# 1. Análise de Complexidade de Algoritmos

## 1.1 – Conceitos

Análise de Algoritmos é a área da computação que visa determinar a complexidade (custo) de um algoritmo, o que torna possível:

- Comparar algoritmos
- Determinar se um algoritmo é "ótimo".

Custo de um algoritmo:

- Tempo (número de passos)
- Espaço (memória)

Ex. 1: Organizar uma corda de tamanho T de maneira que ocupe o menor espaço possível.

1. Enrolar no braço
   (mesmo comprimento do laço)
2. Enrolar sobre si
   (aumentando gradativamente o tamanho do laço)
3. Dobrar sucessivamente
   (até que não seja mais possível dobrar)

Qual o método mais eficiente?

A complexidade de um algoritmo é medida segundo um **modelo matemático** que supõe que este vai trabalhar sobre uma entrada (massa de dados) de tamanho N.

O processo de execução de um algoritmo pode ser dividido em etapas elementares denominadas **passos** (número fixo de operações básicas, tempo constante, operação de maior freqüência chamada **dominante**) . O **número de passos** de um algoritmo é considerado como o número de execuções da operação dominante em função das entradas, desprezando-se constantes aditivas ou multiplicativas.

Definimos a **expressão matemática** de avaliação do tempo de execução de um algoritmo como sendo uma **função** que fornece o **número de passos** efetuados pelo algoritmo a partir de uma certa **entrada**.

Ex. 2: Soma de vetores

```
para  I  de 1 até N  faça
        S[I] ← X[I] + Y[I]
Fim para
```

Número de passos = número de somas (N)

Ex. 3: Soma de matrizes

```
Para I de 1 até N faça
        Para J de1 até N faça
                C[I,J] ←A[I,j] + B[I,J]
        Fim para
Fim para
```

Número de passos = número de somas (N*N)

Ex. 4: Produto  de matrizes

```
Para I de 1 até N faça
        Para J de 1 até N faça
                P[I,J]  ←0
                Para K de 1 até N faça
                        P[I,J] ← P[I,J] + A[I,K] * B[K,J]
                Fim para
        Fim para
Fim para
```

Número de passos = Número de somas e produtos (N*N*N)

A complexidade pode ser qualificada quanto ao seu comportamento como:

- **Polinomial**

A medida que N aumenta o fator que estiver sendo analisado (tempo ou espaço) aumenta linearmente.

- **Exponencial**

A medida que N aumenta o fator que estiver sendo analisado (tempo ou espaço) aumenta exponencialmente.

Algoritmo com complexidade exponencial, não é executável para valores de N muito grandes.

## 1.2 – Notação

A **notação O** é utilizada para expressar comparativamente o crescimento assintótico (velocidade com que tende a infinito) de duas funções.

Por definição, f = **O**(g) se existe uma constante c > 0 e um valor $n_0$ tal que

$$n > n_0 \Rightarrow f(n) \leq c * g(n)$$

ou seja, g atua como limite **superior** para valores assintóticos da função f.

**Funções elementares usadas como referência**: 1, n, lg n, $n^2$, n lg n, $2^n$ (lg indica logaritmo na base 2)

Ex.:

| f(n) | g(n) |
|---|---|
| 2N + 5 | N |
| N | N |
| 127N * N + 457 * N | N * N |
| N * N * N + 5 | N * N * N |

(f(n) = **O**(g(n))

**Propriedades**: Sejam f e g funções reais positivas e k uma constante. Então

(i) **O**(f + g) = **O**(f) + **O**(g)
(ii) **O**(k * f) = k * **O**(f) = **O**(f)

A **notação** θ é usada para exprimir limites **superiores** justos. Sejam f e g funções reais positivas da variável n.

f = θ(g) ⇔ f = **O**(g) e g = **O**(f)

A notação $\theta$ exprime o fato de que duas funções possuem a mesma ordem de grandeza assintótica.
Ex.: $f = n^2 - 1$; $g = n^2$; $h = n^3$ então
f é **O**(g); f é **O**(h); g é **O**(f) mas h não é **O**(f). Logo
f é $\theta$(g), mas f não é $\theta$(h).

A **notação** $\Omega$ é usada para exprimir limites **inferiores** assintóticos.
Sejam f e g funções reais positivas da variável n. Diz-se que $f = \Omega(g)$ se existe uma constante $c > 0$ e um valor $n_0$ tal que

$$n > n_0 \Rightarrow f(n) \geq c * g(n)$$

Ex.: $f = n^2 - 1$, então são válidas as igualdades:
$f = \Omega(n)$ e $f = \Omega(1)$, mas não vale $f = \Omega(n^3)$.

## 1.3 Pior caso, melhor caso, caso médio

Seja A um algoritmo, $\{E_1, ..., E_m\}$ o conjunto de todas as entradas possíveis de A. Seja $t_i$ o número de passos efetuados por A, quando a entrada for $E_i$. Definem-se:

complexidade do **pior caso** = $\max E_i \in E\{t_i\}$
complexidade do **melhor caso** = $\min E_i \in E\{t_i\}$
complexidade do **caso médio** = $\sum_{1 \leq i \leq m} p_i * t_i$

onde $p_i$ é a probabilidade de ocorrência da entrada $E_i$.

## 1.4 Algoritmos ótimos

Seja P um problema. Um limite inferior par P é uma função l, tal que a complexidade de pior caso de qualquer algoritmo que resolva P é $\Omega$(l). Isto é, todo algoritmo que resolve P efetua, pelo menos, $\Omega$(l) passos. Se existir um algoritmo A, cuja complexidade seja O(l), então A é denominado algoritmo **ótimo** para P. Ou seja, A apresenta a menor complexidade dentre todos os algoritmos que resolvem P.