ALGORITMOS, ESTRUTURA DE DADOS E PROGRAMAÇÃO – ATIVIDADE 1

GIOVANNA LARA DOS SANTOS, RA 24.01779-5

GUILHERME LOSCHIAVO ALVAREZ, RA 23.00821-0

LUISA LÉRIO LEITE, RA 24.01218-0

TIAGO TOKUGI DE ALBUQUERQUE MASSUDA, RA 24.01217-3

1. Introdução

Esse documento é relativo à atividade do primeiro bimestre, em Java, foi implementado uma classe de vetor com funções para preenchimento, ordenação e busca de um elemento, a atividade consiste de criar um vetor com tamanho 100 mil, 200 mil, 400 mil, 800 mil ou 1,6 milhões e fazer a busca de um elemento seguindo um algoritmo linear (indo casa por casa e verificando) em seguida ordenamos, registrando a quantidade de comparações e trocas nessa ordenação e depois fazemos a busca do mesmo número com um algoritmo de busca binária. Após a coleta de dados, comparamos os três algoritmos de ordenação e seus custos com o custo da busca linear e binária para saber o quanto vale a ordenação para fazer a busca binária.

2. Implementação dos algoritmos

A implementação dos algoritmos foi em Java e seguia ao estudado na sala com uma diferença na contagem feita nos algoritmos, ao invés de retornarem um "swap" do vetor, optamos por contar a quantidades de atribuições feitas e comparações necessárias para ordenar o vetor, segue os códigos:

```java
public long selectionsort() {
    long counter = 0;
    for (int i = 0; i < vetor.length - 1; ++i) {
        int min = i;
        for (int j = i + 1; j < vetor.length; ++j) {
            counter++;
            if (vetor[j] < vetor[min]) {
                min = j;
                counter++;
            }
        }

        if (min != i) {
            int x = vetor[i];
            vetor[i] = vetor[min];
            vetor[min] = x;
            counter += 3;
        }
    }
    return counter;
}
```

```java
public long insertionSort() {
    long counter = 0;
    for (int j = 1; j < vetor.length; ++j) {
        int x = vetor[j];
        int i;
        for (i = j - 1; i >= 0 && vetor[i] > x; --i) {
            counter++;
            vetor[i + 1] = vetor[i];
        }
        vetor[i + 1] = x;
        counter++;
    }
    if (counter == 0) {
        System.out.println("Vetor já ordenado");
    }
    return counter;
}
```

```java
public long bubbleSort() {
    long counter = 0;
    boolean trocou = true;
    for (int i = 0; i < ocupacao - 1 && trocou; i++) {
        trocou = false;
        for (int j = 0; j < ocupacao - i - 1; j++) {
            counter++;
            if (vetor[j] > vetor[j + 1]) {
                int aux = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = aux;
                counter+=3;
                trocou = true;
            }
        }
    }
    return counter;
}
```

Os algoritmos de busca implementados são os seguintes:

- Busca Linear:

```java
public long buscaLinear(int elemento) {
    long contadorLinear = 0;

    for (int i = 0; i < this.vetor.length; i++) {
        contadorLinear++;
        if (this.vetor[i] == elemento) {
            // System.out.prlongln(i);
            return contadorLinear;
        }
    }
    return contadorLinear;
}
```

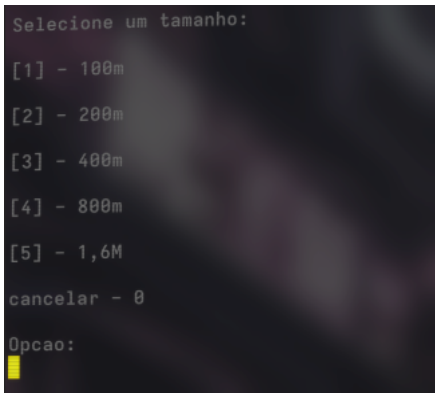
- Busca Binária:

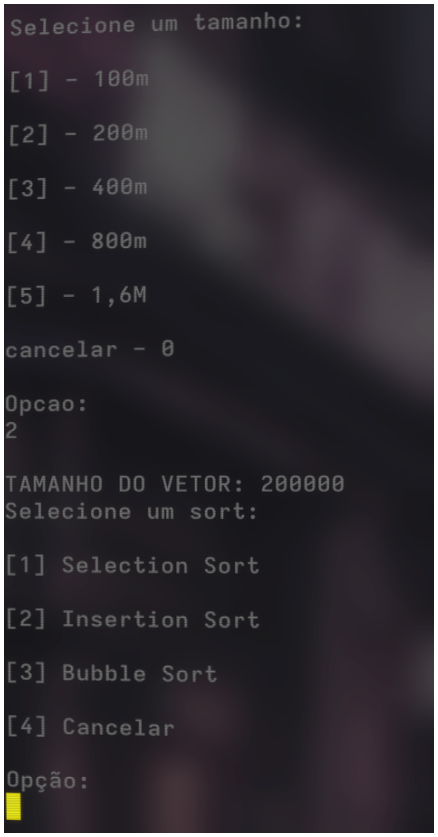
```java
public long buscaBinaria(int elemento) {
    long contadorBinario = 0;

    int inicio = 0;
    int fim = this.vetor.length - 1;
    while (inicio ⩽ fim) {
        int meio = ((inicio + fim) / 2);
        contadorBinario++;
        if (this.vetor[meio]  == elemento ) {
            return contadorBinario;
        }
        contadorBinario++;
        if (this.vetor[meio] < elemento ) {
            inicio = meio + 1;
        } else {
            fim = meio - 1;
        }
    }
    return contadorBinario;
}
```

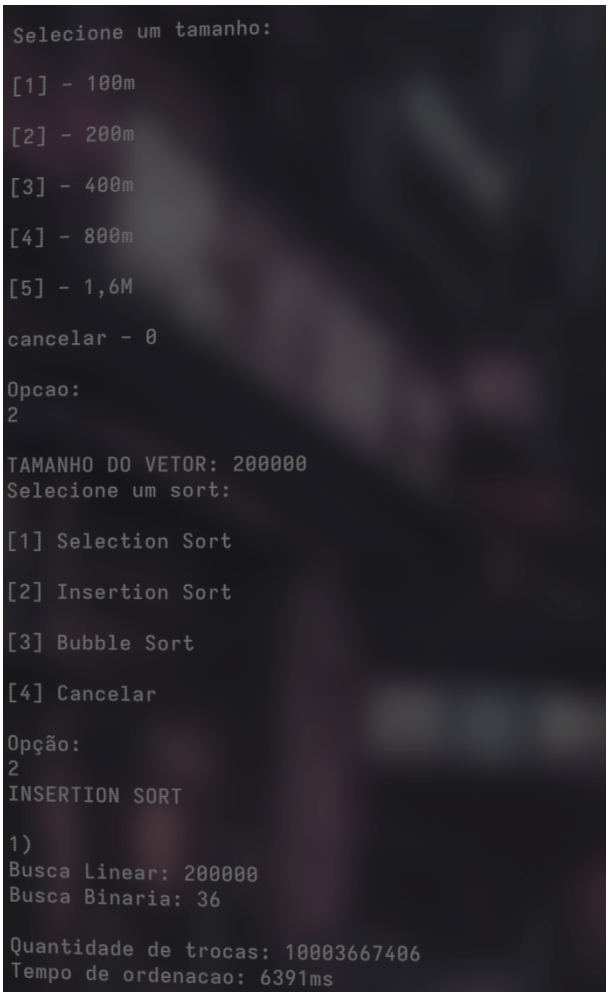
3. Interface

A interface foi feita com um loop do while onde o usuário seleciona o tamanho do vetor e depois qual algoritmo de ordenação deseja usar, sendo a opção de tamanho do vetor de 100.000, 200.000, 400.000, 800.000 e 1.600.000 e as opções de ordenação o bubbleSort, selectionSort e o insertionSort. O código para exibir o menu está ao final deste documento, segue registros da saída no terminal:





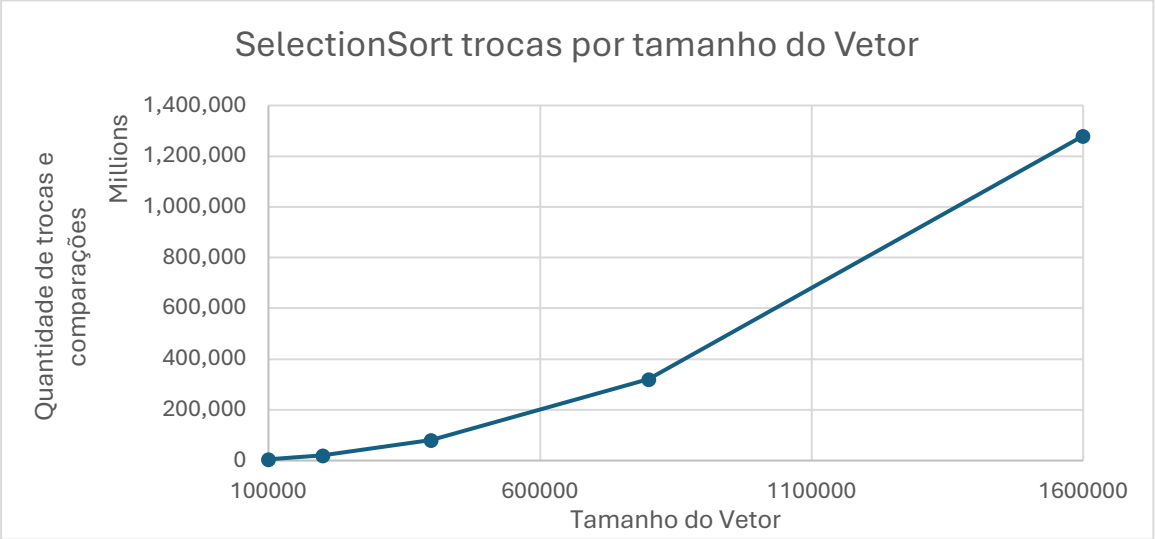
Além disso, todos os códigos e resultados podem ser vistos no repositório: https://github.com/tokujiTO/algoritmos-estrutura-de-dados/tree/main/ATIVIDADES/atividade1



4. Análise

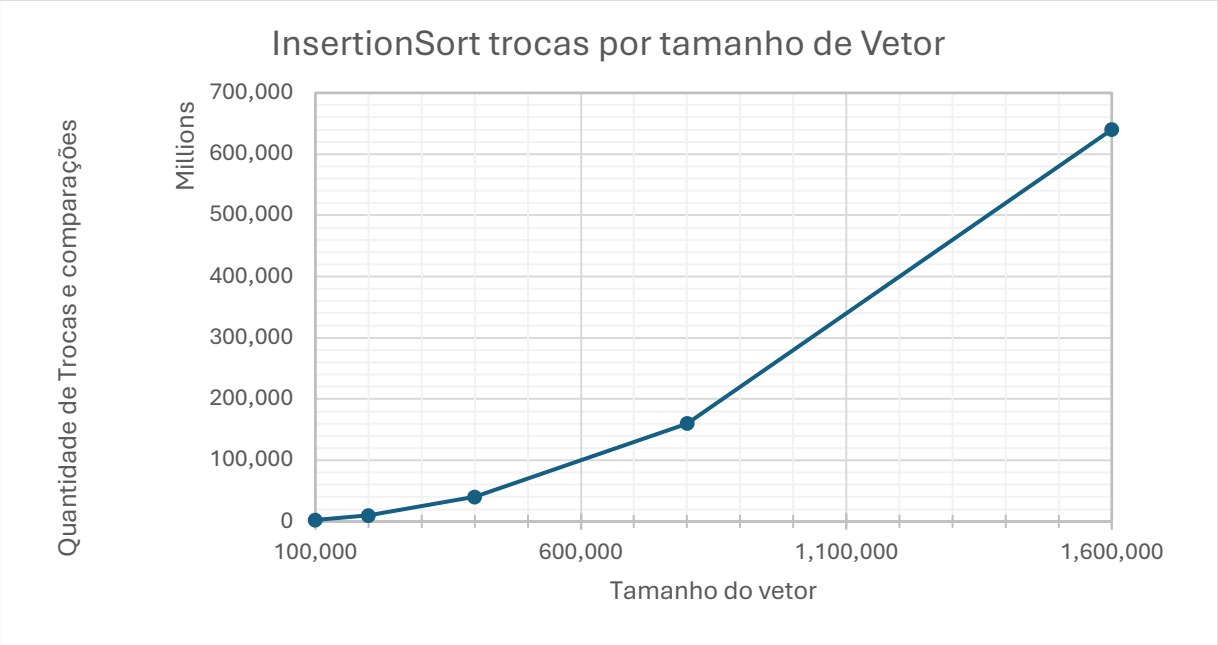
a. Ordenação por Seleção

A ordenação por seleção funciona percorrendo a lista e resgatando o menor resultado e "jogando" para o lado esquerdo, gerando assim durante seu funcionamento uma parte do vetor já ordenado e outra não ordenada percorrendo a cada ciclo a parte não ordenada e colocando o menor elemento na parte ordenada. O gráfico a seguir mostra a quantidade de trocas e comparações por tamanho de vetor no algoritmo de seleção:



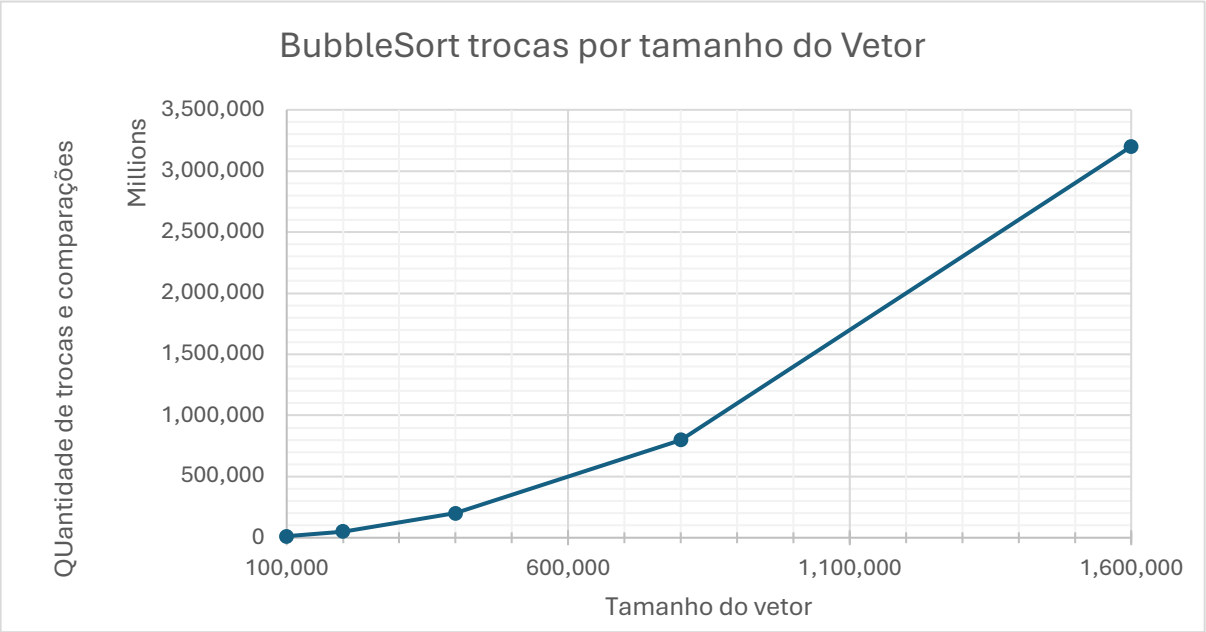
b. Ordenação por Inserção

Ordenação por Inserção opera de forma semelhante à organização de cartas em um baralho. Ele percorre o vetor da esquerda para a direita, inserindo cada elemento em sua posição correta dentro da parte já ordenada. A cada iteração, um elemento é selecionado e comparado com os elementos à sua esquerda, deslocando-os para a direita até encontrar sua posição adequada. O gráfico a seguir mostra a quantidade de trocas e comparações por tamanho de vetor no algoritmo de seleção:



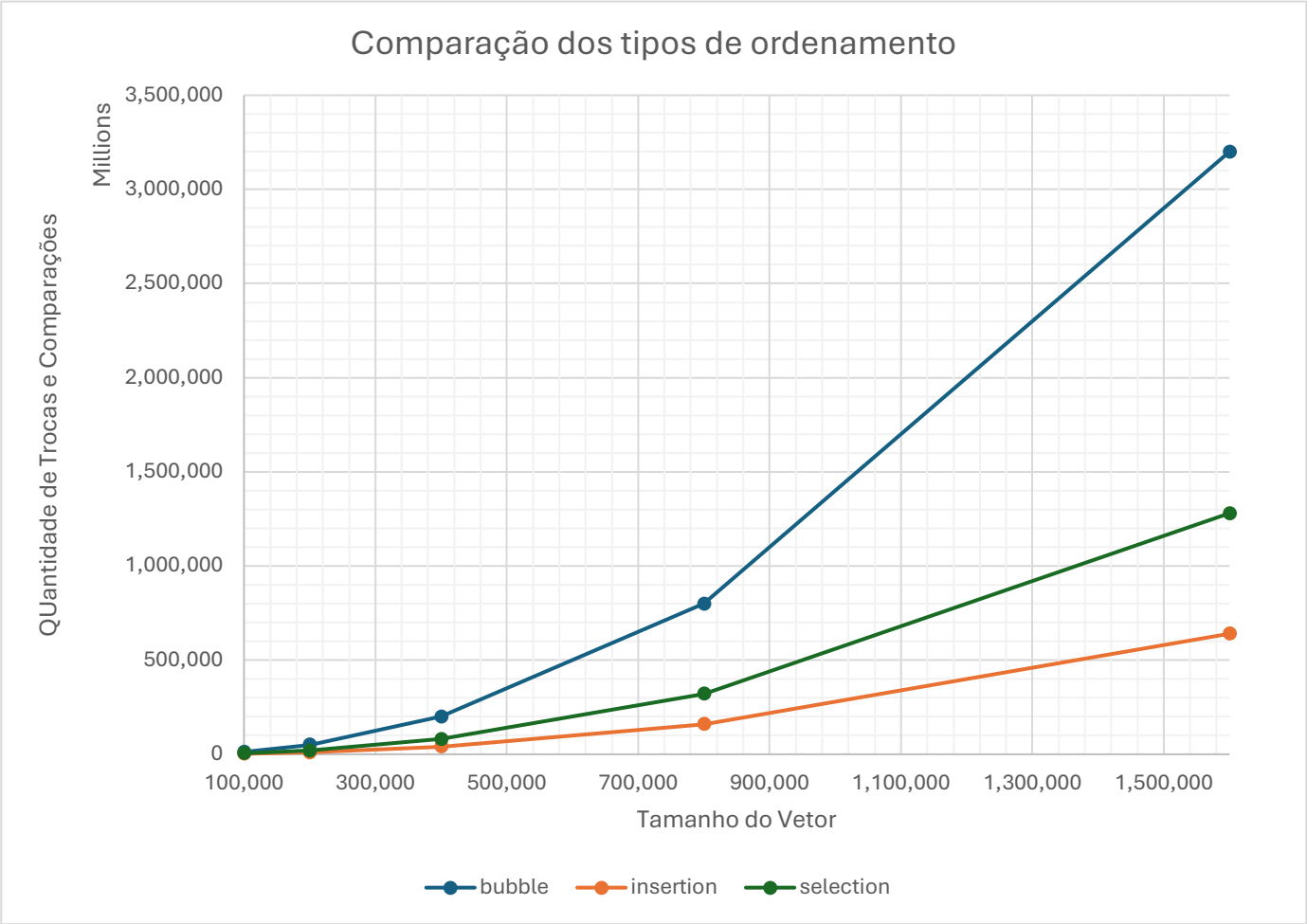
c. Ordenação por Bolha

O algoritmo por bolha funciona percorrendo repetidamente a lista, comparando elementos adjacentes e trocando-os se estiverem na ordem errada. A cada passagem completa, o maior elemento "flutua" para o final do vetor (como uma bolha), criando uma parte ordenada à direita e uma parte não ordenada à esquerda. Esse processo se repete até que nenhuma troca seja necessária, indicando que o vetor está completamente ordenado. O gráfico a seguir mostra a quantidade de trocas e comparações por tamanho de vetor no algoritmo de seleção:



d. Comparação

Para a comparação dos três algoritmos, foi plotado um gráfico com a quantidade de trocas e comparações feitas em cada um dos gráficos pelo tamanho do vetor:

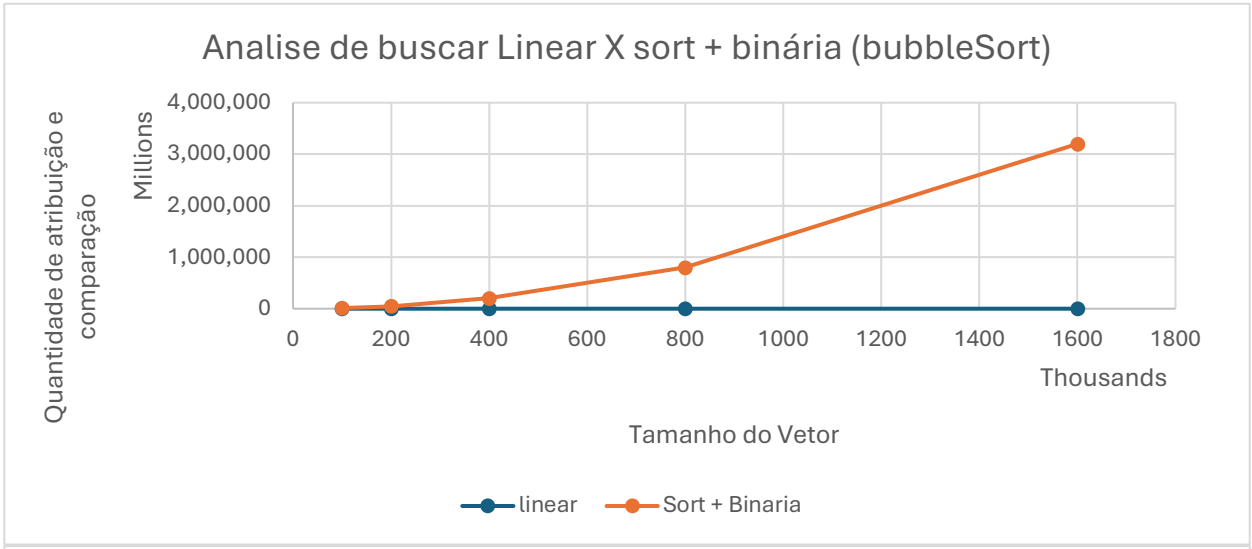


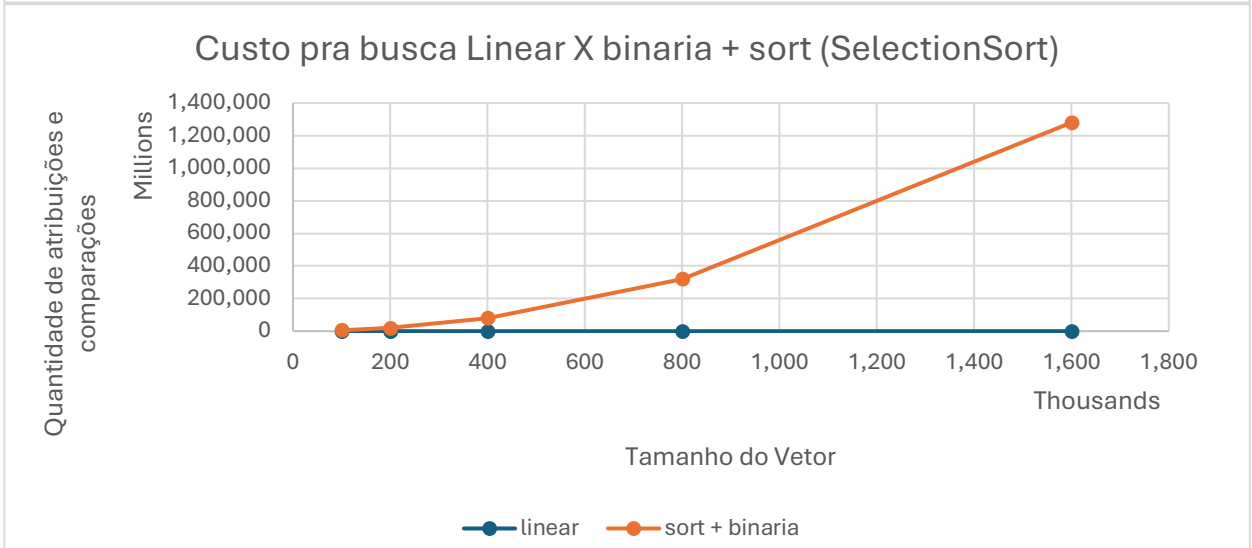
O Bubble Sort é disparado o pior. Seu número de operações cresce muito mais rápido do que os outros, porque ele faz muitas comparações e trocas desnecessárias. O Selection Sort é um pouco melhor, pois faz menos trocas, mas ainda é bem ineficiente para listas grandes. Já o Insertion Sort se sai melhor, principalmente quando os dados já estão quase ordenados.

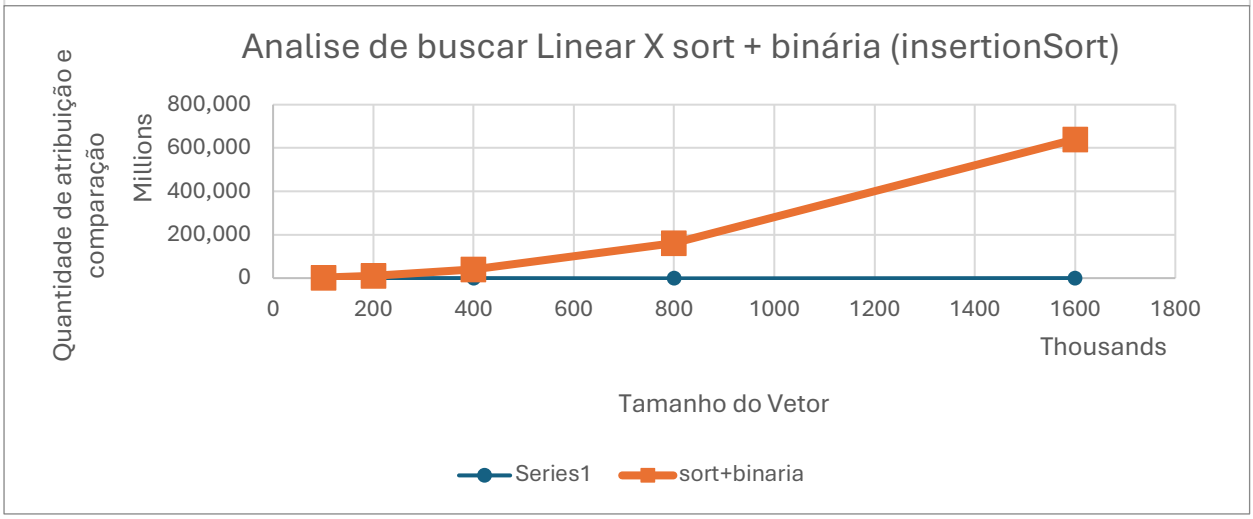
Isso ocorre por conta de algumas propriedades, no caso do bubbleSort, ele fica maior por conta da sua lógica de “arrastar” os números pelo vetor todo, quanto ao selectionSort, percorre a parte não organizada diversas vezes pegando o menor número independente da ordenação atual, já o insertionSort faz comparações e trocas locais enquanto desloca os elementos para a localização correta e tem o melhor dos casos sendo um total de n-1 (Ordem O(n)) iterações no vetor, enquanto todos os outros são da ordem de n^2 ($O(n^2)$)

e. Busca binária e linear

Analisando o custo da busca binária e da busca linear, fica claro que a busca binária é bem mais eficiente e menos custosa, entretanto, ela tem um custo a mais que seria o de ordenar o vetor, durante os nossos testes foi observado que ao somar o custo de ordenação com o custo de busca binária, em nenhum dos casos compensou ordenar o vetor para fazer a busca binária, sendo a busca linear simples relativamente menos custoso, no sentido de alocação e quantidade de comparação:







Assim chega-se à conclusão que ainda que tenhamos uma economia grande utilizando a busca binária no lugar da linear, o custo pago para fazer a ordenação torna esse processo mais custoso do que fazer a busca linear simples.

5. Nossas tabelas:

a. SelectionSort:

tamanho do vetor	trocas	linear	binaria	sort + binaria
100000	5,001,256,183	98342.40	33.40	5001256216.40
200000	20,002,652,536	190017.90	35.30	20002652571.30
400000	80,005,579,117	353340.60	36.60	80005579153.60
800000	320,011,712,852	749984.70	39.10	320011712891.10
1600000	1,280,024,551,410	1600000.00	41.60	1280024551451.60

b. InsertionSort:

tamanho do vetor	trocas	linear	binaria	Sort + Binaria
100000	2,499,498,964	93339,2	33.0	2,499,498,997.0
200000	10,003,455,881	195000,8	35.0	10,003,455,916.0
400000	39,998,391,860	376668,8	37.1	39,998,391,897.1
800000	160,002,414,205	749986,8	39.3	160,002,414,244.3
1600000	640,123,779,691	1600000,0	41.0	640,123,779,732.0

c. BubbleSort:

tamanho do vetor	trocas	linear	binaria	Sort + Binaria
100000	12,500,804,057	96,662.4	33.1	12,500,804,090.1
200000	50,005,376,488	180005,9	35.1	50,005,376,523.1
400000	200,018,469,882	376673,3	37.2	200,018,469,919.2
800000	799,900,122,889	775,005.8	39.2	799,900,122,927.8
1600000	3,200,181,389,331	1,444,970.2	40.9	3,200,181,389,371.8

d. Geral:

tamanho do vetor	bubble	insertion	selection
100000	12,500,804,057	2,499,498,964	5,001,256,183
200000	50,005,376,488	10,003,455,881	20,002,652,536
400000	200,018,469,882	39,998,391,860	80,005,579,117
800000	799,900,122,889	160,002,414,205	320,011,712,852
1600000	3,200,181,389,331	640,123,779,691	1,280,024,551,410

6. Código do menu:

```java
import java.util.Scanner;

public class Teste {
    Run | Debug
    public static void main(String[] args) {
        // vamos testar a eficácia dos métodos de ordenação
        Scanner scanner = new Scanner(System.in);
        int tamanho = 0;
        boolean segue = true;
        do {
            System.out.println(x:"\n\nSelecione um tamanho: \n");
            System.out.println(x:"[1] - 100m\n");
            System.out.println(x:"[2] - 200m\n");
            System.out.println(x:"[3] - 400m\n");
            System.out.println(x:"[4] - 800m\n");
            System.out.println(x:"[5] - 1,6M\n");
            System.out.println(x:"cancelar - 0\n");

            System.out.println(x:"Opcao: ");
            tamanho = scanner.nextInt();

            switch (tamanho) {
                case 1:
                    tamanho = 100000;
                    break;
                case 2:
                    tamanho = 200000;
                    break;
                case 3:
                    tamanho = 400000;
                    break;
                case 4:
                    tamanho = 800000;
                    break;
                case 5:
                    tamanho = 1600000;
                    break;
                default:
                    tamanho = 0;
                    System.out.println(x:"Fim das operacoes");
                    segue = false;
                    break;
            }
            ;

            if (!segue)
                break;
```

```java

        System.out.println("\nTAMANHO DO VETOR: " + tamanho);
        NossoVetor vetor = new NossoVetor(tamanho);

        System.out.println(x:"Selecione um sort:\n");
        System.out.println(x:"[1] Selection Sort\n");
        System.out.println(x:"[2] Insertion Sort\n");
        System.out.println(x:"[3] Bubble Sort\n");
        System.out.println(x:"[4] Cancelar\n");

        System.out.println(x:"Opção: ");
        int sort = scanner.nextInt();


        long somaTrocas = 0;
        long somaTempo = 0;
        long somaLinear = 0;
        long somaBinaria = 0;
        long tempoInicial = System.currentTimeMillis();
        double numIteracoes = 30.0;
        if (tamanho == 1600000) {
          System.out.println(x:"vetor de 1.6M !!");
          numIteracoes = 10.0;
        }

        switch (sort) {
          case 1:
            System.out.println(x:"SELECTION SORT");
            for (int i = 0; i < numIteracoes; i++) {
              System.out.println("\n" + (i + 1) + ")");
              long buscaLi = vetor.buscaLinear(elemento:500000);
              somaLinear += buscaLi;
              System.out.println("Busca Linear: " + buscaLi);
              vetor.esvaziaVetor();
              vetor.preencheVetor();
              long tempoInicialOrdenacao = System.currentTimeMillis();
              long iteracoes = vetor.selectionsort();
              long buscaBi = vetor.buscaBinaria(elemento:500000);
              System.out.println("Busca Binaria: " + buscaBi);
              somaBinaria += buscaBi;
              System.out.println("\nQuantidade de trocas: " + iteracoes);
              long tempoFinalOrdenacao = System.currentTimeMillis();
              System.out.println("Tempo de ordenacao: " + (tempoFinalOrdenacao - tempoInicialOrdenacao) + "ms");
              somaTrocas += iteracoes;
              somaTempo += tempoFinalOrdenacao - tempoInicialOrdenacao;
            }
            break;
```

```
93                  break;
94                case 2:
95                  System.out.println(x:"INSERTION SORT");
96                  for (int i = 0; i < numIteracoes; i++) {
97                    System.out.println("\n" + (i + 1) + ")");
98                    long buscaLi = vetor.buscaLinear(elemento:500000);
99                    somaLinear += buscaLi;
100                   System.out.println("Busca Linear: " + buscaLi);
101                   vetor.esvaziaVetor();
102                   vetor.preencheVetor();
103                   long tempoInicialOrdenacao = System.currentTimeMillis();
104                   long iteracoes = vetor.insertionSort();
105                   long buscaBi = vetor.buscaBinaria(elemento:500000);
106                   System.out.println("Busca Binaria: " + buscaBi);
107                   somaBinaria += buscaBi;
108                   System.out.println("\nQuantidade de trocas: " + iteracoes);
109                   long tempoFinalOrdenacao = System.currentTimeMillis();
110                   System.out.println("Tempo de ordenacao: " + (tempoFinalOrdenacao - tempoInicialOrdenacao) + "ms");
111                   somaTrocas += iteracoes;
112                   somaTempo += tempoFinalOrdenacao - tempoInicialOrdenacao;
113                 }
114                 break;
115               case 3:
116                 System.out.println(x:"BUBBLE SORT");
117                 for (int i = 0; i < numIteracoes; i++) {
118                   System.out.println("\n" + (i + 1) + ")");
119                   long buscaLi = vetor.buscaLinear(elemento:500000);
120                   somaLinear += buscaLi;
121                   System.out.println("Busca Linear: " + buscaLi);
122                   vetor.esvaziaVetor();
123                   vetor.preencheVetor();
124                   long tempoInicialOrdenacao = System.currentTimeMillis();
125                   long iteracoes = vetor.bubbleSort();
126                   long buscaBi = vetor.buscaBinaria(elemento:500000);
127                   System.out.println("Busca Binaria: " + buscaBi);
128                   somaBinaria += buscaBi;
129                   System.out.println("\nQuantidade de trocas: " + iteracoes);
130                   long tempoFinalOrdenacao = System.currentTimeMillis();
131                   System.out.println("Tempo de ordenacao: " + (tempoFinalOrdenacao - tempoInicialOrdenacao) + "ms");
132                   somaTrocas += iteracoes;
133                   somaTempo += tempoFinalOrdenacao - tempoInicialOrdenacao;
134                 }
135                 break;
136               default:
137                 System.out.println(x:"Fim das operacoes");
138                 segue = false;
139                 break;
140             }
141
142             if (!segue) break;
143
144             long tempoFinal = System.currentTimeMillis();
145             double tempoTotalSegundos = (tempoFinal - tempoInicial)/1000.0;
146             System.out.printf("\nMedia de trocas: " + (somaTrocas / (long) numIteracoes ) );
147             double mediaSegundos = (somaTempo/numIteracoes)/1000.0;
148             System.out.printf(format:"\nTempo total: %.1f segundos\n", tempoTotalSegundos);
149             System.out.printf(format:"Media para completar: %.1f segundos\n", mediaSegundos);
150             System.out.printf(format:"Media Busca Linear: %.1f\n", (somaLinear / numIteracoes));
151             System.out.printf(format:"Media Busca Binaria: %.1f\n", (somaBinaria / numIteracoes));
152
153
154
155           } while (tamanho != 0);
156           scanner.close();
157         }
158       }
159
```