# Atividade B3-1 – Cálculo do tempo de execução do algoritmo Insertion Sort

**Enunciado:**

**Instruções**

Vimos que o tempo de execução de um algoritmo é dado pela quantidade de passo básicos executados por ele sobre uma certa instância de entrada.

posto isto, elabore a contagem de tempo para o seguinte algoritmo; INSERTION-SORT

```
INSERTION-SORT(A)
for j ← 2 to length[A]
     do key ← A[j]
          ▷ Insert A[j] into the sorted
                sequence A[1 . . j − 1].
          i ← j − 1
          while i > 0 and A[i] > key
              do A[i + 1] ← A[i]
                 i ← i − 1
          A[i + 1] ← key
```

FONTE; Algoritmos, teoria e prática (Cormen,2002)

Importante:
Considere cada instrução tempo um tempo ***t***:

- Atribuição de valores á variáveis
- operação de lógica
- operação aritmética
- operação de acesso
- operação de retorno

a entrega deverá no Github, conforme padrão estabelecido na disciplina

**Explicação do Código:**

Para analisar o tempo de execução do algoritmo, precisamos contar o número de operações básicas executadas em cada linha de código, considerando cada uma das instruções como uma operação que leva um tempo "t".

- **Análise Linha por Linha:**

**1. Linha 1: `for j <- 2 to length[A]`**

- A operação de atribuição (` j <- 2` ) ocorre uma vez: t.

- O teste da condição no `for` (` j <= length[A]` ) ocorre (n - 1) vezes (onde n é o tamanho do array): (n – 1) x t.

- A operação de incremento (` j++` ) ocorre (n – 1) vezes: (n-1) x t.

**Total da linha 1: t + 2(n - 1) t = (2n – 1)t.**

**2. Linha 2: `key <- A[j]`**

- A operação de atribuição (` key <- A[j]` ) ocorre (n - 1) vezes, uma para cada valor de "j" no loop: (n - 1)t.

**Total da linha 2: (n – 1)t.**

**3. Linha 3: `i <- j - 1`**

- A operação de atribuição (` i <- j - 1` ) ocorre (n - 1) vezes: (n - 1)t.

**Total da linha 3: (n – 1)t.**

**4. Linha 4: `while i > 0 and A[i] > key`**

- O número de vezes que este `while` é executado depende da posição do elemento `key` em relação aos elementos anteriores.

- Vamos supor que em um cenário específico, o `while` faz o máximo de comparações possíveis, ou seja, percorre todos os elementos anteriores a `key`. Neste caso, o número de iterações para cada valor de "j" será j – 1.

- O teste da condição ` i > 0` ocorre uma vez para cada iteração do `while`, então o número total de comparações na pior hipótese é: $\sum j=2n(j-1)=2(n-1)n$.

- Cada comparação tem 2 operações básicas (comparação ` i > 0` e ` A[i] > key` ): $2 \times 2(n - 1)n/2\ t=n(n-1)t$.

**Total da linha 4: n (n – 1)t.**

**5. Linha 5: `A[i + 1] <- A[i]`**

- Atribuição dentro do `while`, executada sempre que a condição for verdadeira.

- O número de atribuições é igual ao número de iterações do `while`, que no caso é ∑j=2n (j−1)=2(n−1)n.

**Total da linha 5: 5: (n−1)n/2 × t.**

**6. Linha 6: `i <- i - 1`**

- Atribuição dentro do `while`, também executada no pior caso: ∑j=2n(j−1)=2(n−1)n vezes.

**Total da linha 6: (n – 1)n/2 × t.**

**7. Linha 7: `A[i + 1] <- key`**

- Atribuição após o `while`, ocorre (n – 1) vezes.

**Total da linha 7: (n – 1)t.**

**Soma Total do Tempo de Execução:**

- **Somando o tempo gasto em todas as linhas:**

**- Linha 1: (2n - 1)t.**

**- Linha 2: (n – 1)t .**

**- Linha 3: (n – 1)t.**

**- Linha 4: n(n – 1)t.**

**- Linha 5: (n – 1)n/2 × t.**

**- Linha 6: (n – 1)n/2 × t.**

**- Linha 7: (n – 1)t.**

- **Somando todos os termos:**
  **T(n)=(2n−1)t+(n−1)t+(n−1)t+n(n−1)t+2(n−1)nt+2(n−1)nt+(n−1)t**

- **Simplificando:**

**T(n)=4(n−1)t+n(n−1)t+(n−1)nt=4nt−4t+n2t−nt+n2t−nt T(n)=2n2t+2nt−4tT(n) = 2n^2t + 2nt - 4t**

**T(n)=2n2t+2nt−4t**

**Conclusão:**

O tempo de execução do algoritmo Insertion Sort no pior caso é T(n) = (2n^2t + 2nt - 4t), o que corresponde à complexidade O(n^2).