Bubble Sort

O método Bubble Sort é um algoritmo simples de ordenação que funciona comparando pares de elementos adjacentes e os trocando de lugar se estiverem na ordem errada. Ele continua passando pela lista até que nenhuma troca seja necessária, o que significa que a lista está ordenada.

Código em ordem crescente:

```
public class ordemCrescente {
    public static void main(String[] args) {
    //Altere o tamanho na variavel tam;
    int tam = 100;
    long contadorDeComparacao = 0;
    long contadorDeMovimento = 0;
    int vetor1[]= new int[tam];
    int aux;
    Boolean controle;
    // preenchendo o vetor
    for (int i = 0; i < vetor1.length; i++) {
          vetor1[i]=i;
          }
  //bubble sort ordem crescente
    // for que garante que o processo de comparação ocorra durante todo o tamanho do
vetor
    for (int j = 0; j < vetor1.length; j++) {
        controle = true;
        //responsavel pelas comparaçoes
        for (int k = 0; k < vetor1.length - 1; k++) {
            contadorDeComparacao++;
            if (vetor1[k] > vetor1[k + 1]) {
                aux = vetor1[k];
                vetor1[k] = vetor1[k + 1];
                vetor1[k + 1] = aux;
                contadorDeMovimento++;
                controle = false;
            }
        }
        if (controle) {
            break;
```

```
        }
    }
    System.out.println("Contador de Comparação: " + contadorDeComparacao);
    System.out.println("Contador de Movimento: " + contadorDeMovimento);
    }
}
```

Ordem decrescente:

```
 public class ordemDecrecente {
    public static void main(String[] args) {
        //Altere o tamanho na variavel tam;
        int tam = 100;
        long contadorDeComparacao = 0;
          long contadorDeMovimento = 0;
          int vetor1[] = new int[tam];
          int aux;
        Boolean controle;
            // Preenchendo o vetor
            for (int i = 0; i < vetor1.length; i++) {
                vetor1[i] = i;
            }
            // Bubble sort decrecente
            //for garante que o processo de comparação ocorra durante todo o tamanho
do vetor
            for (int j = 0; j < vetor1.length; j++) {
                controle = true;
                //responsavel pelas comparaçoes
                for (int k = 0; k < vetor1.length - 1; k++) {
                    contadorDeComparacao++;
                    if (vetor1[k] < vetor1[k + 1]) {
                        aux = vetor1[k];
                        vetor1[k] = vetor1[k + 1];
                        vetor1[k + 1] = aux;
                        contadorDeMovimento++;
                        controle = false;
                    }
                }
                if (controle) {
                    break;
```

```
            }
        }
        System.out.println("Contador de Comparação: " + contadorDeComparacao);
        System.out.println("Contador de Movimento: " + contadorDeMovimento);
    }
}
```

Aleatório

```
public class Aleatorio {
    public static void main(String[] args) {
    //Altere o tamanho na variavel tam;
    int tam=100;
    Random aleatorio = new Random();
    long contadorDeComparacao = 0;
    long contadorDeMovimento = 0;
    int vetor1[]= new int[tam];
    int aux;
    Boolean controle;
    // preenchendo o vetor
    for (int i = 0; i < vetor1.length; i++) {
        vetor1[i]=aleatorio.nextInt(101);
          }
  //bubble sort ordem crescente
    // for que garante que o processo de comparação ocorra durante todo o tamanho do
vetor
    for (int j = 0; j < vetor1.length; j++) {
        controle = true;
        //responsavel pelas comparaçoes
        for (int k = 0; k < vetor1.length - j - 1; k++) {
            contadorDeComparacao++;
            if (vetor1[k] > vetor1[k + 1]) {
                aux = vetor1[k];
                vetor1[k] = vetor1[k + 1];
                vetor1[k + 1] = aux;
                contadorDeMovimento++;
                controle = false;
            }
        }
        if (controle) {
```

```
            break;
        }
    }
    System.out.println("Contador de Comparação: " + contadorDeComparacao);
    System.out.println("Contador de Movimento: " + contadorDeMovimento);
    }
}
```

Ordem Crescente

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Bubble Sort				
	100	**0 seg**	**0**	**99**
	1000	**0 seg**	**0**	**999**
	10000	**0 seg**	**0**	**9999**
	100000	**6 seg**	**0**	**99999**

Ordem Decrescente

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Bubble Sort				
	100	**0 seg**	**4950**	**9900**
	1000	**0 seg**	**499500**	**999000**
	10000	**0 seg**	**49995000**	**99990000**
	100000	**13 seg**	**4999950000**	**9999900000**

Ordem Aleatório

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Bubble Sort				
	100	**0 seg**	**2465**	**4830**
	1000	**0 seg**	**248673**	**497085**
	10000	**0 seg**	**24779742**	**49993775**
	100000	**17 seg**	**2504106677**	**4999917104**

Insertion Sort

O Insertion Sort é um algoritmo de ordenamento simples usado para organização de listas. Ele funciona de forma simples e eficiente, sempre ordenando as listas da esquerda para a direita, caso encontre um elemento desordenado, ele pega esse elemento e verifica os anteriores, comparando e alterando suas posições até que todos estejam em seus devidos lugares. Ele pode ser mais eficiente em listas menores e quase ordenadas, porém em listas maiores ele demora mais por conta de sua grande quantidade de movimentações e comparações.

Código

```
public class Insertion {

    public static void main(String[] args) {
        //tamanho do vetor desejado
        int tamanho = 100;
        int[] v = new int[tamanho];
        Random rand = new Random();
        long contadorDeComparacoes = 0;
        long contadorDeMovimentacoes = 0;


        for (int i = 0; i < v.length; i++) {
            v[i] = rand.nextInt(1, tamanho+1);
```

```
        }
        //ordenarDecrescente(v);
        //Arrays.sort(v);

        for (int i = 1; i < v.length; i++) {
            int aux = v[i];
            int j = i - 1;
            contadorDeComparacoes++;
            while ((j >= 0) && (v[j] > aux)) {
                contadorDeMovimentacoes++;
                contadorDeComparacoes++;
                v[j+1] = v[j];
                j--;
            }
            v[j + 1] = aux;
            contadorDeMovimentacoes++;
        }

        System.out.println("Quantidade de comparações: " + contadorDeComparacoes);
        System.out.println("Quantidade de movimentações: " +
contadorDeMovimentacoes);


    }

    public static void ordenarDecrescente(int[] v ){
        int aux;
        for (int i = 0; i < v.length; i++) {
            for (int j = 0; j < v.length - 1 - i; j++) {
                if (v[j] < v[j + 1]) {
                    aux = v[j];
                    v[j] = v[j + 1];
                    v[j + 1] = aux;

                }
            }
        }

    }
}
```

Ordem crescente

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Insertion				
	100	0 seg	99	99
	1000	0 seg	999	999
	10000	0 seg	9999	9999
	100000	0 seg	99999	99999

Ordem decrescente

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Insertion				
	100	0 seg	4999	4999
	1000	0 seg	499994	499994
	10000	0 seg	49999937	49999937
	100000	13 seg	4999999863	4999999863

Ordem aleatória

Tipo de Ordenação	Quantidade de Números	Tempo	Número de Movimentos	Número de Comparação
Insertion				
	100	0 seg	2286	2286
	1000	0 seg	253005	253005
	10000	0 seg	24996152	24996152
	100000	1 seg	2507995850	2507995850

Selection Sort

Funcionamento:
O algoritmo consiste em três passos.

- Primeiro: encontrar o menor elemento do vetor o percorrendo com o laço de for;
- Segundo: trocar o menor elemento e colocá-lo na posição correta;
- Terceiro: repetir o primeiro e o segundo passo com o próximo índice do vetor, ignorando o elemento que já foi colocado na sua posição correta.

Como o algoritmo varre todo o vetor, não importa o seu tamanho, o número de comparações feitas pelo algoritmo pode ser determinado pela função (n^2 - n) / 2, onde n = número de elementos no vetor.

Algoritmo utilizado no estudo:

```
  public static void main(String[] args) {

      long tempoInicialMili, tempoInicialNano, tempoFinalMili, tempoFinalNano,
duracaoMili, duracaoNano;

      tempoInicialMili = System.currentTimeMillis();
      tempoInicialNano = System.nanoTime();

      int[] v = new int[100];
      preencherVetorAleatorio(v);

      long countTrocas = 0;
      int i, j, aux;
      for (i = 0; i < v.length - 1; i++) {
          int menor = i;
          for (j = i + 1; j < v.length; j++) {
              if (v[menor] > v[j]) {
                  menor = j;
                  countTrocas++;
              }
          }
          aux = v[i];
          v[i] = v[menor];
          v[menor] = aux;
      }
           tempoFinalMili = System.currentTimeMillis();
            tempoFinalNano = System.nanoTime();
```

```
        duracaoMili = tempoFinalMili - tempoInicialMili;
        duracaoNano = tempoFinalNano - tempoInicialNano;

        System.out.println("Trocas realizadas: "+countTrocas);
        System.out.printf("duração Milissegundos: %d\nduração Nanossegundos: %d",
duracaoMili, duracaoNano);
    }

    public static int[] preencherVetorAleatorio(int[] v){
        Random rand = new Random();

        for (int i = 0; i < v.length; i++) {
            v[i] = rand.nextInt(0, 999999);
        }

        return v;
    }

    public static int[] preencherVetorCrescente(int[] v){

        for (int i = 0; i < v.length; i++) {
            v[i] = i;
        }

        return v;
    }

    public static int[] preencherVetorDecrescente(int[] v){
        for (int i = 0; i < v.length; i++) {
            v[i] = v.length-i;
        }

        return v;
    }
```

Resultados obtidos:

100N°:	Tempo (s, ms, ns)	Trocas	Comparações
crescente	0s / 0ms / 141000ns	0	4950
decrescente	0s / 0ms / 243200ns	2500	4950
desordenada	0s / 1ms / 541500ns	327	4950

1000N°:	Tempo (s, ms, ns)	Trocas	Comparações
crescente	0s / 4ms / 3749300ns	0	499500
decrescente	0s / 5ms / 5089700ns	250000	499500
desordenada	0s / 4ms / 5048600ns	5451	499500

10000N°:	Tempo (s, ms, ns)	Trocas	Comparações
crescente	0s / 29ms / 3019500ns	0	49995000
decrescente	0s / 84ms / 81499400ns	25000000	49995000
desordenada	0s / 86ms / 85578300ns	76852	49995000

100000N°:	Tempo (s, ms, ns)	Trocas	Comparações
crescente	2s / 2310ms / 2309830500ns	0	4999950000
decrescente	7s / 6891ms / 6891543800ns	2500000000	4999950000
desordenada	7s / 7728ms / 7727596300ns	1007123	4999950000

Conclusão:

Os algoritmos testados e estudados são comumente utilizados para a ordenação de arrays em computação. Por se tratarem de códigos com lógicas de funcionamento diferentes - suas performances, aplicações e vantagens são únicas e situacionais. Ao compará-los diretamente, a grande semelhança entre eles é que todos performam melhor em arrays menores e todos mantém elementos de chaves iguais em suas ordens originais. Porém em uma análise mais minuciosa, considerando a aplicação desses métodos em arrays maiores, o insertion sort se mostra o algoritmo mais vantajoso devido ao método com que ele compara e ordena o vetor, já o bubble sort e o selection sort não realizam a ordenação com a mesma eficiência.
Em suma, é possível concluir que considerando vetores de baixa escala, o algoritmo escolhido não fará tanta diferença na performance, podendo ser considerado como critério de escolha o algoritmo cujo a implementação e entendimento sejam mais fáceis, que é o caso do bubble sort e do selection sort. Porém em vetores maiores o insertion sort deve ser considerado como preferência.