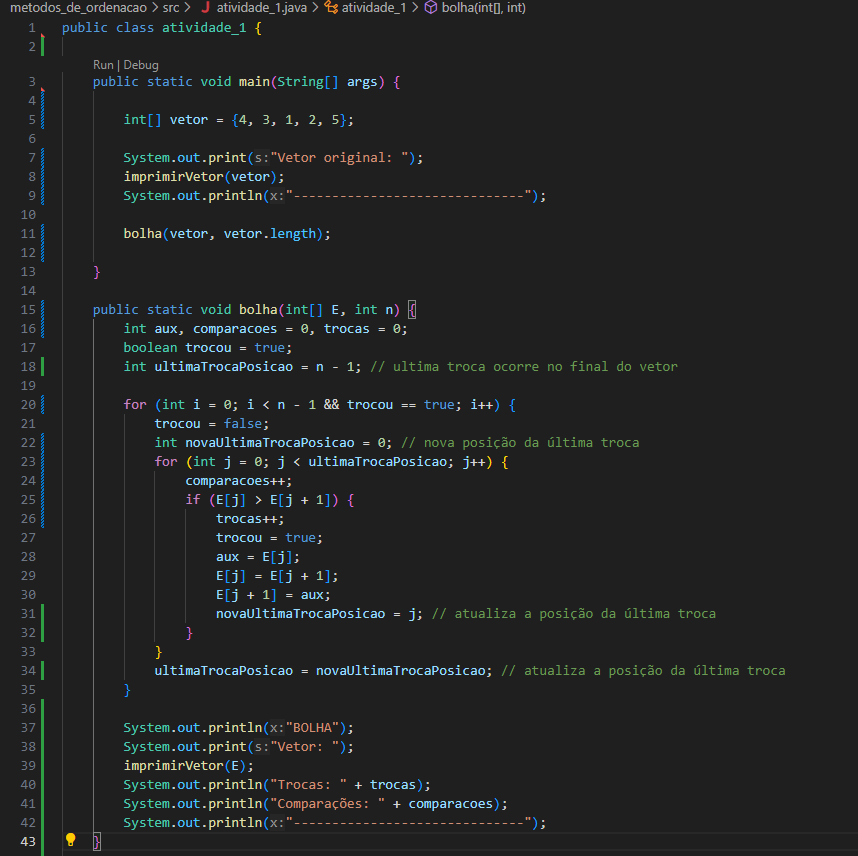
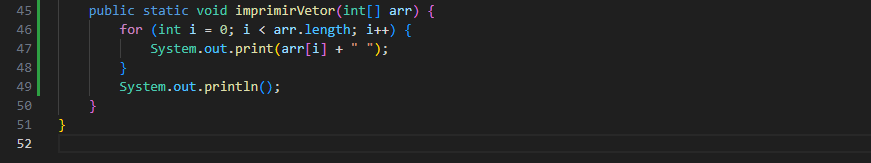
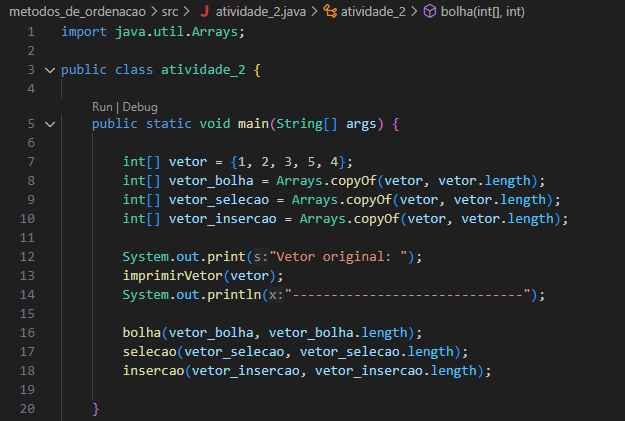
**Nome:** Breno de Sousa Braga

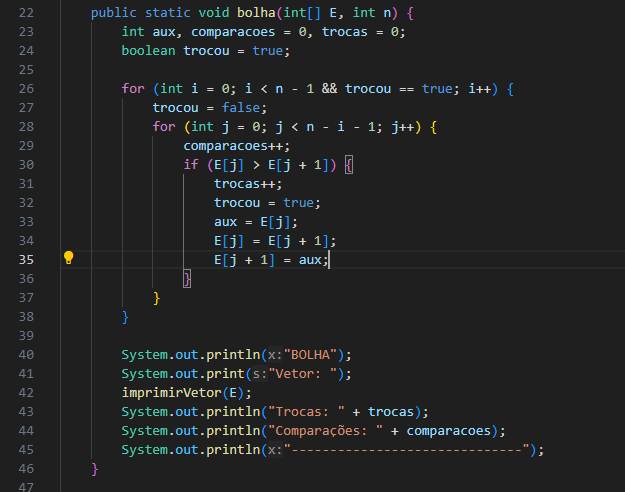
**RA:** 321222619

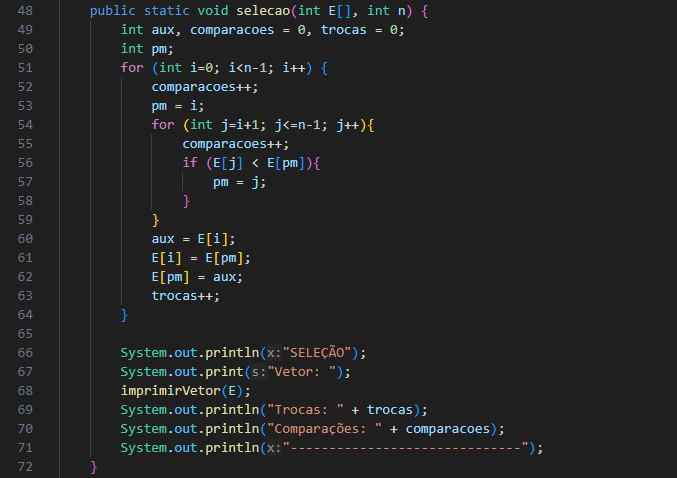
**1.** Adicionar o armazenamento da posição da última troca no algoritmo Bubble Sort pode melhorar a eficiência, uma vez que você pode evitar percorrer o vetor inteiro nas iterações subsequentes, uma vez que você sabe que os elementos após a última troca já estão ordenados. Nesta versão, a variável ultimaTrocaPosicao é usada para rastrear a posição da última troca, e a iteração interna é limitada a essa posição. Isso permite evitar percorrer os elementos que já estão na posição correta.

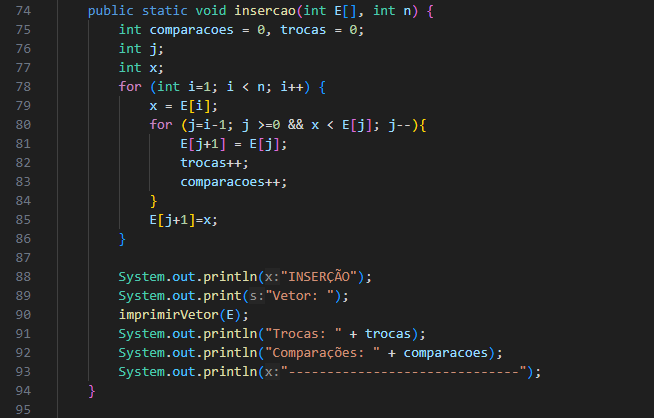
****

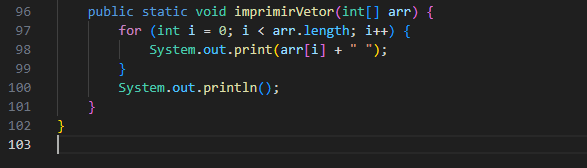
****

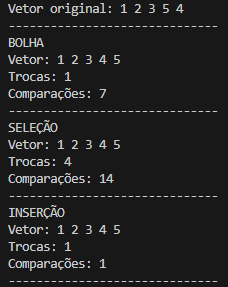
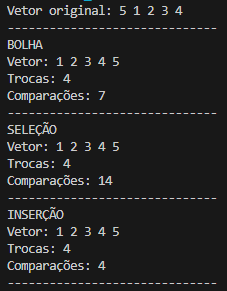
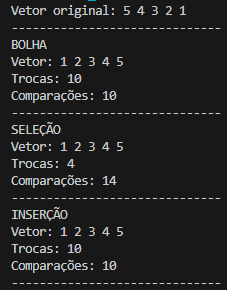
**2.** 







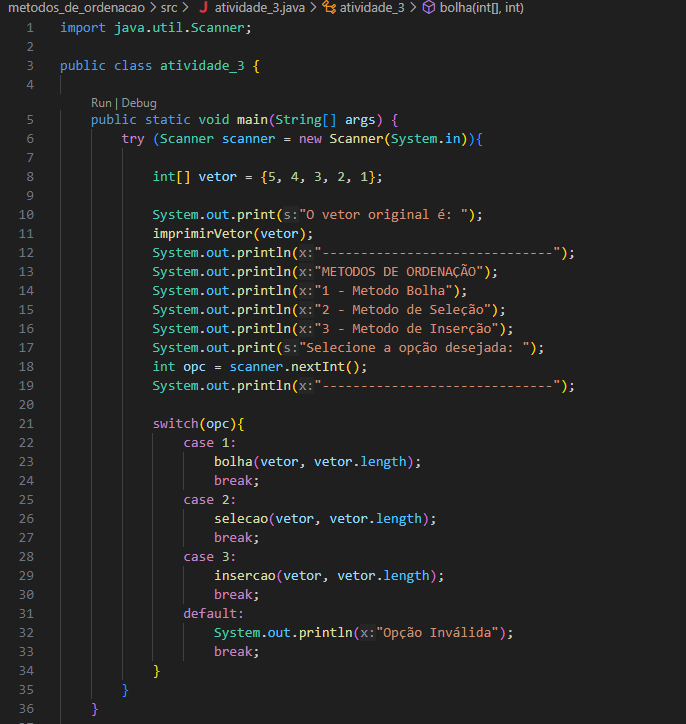


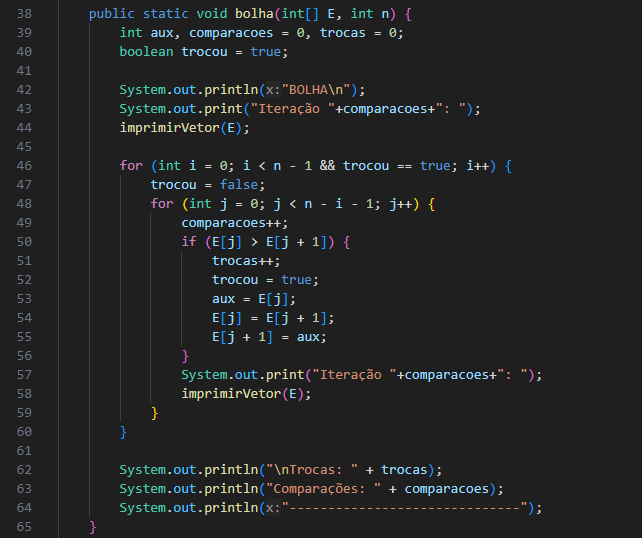
* Um vetor quase ordenado
  + 
* Um vetor parcialmente ordenado
  + 
* Um vetor totalmente desordenado (ordem inversa)
  + 

O método de Bolha em todos os casos precisa realizar muitas comparações para realizar a ordenação. Seu desempenho vai depender do tamanho do vetor. Um vetor muito grande teria um desempenho muito ruim.

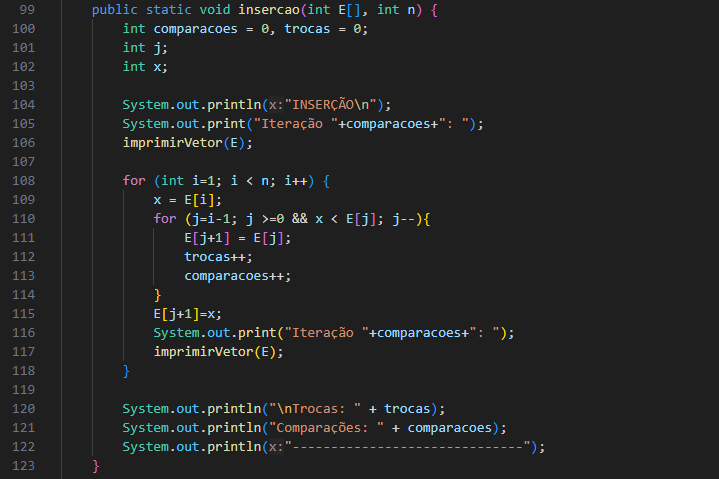
O método de Seleção teve a mesma quantidade de trocas e comparações em todos os casos, é muito interessante para uma quantidade grande de dados, porém com o vetor quase ordenado não ajudou em nada, pois precisa realizar todas comparações.

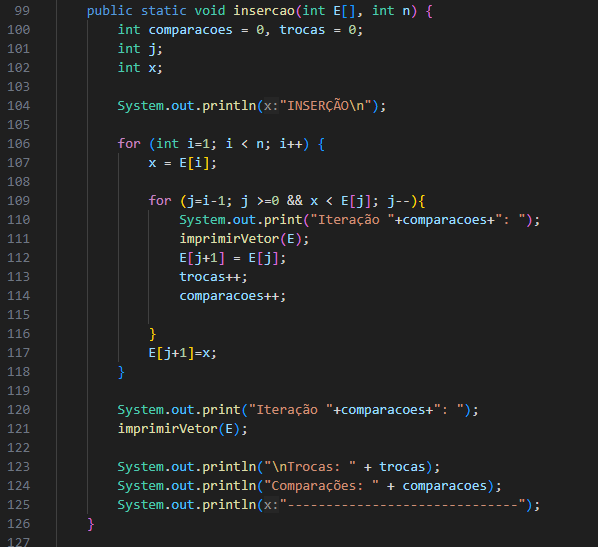
O método de seleção foi o melhor para o vetor quase ordenado, pois precisou apenas de 1 comparação e 1 troca. Porém, não teve nenhuma vantagem quando o vetor estava em ordem inversa, e precisou do número máximo de trocas.

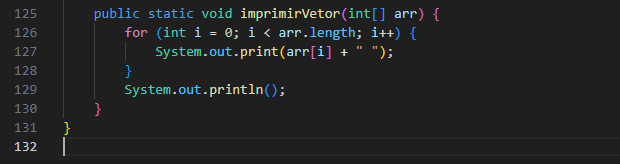
**3. **

****

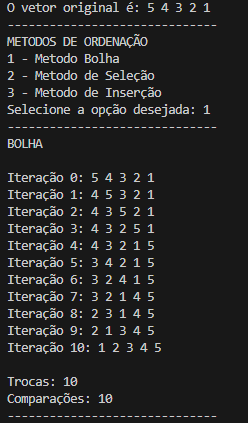
****

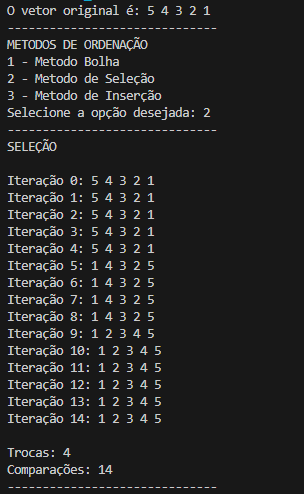
****

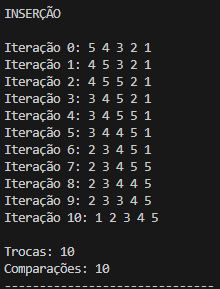
****

****

**Execuções:**

****

****

****

**4.**

* MergeSort

public class atividade\_4\_a {

    public static void main(String[] args) {

        int[] vetor = {3, 4, 9, 2, 5, 8, 2, 1, 7, 4, 6, 2, 9, 8, 5, 1};

        mergeSort(vetor, 0, vetor.length-1);

    }

    public static void mergeSort(int[] arr, int left, int right) {

        if (left < right) {

            int mid = (left + right) / 2;

            mergeSort(arr, left, mid);

            mergeSort(arr, mid + 1, right);

            merge(arr, left, mid, right);

        }

    }

    public static void merge(int[] arr, int left, int mid, int right) {

        int n1 = mid - left + 1;

        int n2 = right - mid;

        int[] leftArr = new int[n1];

        int[] rightArr = new int[n2];

        for (int i = 0; i < n1; i++) {

            leftArr[i] = arr[left + i];

        }

        for (int j = 0; j < n2; j++) {

            rightArr[j] = arr[mid + 1 + j];

        }

        int i = 0, j = 0, k = left;

        while (i < n1 && j < n2) {

            if (leftArr[i] <= rightArr[j]) {

                arr[k] = leftArr[i];

                i++;

            } else {

                arr[k] = rightArr[j];

                j++;

            }

            k++;

        }

        while (i < n1) {

            arr[k] = leftArr[i];

            i++;

            k++;

        }

        while (j < n2) {

            arr[k] = rightArr[j];

            j++;

            k++;

        }

        imprimirVetor(arr);

    }

    public static void imprimirVetor(int[] arr) {

        for (int i = 0; i < arr.length; i++) {

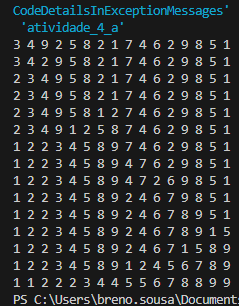
            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

}

****

* QuickSort

public class atividade\_4\_b {

    public static void main(String[] args) {

        int[] vetor = {3, 4, 9, 2, 5, 8, 2, 1, 7, 4, 6, 2, 9, 8, 5, 1};

        quickSort(vetor, 0, vetor.length-1);

    }

    public static void partition(int E[], int left, int right, int[] fe\_ir) {

        int fe, ir;

        int i, j;

        int aux;

        int[] pivot = {(left + right) / 2};

        int pivo = E[pivot[0]];

        i = left;

        j = right;

        do {

            while (E[i] < pivo) i++;

            while (pivo < E[j]) j--;

            if (i <= j) {

                aux = E[i];

                E[i] = E[j];

                E[j] = aux;

                i++;

                j--;

            }

        } while (i <= j);

        fe = j;

        ir = i;

        fe\_ir[0] = fe;

        fe\_ir[1] = ir;

        imprimirVetor(E);

    }

    public static void quickSort(int E[], int left, int right) {

        if (left < right) {

            int[] fe\_ir = new int[2];

            partition(E, left, right, fe\_ir);

            int fe = fe\_ir[0];

            int ir = fe\_ir[1];

            quickSort(E, left, fe);

            quickSort(E, ir, right);

        }

    }

    public static void imprimirVetor(int[] arr) {

        for (int i = 0; i < arr.length; i++) {

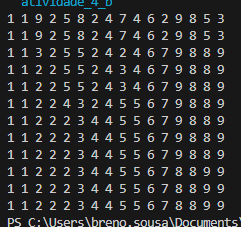
            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

}

****

* ShellSort

public class atividade\_4\_c {

    public static void main(String[] args) {

        int[] vetor = {3, 4, 9, 2, 5, 8, 2, 1, 7, 4, 6, 2, 9, 8, 5, 1};

        shellSort(vetor, vetor.length);

    }

    public static void shellSort(int E[], int n) {

        int i, j, h;

        int x;

        h = 1;

        while (h < n) {

            h = 3 \* h + 1;

        }

        do {

            h = h / 3;

            for (i = h; i < n; i++) {

                x = E[i];

                for (j = i - h; j >= 0 && x < E[j]; j -= h) {

                    E[j + h] = E[j];

                    imprimirVetor(E);

                }

                E[j + h] = x;

            }

        } while (h >= 1);

        imprimirVetor(E);

    }

    public static void imprimirVetor(int[] arr) {

        for (int i = 0; i < arr.length; i++) {

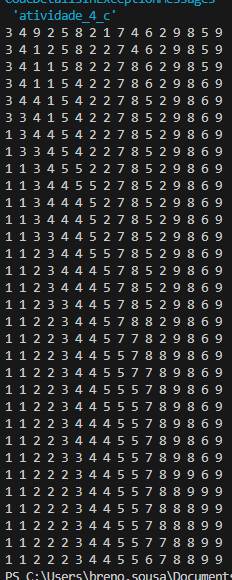
            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

}

****