**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ**

**ESCOLA POLITÉCNICA**

**RESOLUÇÃO DE PROBLEMAS ESTRUTURADOS NA COMPUTAÇÃO**

**PROFESSOR: ANDREY CABRAL MEIRA**

**ALUNO: RODRIGO AUGUSTO FIGUEIRA MOREIRA**

# ATIVIDADE TDE3 / RA4

# RELATÓRIO DO PROJETO DE ORDENAÇÃO

**CURITIBA**

**2024**

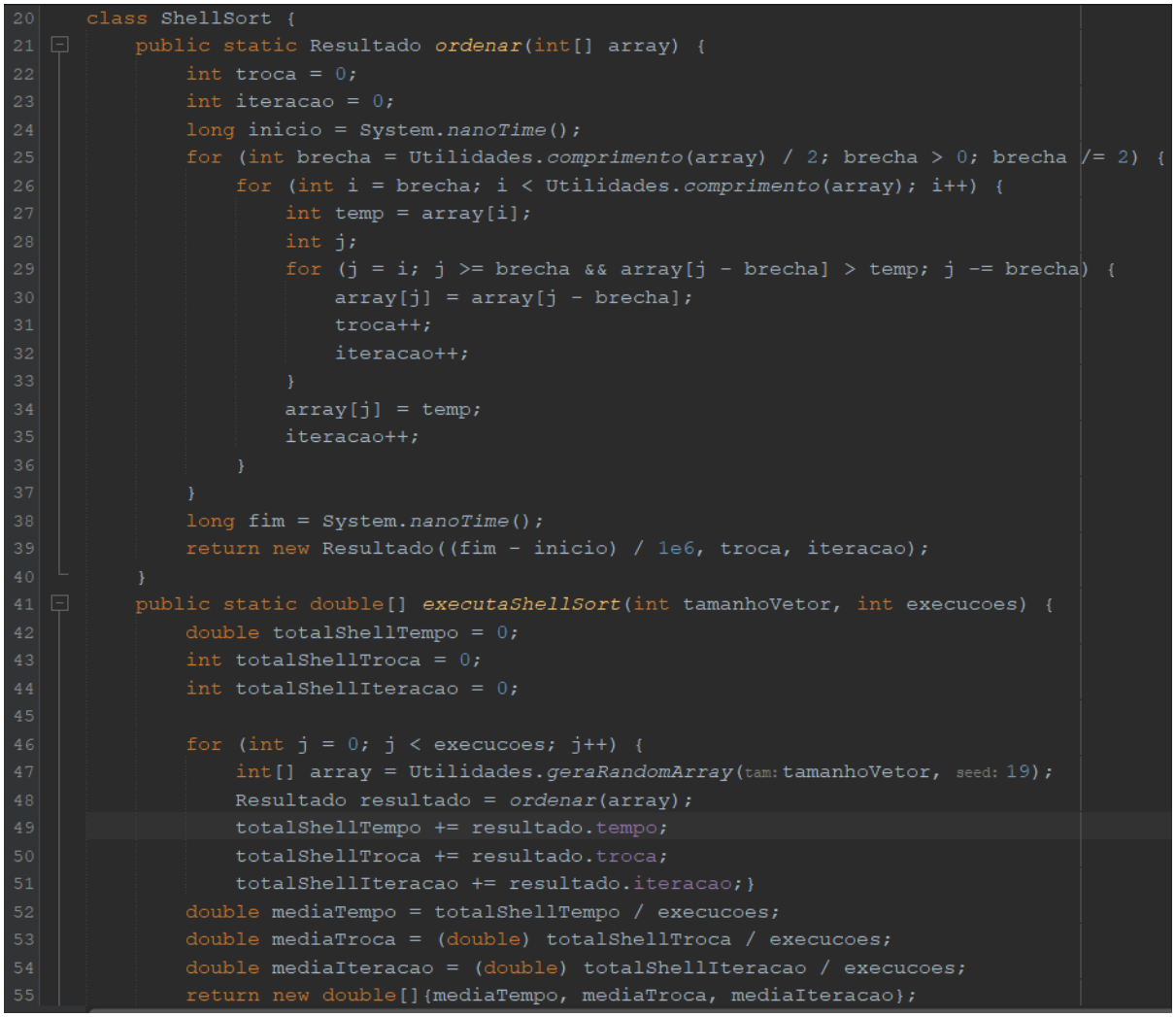
## INTRODUÇÃO

O algoritmo escolhido do grupo B foi Shell Sort, minha escolha deve-se a ser uma versão mais otimizada de um Insertion Sort regular, o que achei interresante e o do grupo C foi o Gnome Sort, o qual também é uma melhoria do Insert sort porém de uma maneira mas perculiar. Para gerar os gráficos foi utilizada a extensão XChart, e a seed sendo a 19.

## EXPLICAÇÃO DO FUNCIONAMENTO DO CÓDIGO E JUSTIFICATIVA

```
7
8   class Resultado {
9       double tempo;
10      int troca;
11      int iteracao;
12
13      public Resultado(double tempo, int troca, int iteracao) {
14          this.tempo = tempo;
15          this.troca = troca;
16          this.iteracao = iteracao;
17      }
18  }
19
```

**Classe Resultado**: Tem o propósito de encapsular e organizar os dados sobre o desempenho dos algoritmos de ordenação (shell sort e gnome sort) ao longo das execuções no código. Sua função principal é armazenar as métricas: tempo, trocas e iterações de cada execução. Justificativa: É um componente vital para atributos requisitados no exercício.



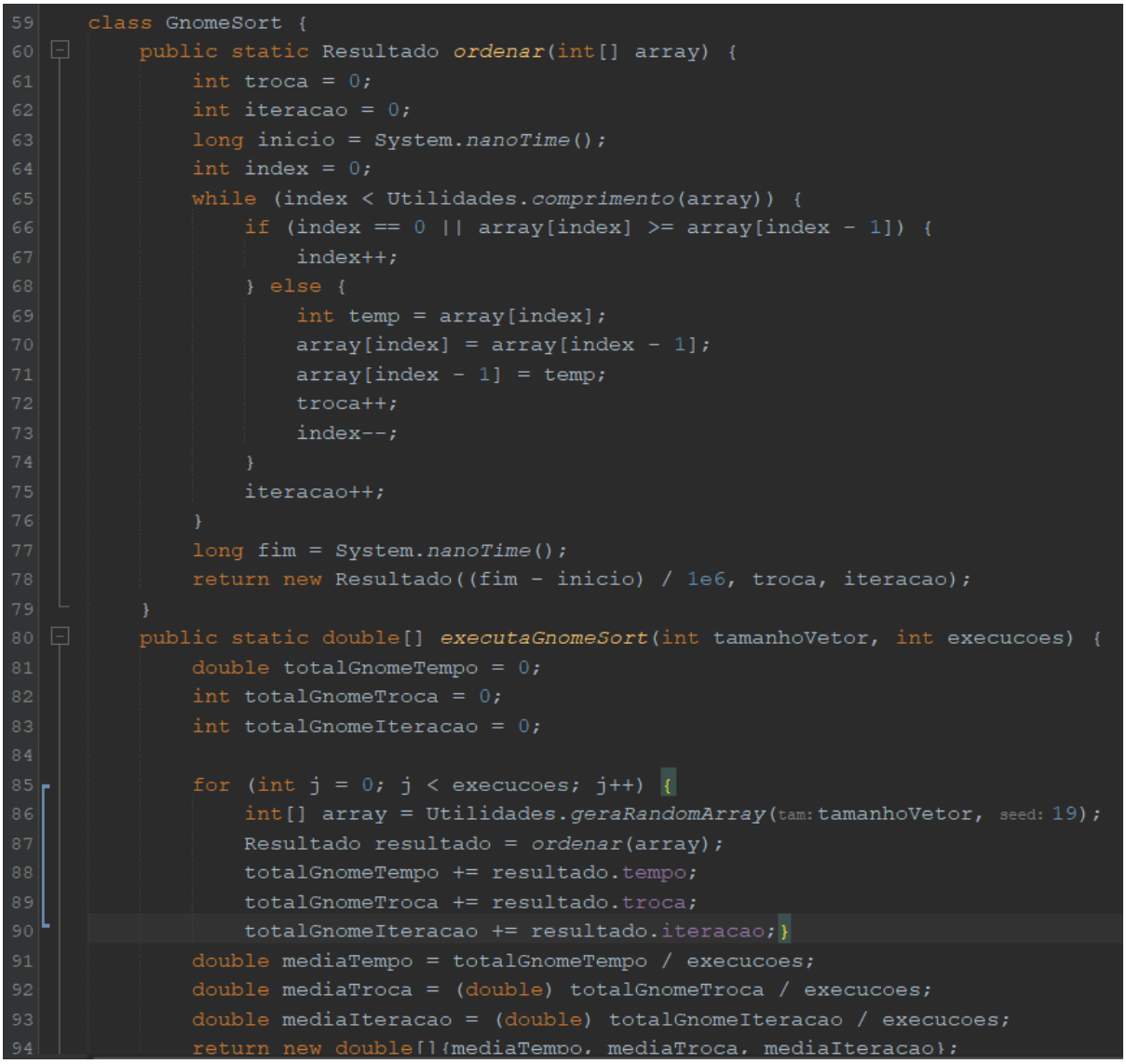
```java
class ShellSort {
    public static Resultado ordenar(int[] array) {
        int troca = 0;
        int iteracao = 0;
        long inicio = System.nanoTime();
        for (int brecha = Utilidades.comprimento(array) / 2; brecha > 0; brecha /= 2) {
            for (int i = brecha; i < Utilidades.comprimento(array); i++) {
                int temp = array[i];
                int j;
                for (j = i; j >= brecha && array[j - brecha] > temp; j -= brecha) {
                    array[j] = array[j - brecha];
                    troca++;
                    iteracao++;
                }
                array[j] = temp;
                iteracao++;
            }
        }
        long fim = System.nanoTime();
        return new Resultado((fim - inicio) / 1e6, troca, iteracao);
    }
    public static double[] executaShellSort(int tamanhoVetor, int execucoes) {
        double totalShellTempo = 0;
        int totalShellTroca = 0;
        int totalShellIteracao = 0;

        for (int j = 0; j < execucoes; j++) {
            int[] array = Utilidades.geraRandomArray(tam: tamanhoVetor, seed: 19);
            Resultado resultado = ordenar(array);
            totalShellTempo += resultado.tempo;
            totalShellTroca += resultado.troca;
            totalShellIteracao += resultado.iteracao;}
        double mediaTempo = totalShellTempo / execucoes;
        double mediaTroca = (double) totalShellTroca / execucoes;
        double mediaIteracao = (double) totalShellIteracao / execucoes;
        return new double[]{mediaTempo, mediaTroca, mediaIteracao};
```

**Classe ShellSort**: Contém dois métodos ordenar e executaShellSort. Justificativa: Para maior organização do código optei por ter classes para o shell sort e gnome sort com o máximo de lógica remetente aos mesmos, isto mantém claro seu propósito.

- **Método ordenar:** Ele implementa o algoritmo Shell Sort, o qual utiliza a técnica de divisão por brechas (gaps) para ordenar os elementos do array. O método percorre o array com uma brecha inicial na metade do tamanho do array e a reduz progressivamente, assim realizando comparações e movimentando elementos para as posições corretas. Durante a execução ele também rastreia o número de tempo, trocas e iterações realizadas.

- **Método executaShellSort:** Tem como objetivo executar o Shell Sort repetidas vezes em acordo com execuções para um tamanho de vetor específico ambos de acordo com o enunciado do trabalho. Para cada execução, ele gera um array aleatório, chama o método ordenar e acumula os valores de tempo, trocas e iterações. Por último, ele calcula a média desses valores.



```
class GnomeSort {
    public static Resultado ordenar(int[] array) {
        int troca = 0;
        int iteracao = 0;
        long inicio = System.nanoTime();
        int index = 0;
        while (index < Utilidades.comprimento(array)) {
            if (index == 0 || array[index] >= array[index - 1]) {
                index++;
            } else {
                int temp = array[index];
                array[index] = array[index - 1];
                array[index - 1] = temp;
                troca++;
                index--;
            }
            iteracao++;
        }
        long fim = System.nanoTime();
        return new Resultado((fim - inicio) / 1e6, troca, iteracao);
    }
    public static double[] executaGnomeSort(int tamanhoVetor, int execucoes) {
        double totalGnomeTempo = 0;
        int totalGnomeTroca = 0;
        int totalGnomeIteracao = 0;

        for (int j = 0; j < execucoes; j++) {
            int[] array = Utilidades.geraRandomArray(tam: tamanhoVetor, seed: 19);
            Resultado resultado = ordenar(array);
            totalGnomeTempo += resultado.tempo;
            totalGnomeTroca += resultado.troca;
            totalGnomeIteracao += resultado.iteracao;}
        double mediaTempo = totalGnomeTempo / execucoes;
        double mediaTroca = (double) totalGnomeTroca / execucoes;
        double mediaIteracao = (double) totalGnomeIteracao / execucoes;
        return new double[]{mediaTempo, mediaTroca, mediaIteracao};
```

**Classe GnomeSort**: Contém dois métodos principais ordenar e executaGnomeSort. Justificativa: Para maior organização do código optei por ter classes para o shell sort e gnome sort com o máximo de lógica remetente aos mesmos, isto mantém claro seu propósito.

- **Método Ordenar:** Implementa o algoritmo Gnome Sort, o qual utiliza uma técnica em que os elementos são trocados até que estejam na posição correta, subindo ou descendo pelo array conforme necessário. Durante o processo de ordenação, ele verifica se o elemento atual é maior ou igual ao anterior e caso não seja, realiza uma troca e retrocede uma posição. O método rastreia o número de trocas e iterações realizadas e calcula o tempo total de execução assim como a media.

- **Método ExecutaGnomeSort:** Funcionamento similar ao executaShellSort adaptado para Gnome; tem como objetivo executar o Gnome Sort repetidas vezes em acordo com execuções para um tamanho de vetor específico ambos de acordo com o enunciado do trabalho. Para cada execução, ele gera um array aleatório, chama o

método ordenar e acumula os valores de tempo, trocas e iterações. Por último, ele calcula a média desses valores.

```java
public class RA03 {

    public static void main(String[] args) {
        int[] tam = {1000, 10000, 100000, 500000, 1000000};
        double[] tamanhoVetor = new double[Utilidades.comprimento(array: tam)];

        double[] tempoShellSort = new double[Utilidades.comprimento(array: tam)];
        double[] trocaShellSort = new double[Utilidades.comprimento(array: tam)];
        double[] iteracaoShellSort = new double[Utilidades.comprimento(array: tam)];

        double[] tempoGnomeSort = new double[Utilidades.comprimento(array: tam)];
        double[] trocaGnomeSort = new double[Utilidades.comprimento(array: tam)];
        double[] iteracaoGnomeSort = new double[Utilidades.comprimento(array: tam)];

        for (int i = 0; i < Utilidades.comprimento(array: tam); i++) {
            int t = tam[i];
            tamanhoVetor[i] = t;
            // shell
            double[] resultadosShell = ShellSort.executaShellSort(tamanhoVetor: t, execucoes:5);
            tempoShellSort[i] = resultadosShell[0];
            trocaShellSort[i] = resultadosShell[1];
            iteracaoShellSort[i] = resultadosShell[2];
            System.out.println(x: "----------------------------");
            System.out.println("Shell Sort - Tamanho do vetor: " + t);
            System.out.println("Media de tempo (ms): " + tempoShellSort[i]);
            System.out.println("Media de trocas: " + trocaShellSort[i]);
            System.out.println("Media de iteracoes: " + iteracaoShellSort[i]);
            //gnome
            double[] resultadosGnome = GnomeSort.executaGnomeSort(tamanhoVetor: t, execucoes:5);
            tempoGnomeSort[i] = resultadosGnome[0];
            trocaGnomeSort[i] = resultadosGnome[1];
            iteracaoGnomeSort[i] = resultadosGnome[2];
            System.out.println("Gnome Sort - Tamanho do vetor: " + t);
            System.out.println("Media de tempo (ms): " + tempoGnomeSort[i]);
            System.out.println("Media de trocas: " + trocaGnomeSort[i]);
            System.out.println("Media de iteracoes: " + iteracaoGnomeSort[i]);
```

**Classe RA03**: Nela são executados os métodos de ordenação e é gerado o gráfico final que compara o desempenho dos algoritmos. Justificativa: O main lida com a construção do código em si chamando métodos para auxiliá-lo.

- **Método main**: O método principal que executa os testes de desempenho para ambos os algoritmos. Ele define tamanhos diferentes de vetores e realiza cinco testes para cada tamanho assim como descrito na atividade. Em cada teste, ele gera um vetor aleatório, aplica os algoritmos e calcula a média dos resultados, armazenando-os para cada algoritmo e tamanho de vetor.

```java
        // Gráficos
        String[] serieShell = {"Tempo Shell Sort (ms)", "Trocas Shell Sort", "Iterações Shell Sort"};
        double[][] yDadoShell = {tempoShellSort, trocaShellSort, iteracaoShellSort};

        String[] serieGnome = {"Tempo Gnome Sort (ms)", "Trocas Gnome Sort", "Iterações Gnome Sort"};
        double[][] yDadoGnome = {tempoGnomeSort, trocaGnomeSort, iteracaoGnomeSort};

        Utilidades.gerarGrafico("Desempenho do Shell Sort", "Tamanho do Vetor", "Tempo (ms) / Contagem", tamanhoVetor, serieShell, yDadoShel
        Utilidades.gerarGrafico("Desempenho do Gnome Sort", "Tamanho do Vetor", "Tempo (ms) / Contagem", tamanhoVetor, serieGnome, yDadoGnom
    }
}
```

- **Gráficos**: O código usa o método da classe Utilidades, gerarGrafico para criar gráficos comparativos do desempenho dos algoritmos *Shell Sort* e *Gnome Sort*. Os gráficos mostram o tempo médio de execução, contagem de trocas e o número de iterações em função do tamanho do vetor. Justificativa: Optei por utilizar XChart.

```java
public class Utilidades {
    public static int[] geraRandomArray(int tam, long seed) {
        Random rand = new Random(seed);
        int[] array = new int[tam];
        for (int i = 0; i < tam; i++) {
            array[i] = rand.nextInt();
        }
        return array;
    }
    public static double medeEmMilise(Runnable task) {
        long inicio = System.nanoTime();
        task.run();
        long fim = System.nanoTime();
        return (fim - inicio) / 1e6;
    }
    public static void gerarGrafico(String titulo, String xAxisTitle, String yAxisTitle,
                                    double[] xDado, String[] serieNome, double[][] yDado) {

        XYChart chart = new XYChartBuilder().width(width: 800).height(height:600)
                .title(title: titulo)
                .xAxisTitle(xAxisTitle)
                .yAxisTitle(yAxisTitle)
                .build();

        for (int i = 0; i < comprimentoString(array: serieNome); i++) {
            chart.addSeries(serieNome[i], xData: xDado, yDado[i]);
        }

        new SwingWrapper<>(chart).displayChart();
    }
```

**Classe Utilidades:** Uma classe de suporte que é utilizada no RA03 e RA04. Justificativa: Escolhi fazer desta maneira pois quis evitar redundância de componentes que são utilizados em ambos os RAs e facilitar legibilidade nos mesmos.

## ANÁLIZE DO RESULTADO

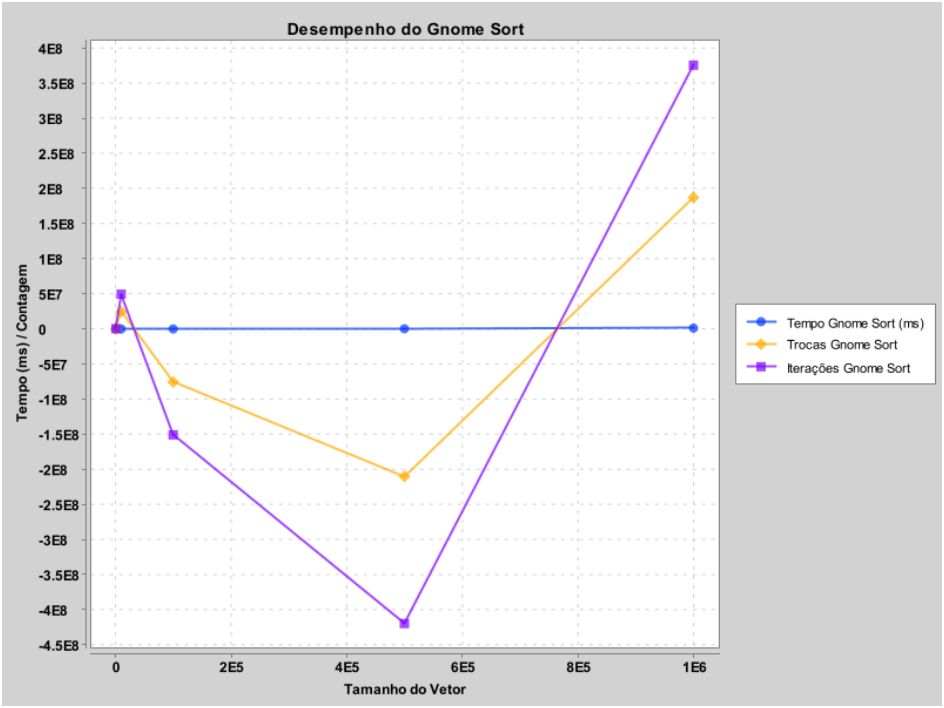
Foram realizados testes com valores abaixo dos propostos pela atividade além dos também propostos. Com base nos resultados, é evidente que há uma anomalia significativa

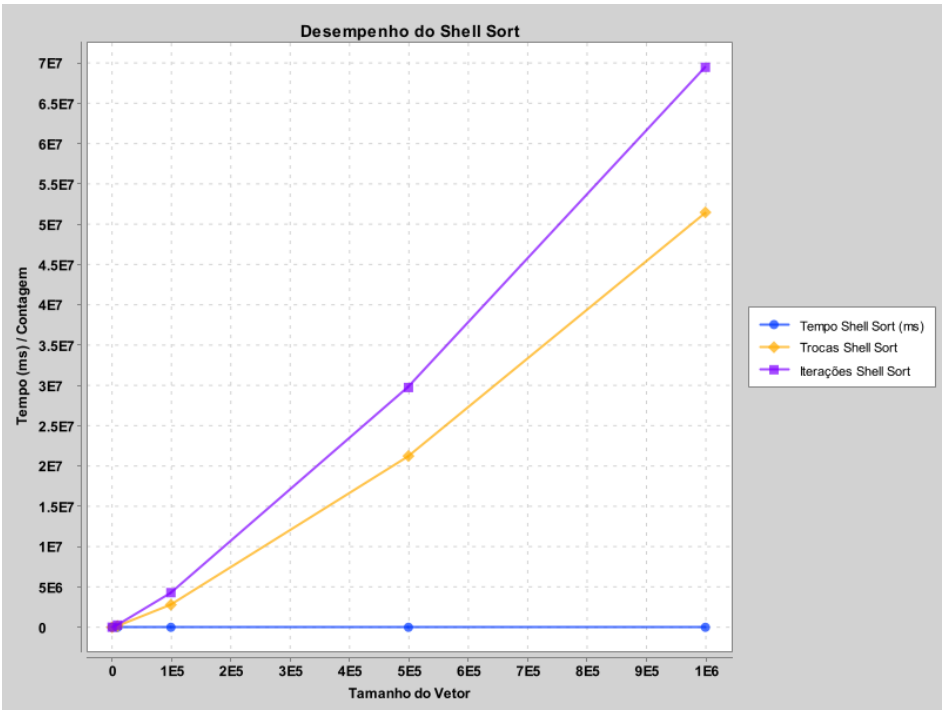
na análise do gráfico do Gnome Sort; isso ocorre porque em meus testes ele apresenta consistência apenas para tamanhos de entrada geralmente menores que 10.000 elementos. O Gnome Sort possui a característica de exibir picos acentuados em iterações, diferente do Shell Sort, que apresenta resultados dentro do esperado.

Concluo que o Shell Sort é mais adequado para quantidades de dados maiores, como as propostas na atividade (1000, 10000, 100000, 500000, 1000000), pois o Gnome Sort essencialmente move os elementos para frente e para trás quando estão fora de ordem, resultando em muitas etapas redundantes e trocas em matrizes maiores. Em ambos os testes com os parametros da atividade, foi necessário cerca de 2 a 3 horas para compilar o código e gerar os gráficos.

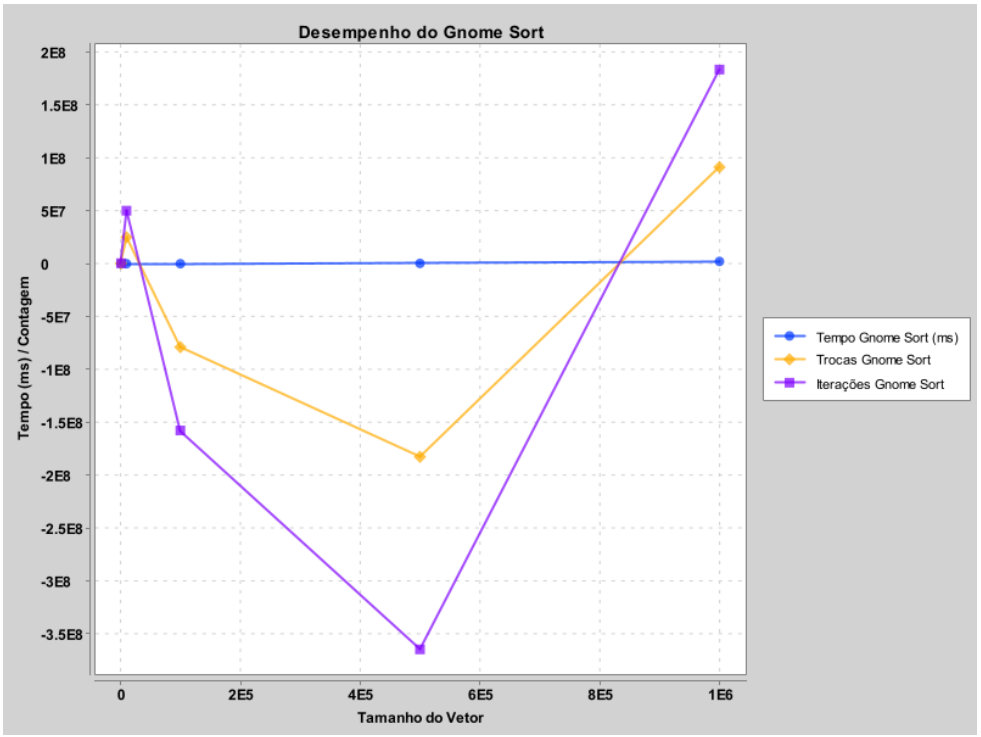
BUILD SUCCESSFUL (total time: 131 minutes 3 seconds)
BUILD SUCCESSFUL (total time: 175 minutes 44 seconds)

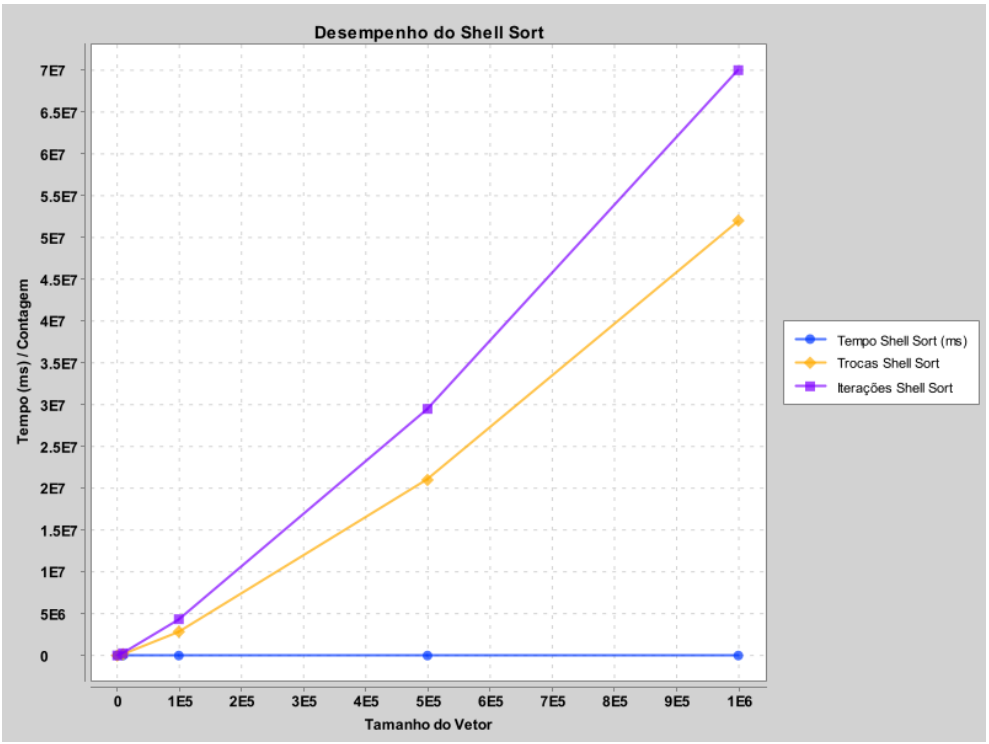
**Primeira execução**




## Segunda Execução


Desempenho do Gnome Sort
Tempo (ms) / Contagem
2E8
1.5E8
1E8
5E7
0
-5E7
-1E8
-1.5E8
-2E8
-2.5E8
-3E8
-3.5E8
0
2E5
4E5
6E5
8E5
1E6
Tamanho do Vetor
Tempo Gnome Sort (ms)
Trocas Gnome Sort
Iterações Gnome Sort


Desempenho do Shell Sort
Tempo (ms) / Contagem
7E7
6.5E7
6E7
5.5E7
5E7
4.5E7
4E7
3.5E7
3E7
2.5E7
2E7
1.5E7
1E7
5E6
0
0
1E5
2E5
3E5
4E5
5E5
6E5
7E5
8E5
9E5
1E6
Tamanho do Vetor
Tempo Shell Sort (ms)
Trocas Shell Sort
Iterações Shell Sort