Universidade do Estado do Amazonas

Escola Superior de Tecnologia

Insertion Sort

Manaus - AM

2019

Universidade do Estado do Amazonas

Escola Superior de Tecnologia

Juliana Fernandes Martins

Leonardo Maia de Lima

Vitor Oliveira Siqueira

Insertion Sort

Trabalho apresentado como recurso avaliativo da disciplina de Algoritmo e Estrutura de Dados II, ministrado pelo professor Sergio Cleger Tamayo.

Manaus - AM

2019

Introdução

Há vários tipos de algoritmos de ordenação, eles têm por objetivo colocar os elementos em uma determinada ordem, efetuando assim uma ordenação completa ou parcial dos elementos. Um dos algoritmos de ordenação que será visto neste trabalho é o *Insertion Sort* que significa ordenação por inserção.

É um tipo de algoritmo simples e eficiente quando se trata em ordenar um número pequeno de elementos. A lista neste algoritmo é percorrida da esquerda para a direita, conforme vai avançando deixa os elementos mais ordenados à esquerda.

Um dos exemplos do cotidiano que pode ser visto sobre esse tipo de ordenação é quando as pessoas usam para ordenar cartas em um jogo de baralho como o pôquer.

Ordenação por Inserção

Onde usar: esse método é utilizado quando o vetor está "quase" ordenado.

Como usar: Neste algoritmo, o primeiro número a ser selecionado é o segundo do vetor porque o primeiro não possui antecessor se o elemento selecionado for menor que o número anterior, então o número selecionado é trocado com o antecedente, do contrário, será realizado o mesmo procedimento com o próximo elemento.

Tipo de Ordenação (Completa ou Parcial):

O tipo de ordenação é completa, pois para ordenar um vetor qualquer o algoritmo precisa percorrer ele por completo.

Forma de Ordenação (Intercambio, Seleção, Inserção, Fusão, outra):

O método de ordenação usado pelo *insertion sort* é o intercambio ou troca que no caso efetua a comparação de elementos contidos no vetor e depois caso atendidas as condições para a forma de ordenação efetua a troca entre estes elementos.

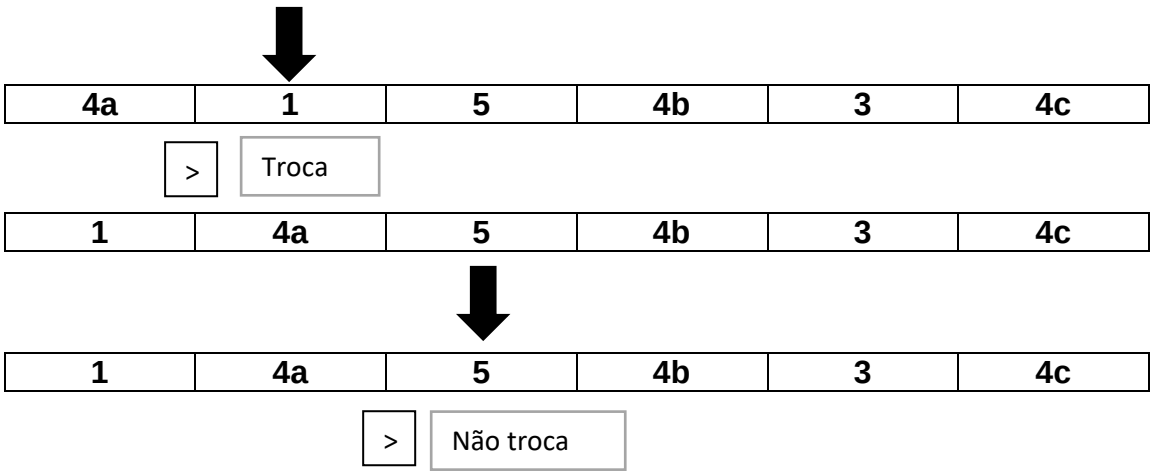
Complexidade

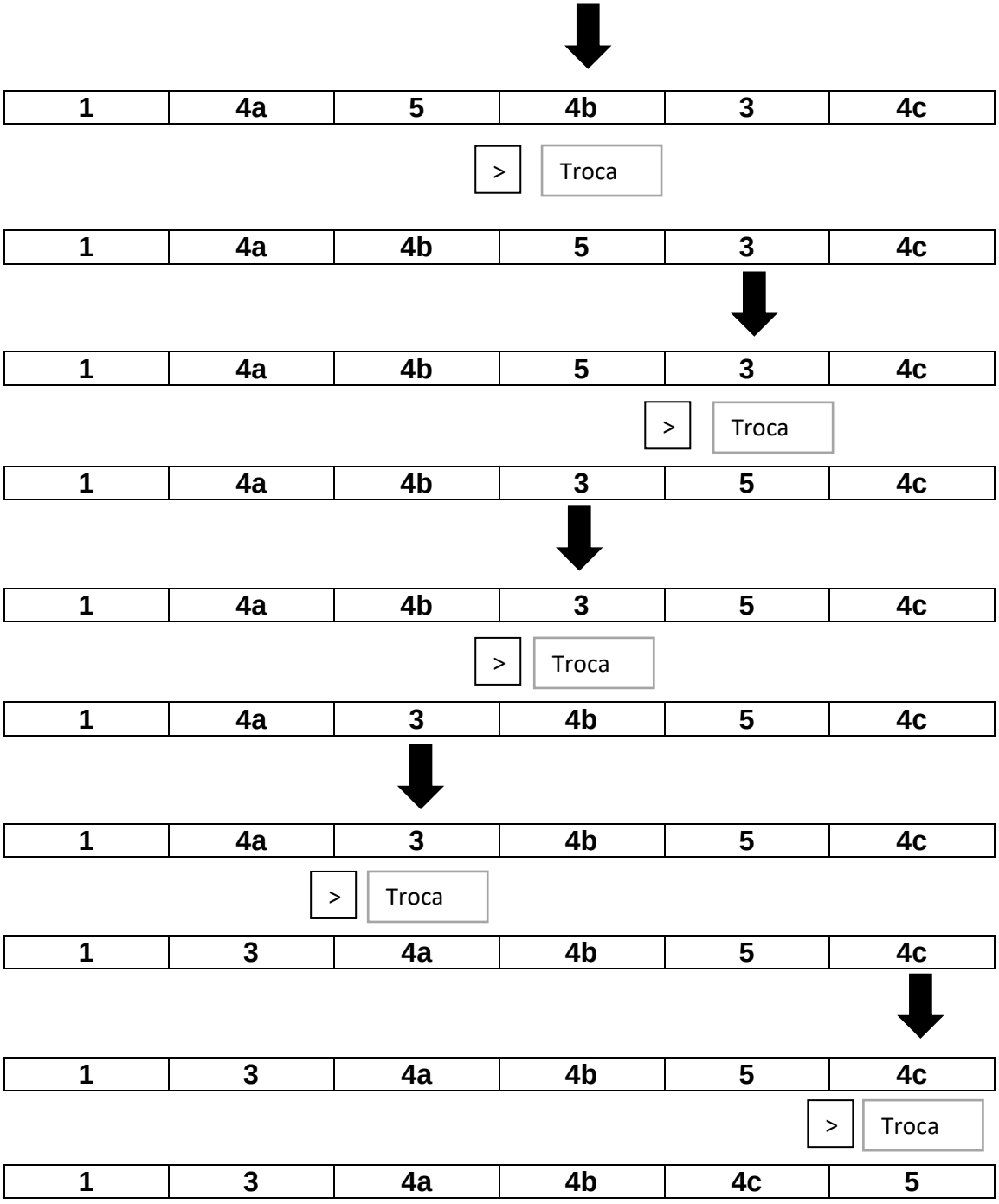
Considerando um vetor com N elementos, o tempo de execução é:

- Melhor Caso: O(N), quando os elementos já estão todos ordenados.
- Caso Médio: O(N^2), quando os elementos possui valores aleatórios sem ordem de classificação(crescente ou decrescente).
- Pior Caso: O(N^2), quando os elementos estão ordenados em ordem inversa.

Estabilidade

É considerado um algoritmo estável, pois não altera a ordem dos dados iguais.




1 4a 5 4b 3 4c
> Troca
1 4a 4b 5 3 4c
1 4a 4b 5 3 4c
> Troca
1 4a 4b 3 5 4c
1 4a 4b 3 5 4c
> Troca
1 4a 3 4b 5 4c
1 4a 3 4b 5 4c
> Troca
1 3 4a 4b 5 4c
1 3 4a 4b 5 4c
> Troca
1 3 4a 4b 4c 5

Vantagens

- Algoritmo simples e de fácil implementação.
- É considerado um dos mais rápidos algoritmos de ordenação quando se trata de um pequeno conjunto de elementos.
- É um bom método quando se quer adicionar poucos elementos em um vetor ordenado, pois o custo é linear.

Desvantagens

Possui um alto custo de movimentação de elementos no vetor.

Exemplo de aplicação real e as linguagens de programação que é utilizado como padrão

Normalmente o *Insertion sort* é utilizado em pequenos processos que se faz necessário a ordenação de poucos números, pois conforme a quantidade de chaves dentro de um vetor cresce o algoritmo leva mais tempo para ser executado por isso é bem difícil ser usado em projetos práticos, pois a preferencia vai para algoritmos que usam como método principal o dividir e conquistar.

Este algoritmo de ordenação é similar a ordenação de cartas de baralho com as mãos, em que se pega uma carta de cada vez e a insira em seu devido lugar, sempre deixando as cartas da mão em ordem.

Aqui temos a implementação do Pseudocódigo e linguagem C, mas também é possível realizar a implementação em C++, C#, Python, Java e entre outros.

Pseudocódigo:

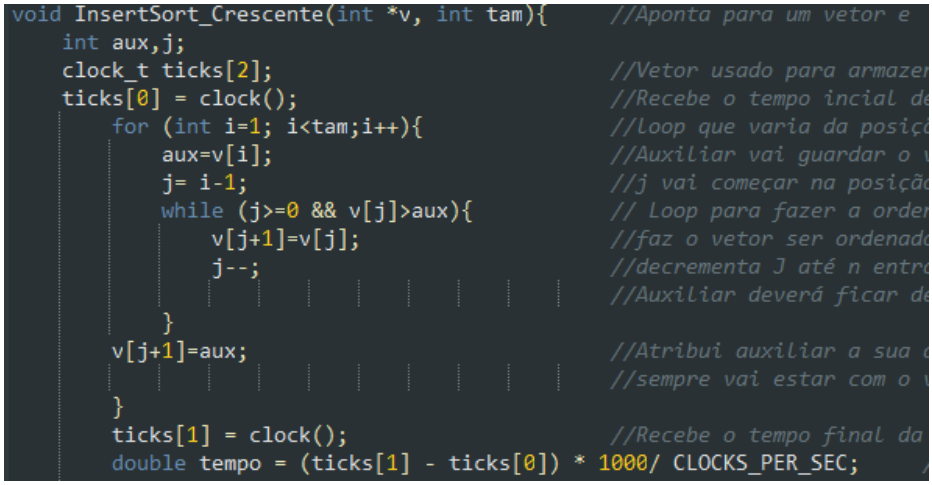
```
FUNÇÃO INSERTION_SORT (A[], tamanho)
    VARIÁVEIS
        i, j, eleito
    PARA i <- 1 ATÉ (tamanho-1) FAÇA
        eleito <- A[i];
        j <- i-1;
        ENQUANTO ((j>=0) E (eleito < A[j])) FAÇA
            A[j+1]:= A[j];
# Elemento de lista numerada
            j:=j-1;
```

```
            FIM_ENQUANTO
            A[j+1] <- eleito;
        FIM_PARA
FIM
```

Em C:

```
void insertion_sort(int vetor[], int tamanhoVetor) {

int escolhido, j, i;

    for (int i = 1; i < tamanhoVetor; i++) {
                escolhido = vetor[i];
                j = i - 1;

                while ((j >= 0) && (vetor[j] > escolhido)) {
                        vetor[j + 1] = vetor[j];
                        j--;
                }

                vetor[j + 1] = escolhido;
        }
}
```

Código comentado em C ou C++;

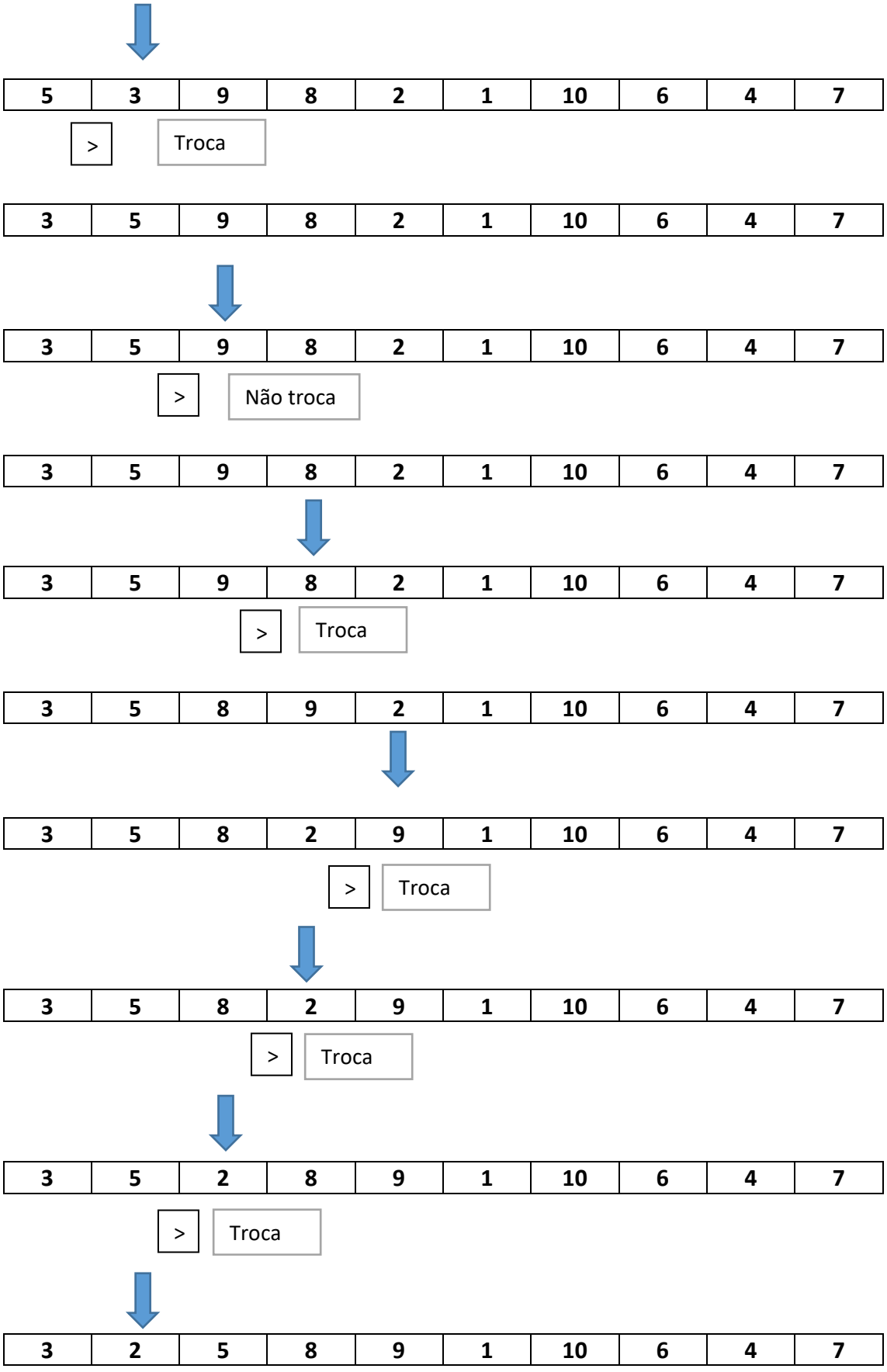


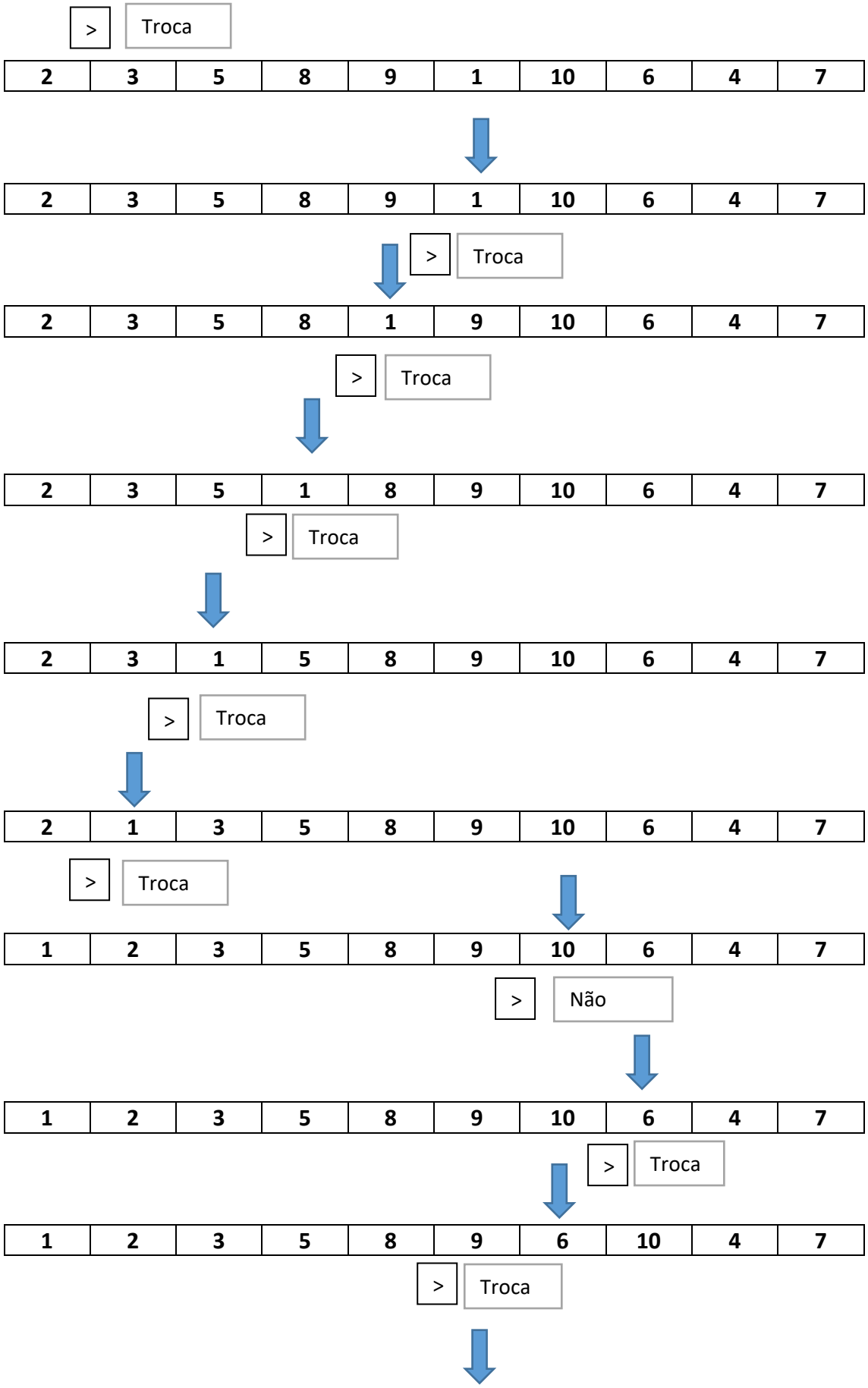
```
void InsertSort_Crescente(int *v, int tam){      //Aponta para um vetor e
    int aux,j;
    clock_t ticks[2];                            //Vetor usado para armazer
    ticks[0] = clock();                          //Recebe o tempo incial de
        for (int i=1; i<tam;i++){                //loop que varia da posiç
            aux=v[i];                            //Auxiliar vai guardar o
            j= i-1;                              //j vai começar na posiçã
            while (j>=0 && v[j]>aux){            // Loop para fazer a order
                v[j+1]=v[j];                     //faz o vetor ser ordenad
                j--;                             //decrementa J até n entr
                                                 //Auxiliar deverá ficar de
            }
        v[j+1]=aux;                              //Atribui auxiliar a sua
                                                 //sempre vai estar com o
        }
        ticks[1] = clock();                      //Recebe o tempo final da
        double tempo = (ticks[1] - ticks[0]) * 1000/ CLOCKS_PER_SEC;
```

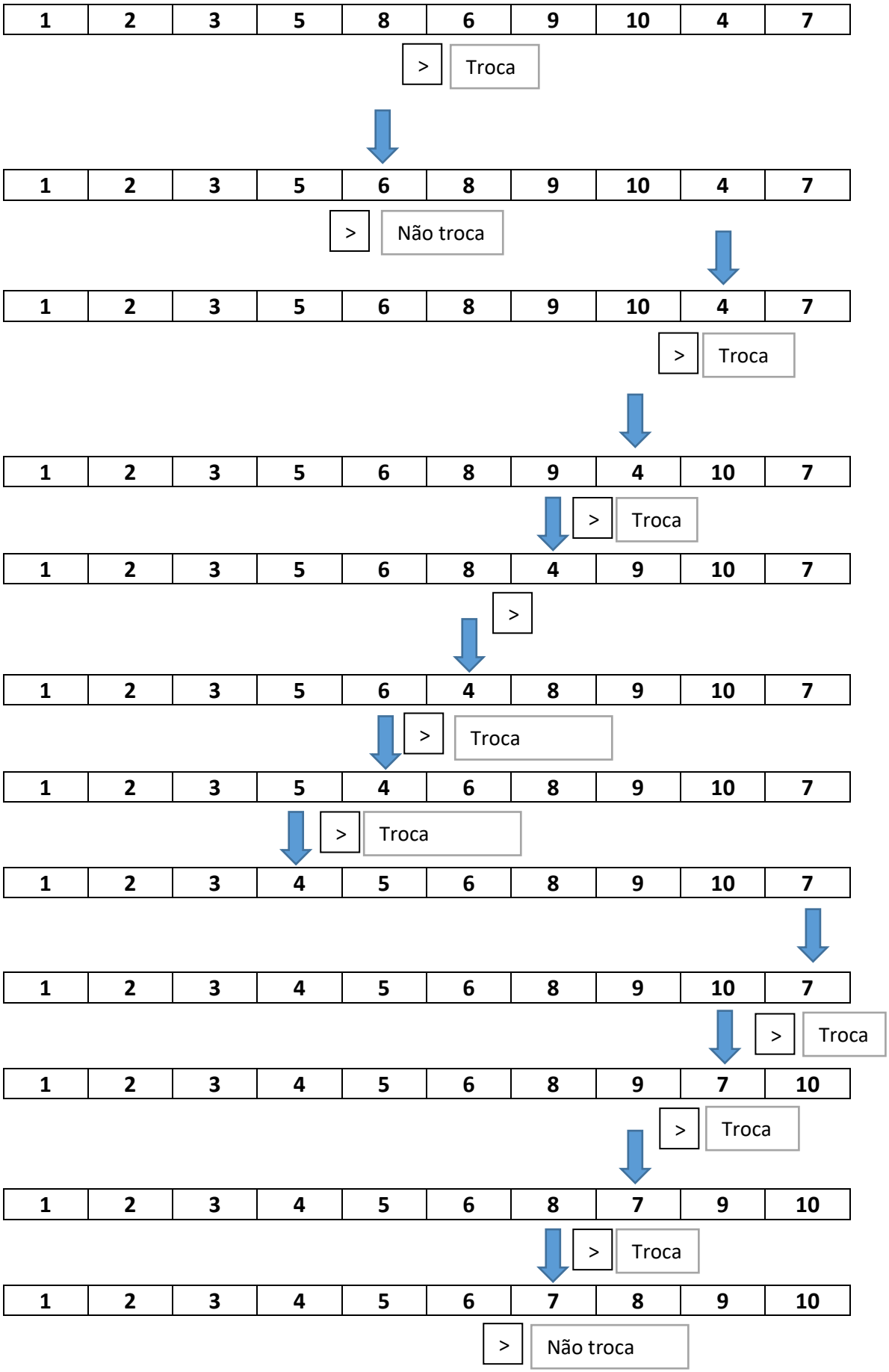
A seguir temos o código escrito:

```
Void InsertSort_Crescente(int *v, int tam){
Int aux,j;
        for (int i=1; i<tam; i++){
                aux=v[i];
                j=i-1;
                while(j>=0 && v[j]>aux){
                        v[j+1]=v[j];
                        j--;
                        }
                v[j+1]=aux;
                }
}
```

Exemplo passo a passo com vetor de 10 números inteiros mínimo




>
Troca
2 3 5 8 9 1 10 6 4 7
2 3 5 8 9 1 10 6 4 7
>
Troca
2 3 5 8 1 9 10 6 4 7
>
Troca
2 3 5 1 8 9 10 6 4 7
>
Troca
2 3 1 5 8 9 10 6 4 7
>
Troca
2 1 3 5 8 9 10 6 4 7
>
Troca
1 2 3 5 8 9 10 6 4 7
>
Não
1 2 3 5 8 9 10 6 4 7
>
Troca
1 2 3 5 8 9 6 10 4 7
>
Troca

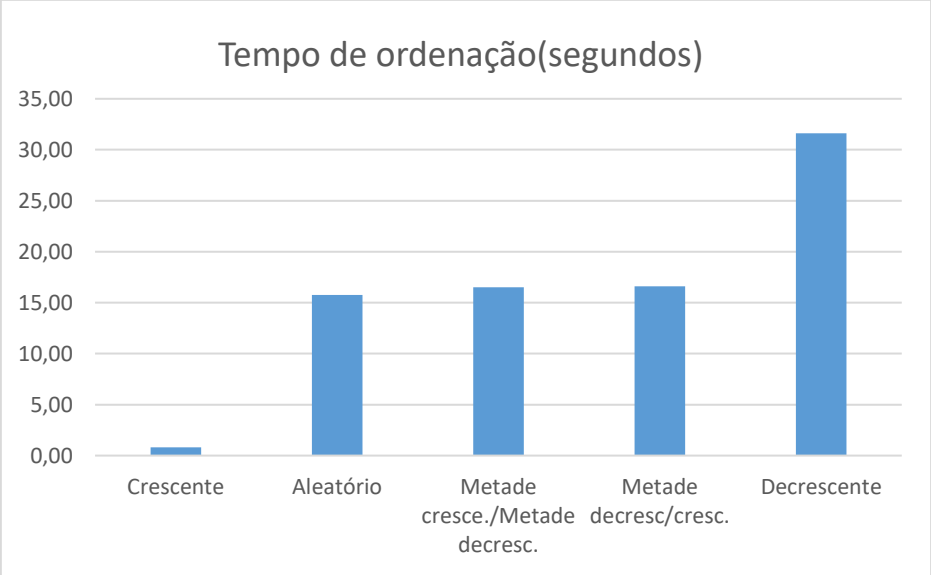

1 2 3 5 8 6 9 10 4 7
> Troca
1 2 3 5 6 8 9 10 4 7
> Não troca
1 2 3 5 6 8 9 10 4 7
> Troca
1 2 3 5 6 8 9 4 10 7
> Troca
1 2 3 5 6 8 4 9 10 7
>
1 2 3 5 6 4 8 9 10 7
> Troca
1 2 3 5 4 6 8 9 10 7
> Troca
1 2 3 4 5 6 8 9 10 7
1 2 3 4 5 6 8 9 10 7
> Troca
1 2 3 4 5 6 8 9 7 10
> Troca
1 2 3 4 5 6 8 7 9 10
> Troca
1 2 3 4 5 6 7 8 9 10
> Não troca

1	2	3	4	5	6	7	8	9	10

Codificação e avaliação com 100000 números:

Tempo(segundos)/Tipo de vetor	Tempo 1	Tempo 2	Tempo 3	Tempo 4	Tempo 5	Soma dos tempos	Média de tempo
Crescente	1,000	1,000	1,000	0,000	1,000	4,000	0,80
Aleatório	15,791	15,817	15,722	15,721	15,799	78,850	15,77
Metade cresce./Metade decresc.	15,877	16,998	15,817	17,002	16,814	82,508	16,50
Metade decresc./cresc.	16,964	16,944	15,960	16,226	16,870	82,964	16,59
Decrescente	31,565	31,498	31,607	31,481	31,977	158,128	31,63

Tabela com os tempos de cada tipo de vetor recebido pelo algoritmo insertion sort dos quais foram ordenados de forma crescente, como necessidade para a obtenção de melhor precisão nos tempos de ordenação foi feita a escolha da utilização de uma média dos tempos em que cada coluna de tempo foi obtida com um vetor diferente garantindo assim melhor precisão. A seguir está anexado um gráfico para uma melhor representação visual dos tempos de ordenação, sendo este um meio para confirmar o que foi apresentado no tópico sobre a complexidade do algoritmo e o seu melhor caso para ordenação, vetor já ordenado, o pior caso para fazer a ordenação, vetor ordenado de forma inversa, e o caso médio com o vetor aleatoriamente ordenado.



Bibliografias

PEREIRA, César. **Ordenação por inserção**. Disponível em: <https://pt.slideshare.net/cesaraugusto181/ordenao-por-insero>. Acesso 23 de agosto as 13:00.

BACKES, André. **Algoritmo de ordenação**. Disponível em: < http://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>. Acesso 22 de agosto as 10:13.

WANDY, Joe. **As vantagens e desvantagens dos algoritmos de ordenação.** Disponível em: < https://www.ehow.com.br/vantagens-desvantagens-algoritmos-ordenacao-info_16277/>. Acesso 23 de agosto as 13:05.

SILVA, Joel. **Pesquisa e Ordenação de Dados.** Disponível em: < http://joeldasilva.com.br/pod_2018_1/aula_02/Aula_02_POD_Ordenacao_de_Dados_+_Insert_Sort.pdf>. Acesso 23 de agosto as 14:50.

RIBEIRO, Cristina. **Ordenação.** Disponível em: < https://web.fe.up.pt/~arocha/AED1/APONTAMENTOS/Ordena01.pdf>. Acesso 22 de agosto as 14:07.