplicação real) - Aulão #002**. 25 de Junho de 2020. Disponível em: https://github.com/devsuperior/aulao002. Acesso em: [Abril-Maio 2022].

JAVATPOINT. **Java Vector**. Disponível em: https://www.javatpoint.com/java-vector#:~:text=Vector%20is%20like%20the%20dynamic,is%20found%20in%20the%20java. Acesso em: [Abril-Maio 2022].

PROGRAMINHAS. **Tutorial Java - Ler e importar arquivos CSV (Comma-separated values)**. 24 de Outubro de 2014. Youtube. Disponível em: https://www.youtube.com/watch?v=HnO13VKQJKo. Acesso em: [Abril-Maio 2022].

RBOYD. **algorithms - Shellsort**. 20 de Março de 2013. Disponível em: https://github.com/rboyd/algorithms/blob/master/src/java/Shellsort.java. Acesso em: [Abril-Maio 2022].

SAKURAI, R. G. **Ordenação de Dados - Merge Sort**. 18 de Maio de 2020. Disponível em: http://www.universidadejava.com.br/pesquisa_ordenacao/merge-sort/. Acesso em: [Abril-Maio 2022].

SIMMONS, J. **Insertion Sort - String ArrayList**. 16 de Fevereiro de 2021. Disponível em: https://www.youtube.com/watch?v=4P95xpLjbQ4. Acesso em: [Abril-Maio 2022].

SINGH, C. **Java Program to Sort Strings in an Alphabetical Order**. 4 de Janeiro de 2019. Disponível em: https://beginnersbook.com/2018/10/java-program-to-sort-strings-in-an-alphabetical-order/. Acesso em: [Abril-Maio 2022].

W3SCHOOLS. **Java String compareTo() Method**. Disponível em: https://www.w3schools.com/java/ref_string_compareto.asp. Acesso em: [Abril-Maio 2022].

Apêndices

APÊNDICE A – InsertionSort

Enquanto era pesquisado sobre InsertionSort foram reutilizados a lógica e o código do vídeo:

https://www.youtube.com/watch?v=4P95xpLjbQ4

APÊNDICE B – ShellSort

Na pesquisa para implementar o ShellSort, foi reutilizado um código em um repositório do Github disponível em:

https://github.com/rboyd/algorithms/blob/master/src/java/Shellsort.java

APÊNDICE C – MergeSort

O código base foi implementado com a ideia apresentada no site:

https://www.baeldung.com/java-merge-sort

```java
public static void mergeSort(int[] a, int n) {
    if (n < 2) {
        return;
    }
    int mid = n / 2;
    int[] l = new int[mid];
    int[] r = new int[n - mid];

    for (int i = 0; i < mid; i++) {
        l[i] = a[i];
    }
    for (int i = mid; i < n; i++) {
        r[i - mid] = a[i];
    }
    mergeSort(l, mid);
    mergeSort(r, n - mid);

    merge(a, l, r, mid, n - mid);
}
```

Figura 11 – Função de chamada recursiva do MergeSort

https://www.baeldung.com/java-merge-sort

```java
public static void merge(
  int[] a, int[] l, int[] r, int left, int right) {

    int i = 0, j = 0, k = 0;
    while (i < left && j < right) {
        if (l[i] <= r[j]) {
            a[k++] = l[i++];
        }
        else {
            a[k++] = r[j++];
        }
    }
    while (i < left) {
        a[k++] = l[i++];
    }
    while (j < right) {
        a[k++] = r[j++];
    }
}
```

Figura 12 – Função principal de ordenação do MergeSort

https://www.baeldung.com/java-merge-sort

```java
@Test
public void positiveTest() {
    int[] actual = { 5, 1, 6, 2, 3, 4 };
    int[] expected = { 1, 2, 3, 4, 5, 6 };
    MergeSort.mergeSort(actual, actual.length);
    assertArrayEquals(expected, actual);
}
```

Figura 13 – Testes de MergeSort

https://www.baeldung.com/java-merge-sort

Foram também utilizadas algumas outras funções conforme a necessidade, como a implementação para métodos com a utilização do Vector, a transferência de testes entre inteiros para String, ou a importação e exportação de arquivos, conforme referências.

Anexos

ANEXO A – Código InsertionSort

```
import java.util.*;

public class InsertionSort {

	public Vector<String> ordenar(Vector<String> vetor) {

		int i, j;

		String x;


		for (i = 1; i < vetor.size(); i++) {

			x = vetor.get(i);

			j = i;

			while (j > 0 && x.compareTo(vetor.get(j - 1)) < 0) {

				vector.set(j, vetor.get(j - 1));

				j--;

			}

			vector.set(j, x);

		}

	return vetor;

	}

}
```

ANEXO B – Código ShellSort

```
import java.util.*;

public class ShellSort {

      Vector<String> listaNomes = new Vector<String>();

            public Vector<String> ordenar(Vector<String> vectorNames) {

                  int i, j, h = 1;

                  String temp;

                  while (h <= vectorNames.size() / 3) {

                        h = h * 3 + 1;

                  }

                  while (h > 0) {


                        for (i = h; i < vectorNames.size(); i++) {

                              temp = vectorNames.get(i);

                              for ( j = i; j >= h && vectorNames.get(j -
                              h).compareTo(temp) > 0; j -= h) {

                                    vectorNames.set(j,  vectorNames.get(j  -
                                    h));

                              }

                        vectorNames.set(j, temp);

                        }

                        h = (h - 1) / 3;

                  }

              return vectorNames;

}
```

ANEXO C – Código MergeSort

```
import java.util.*;

public class MergeSort {

    public Vector<String> OrdenarViaMergeSort(Vector<String>
vetorDeStrings, int tamanhoDoVetor) {

        if (tamanhoDoVetor < 2) {

          return vetorDeStrings;

        }

        int meio = tamanhoDoVetor / 2

        Vector<String> vetorEsquerda = new Vector<>();

        Vector<String> vetorDireita = new Vector<>();

        for (int i = 0; i < meio; i++) {

          vetorEsquerda.add(vetorDeStrings.get(i));

        }

        for (int i = meio; i < tamanhoDoVetor; i++) {

          vetorDireita.add(vetorDeStrings.get(i));

        }

        vetorEsquerda = OrdenarViaMergeSort(vetorEsquerda,
vetorEsquerda.size());

        vetorDireita = OrdenarViaMergeSort(vetorDireita,
vetorDireita.size());

        vetorDeStrings = Merge(vetorEsquerda, vetorDireita,
vetorDeStrings);

        return vetorDeStrings;

    }

    public Vector<String> Merge(Vector<String> vetorEsquerda,
Vector<String> vetorDireita, Vector<String> vetorDeStrings) {

        int i = 0;
```

```
        int j = 0;
        int k = 0;
        while (i < vetorEsquerda.size() && j < vetorDireita.size()) {
            if (vetorEsquerda.get(i).compareTo(vetorDireita.get(j)) < 0) {
              vetorDeStrings.set(k, vetorEsquerda.get(i));
              i++;
            } else {
              vetorDeStrings.set(k, vetorDireita.get(j));
              j++;
            }
            k++;
        }
        while (i < vetorEsquerda.size()) {
            vetorDeStrings.set(k, vetorEsquerda.get(i));
            i++;
            k++;
        }
        while (j < vetorDireita.size()) {
            vetorDeStrings.set(k, vetorDireita.get(j));
            j++;
            k++;
        }
        return vetorDeStrings;
    }
}
```

ANEXO D – Código Leitura

```
import java.io.*;
import java.util.*;
public class OpenTXTFile {
    public Vector<String> LerTXT(String file) {
        Scanner scanner;
        File txt = new File(file);
        Vector<String> vetorDeStrings = new Vector<>();
        try {
            scanner = new Scanner(txt);
            scanner.nextLine();
            while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            vetorDeStrings.add(line);
            }
            scanner.close();
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }
        return vetorDeStrings;
    }
}
```

ANEXO E – Código Exportação

```
import java.io.*;

import java.util.*;

public class ExportToCSVFile {

    public void ExportarCSV(Vector<String> vetorDeStrings, long duration,
String filename ) {

        File csv = new File(filename);

        try {

            PrintWriter writer = new PrintWriter(csv);

            writer.println("Nomes ordenados");

            for (int i = 0; i < vetorDeStrings.size(); i++) {

                writer.println(vetorDeStrings.get(i));

            }

            writer.println();

            writer.println("Tempo de execução: " + duration + "
milissegundos");

            writer.close();

        } catch (Exception e) {

            System.out.println("Erro: " + e.getMessage());

        }

    }

}
```