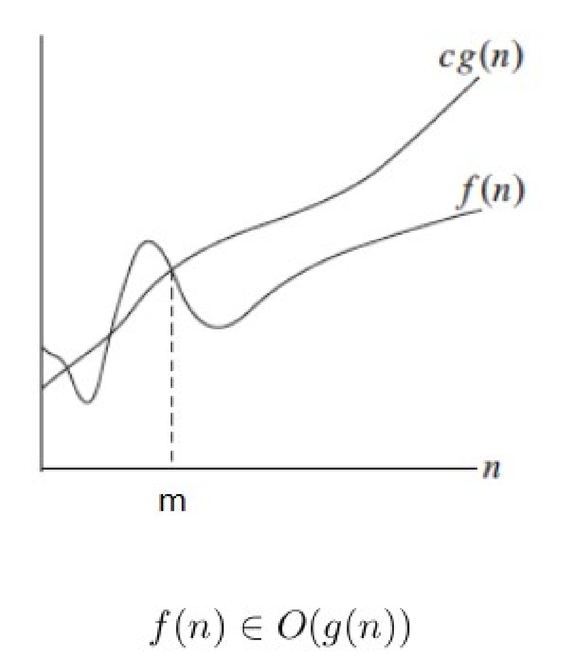
GSI011 – Estrutura de Dados 2

Profª: Christiane Regina Soares Brasil

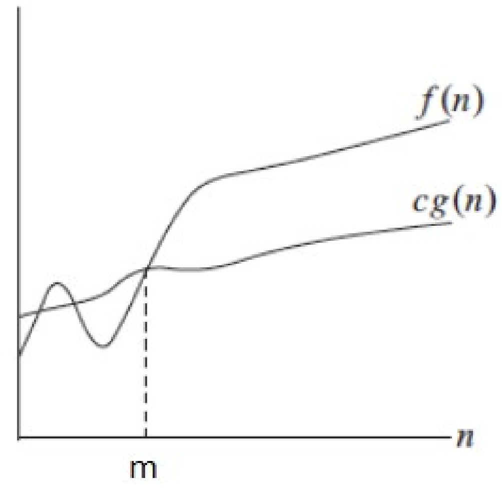
- Notação grande-O
  - Matematicamente:
  - f(n) está em O(g(n)) se existem **c** e **m** (ambos > 0) tais que f(n) <= c g(n) para n>=m (limite superior).

f(n) e g(n) são não negativas



- Notação grande-Ômega
  - Matematicamente:
  - f(n) está em Ω(g(n)) se existem **c** e **m** (ambos > 0) tais que f(n) >= c g(n) para n>=m (limite inferior).

f(n) e g(n) são não negativas



$f(n) \in \Omega(g(n))$

- Notação grande-Theta, ϴ
  - Análise do limite assintótico firme (restrito), ou seja, o limite **inferior** e **superior** do algoritmo.

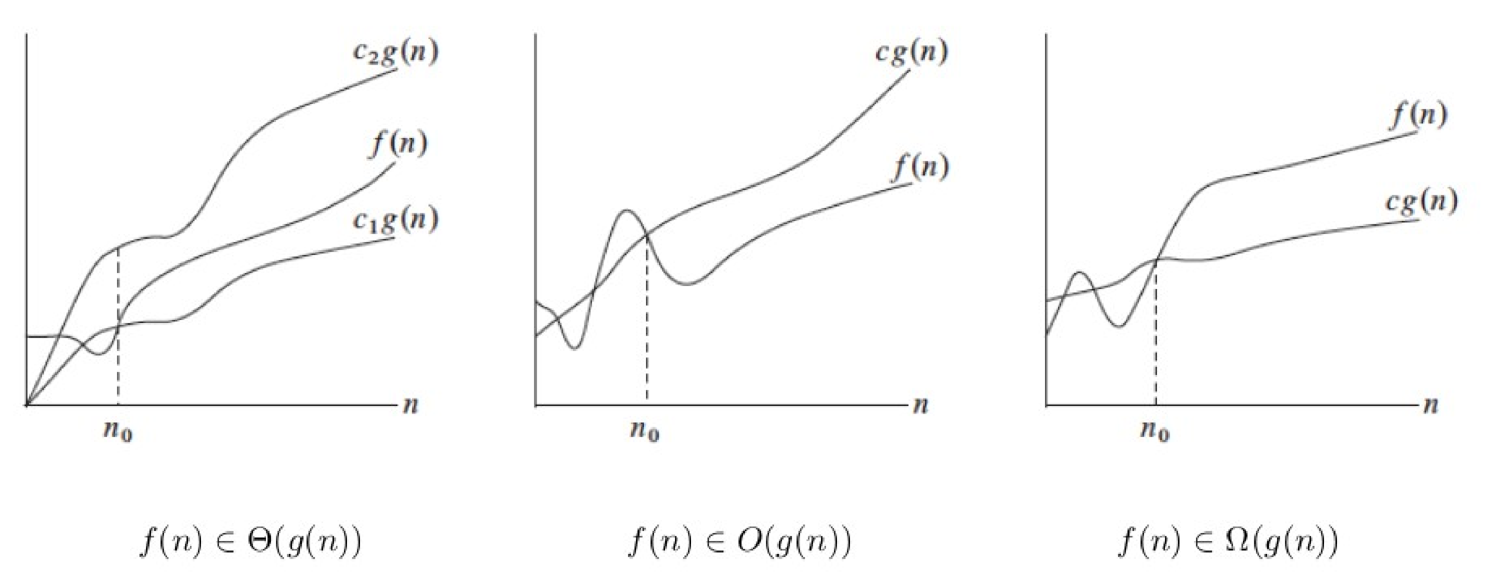
- Notação grande-Theta, ϴ
  - Matematicamente:
  - f(n) está em ϴ(g(n)) se f(n) está em O(g(n)) e também em Ω(g(n)), ou seja, se existem **c1, c2** e **m** (todos > 0) tais que:

    c1 g(n) <= f(n) <= c2 g(n), para n>=m.

f(n) e g(n) são não negativas

- Notação grande-Theta, Θ

f(n) e g(n) são não negativas



- Exemplo 1: Prove que $(1/2)n^2 - 3n$ está em $\Theta(n^2)$.

$c1\ n^2 <= (1/2)n^2 - 3n <= c2\ n^2$

Lado esquerdo:

$c1\ n^2 <= (1/2)n^2 - 3n$

$c1 <= ½ – 3/n$

$(½ – c1) >= 3/n$

Considerando c1 = 1/4, temos:
1/4 >= 3/n
n >= 12

Logo, quando c1 = 1/4 e n>=12 temos o lado esquerdo válido.

- Exemplo 1:

$c1\ n^2 <= (1/2)n^2 - 3n <= c2\ n^2$

Lado direito:

$(1/2)n^2 - 3n <= c2\ n^2$

$½ – 3/n <= c2$

Considerando c2 = 1/2, temos:
$½ – 3/n <= 1/2$
$-3/n <= 0$

Logo, quando c2 = 1/2 e n>=1 temos o lado direito válido.

Portanto, vale para c1=1/4, c2=1/2 e n>=12.

- Exemplo 2: Prove que $n(n+1)/2$ está em $\Theta(n^2)$.

$c1\ n^2 <= n(n+1)/2 <= c2\ n^2$

Lado esquerdo:

$c1\ n^2 <= n(n+1)/2$

$c1 <= (n + 1)/2n$

Considerando c1 = 1/2, temos:
$1/2 <= (n+1)/2n$
$½ <= ½\ (n+1)/n$
$1 <= (n + 1)/n$
$n <= (n+1)$
$0 <= 1$

Logo, quando c1 = 1/2 e para qualquer n>0 temos o lado esquerdo válido.

Exemplo 2: Prove que $n(n+1)/2$ está em $\Theta(n^2)$.

$c1\ n^2 <= n(n+1)/2 <= c2\ n^2$

Lado direito:

$n(n+1)/2 <= c2\ n^2$

$(n+1)/2n <= c2$

Considerando $c2 = 1$, temos:

$(n+1)/2n <= 1$

$n+1 <= 2n$

$n >= 1$

Logo, quando $c2 = 1$ e $n>=1$ temos o lado direito válido.

Portanto, vale para $c1=1/2$, $c2=1$ e $n>=1$.

- Exemplo 3: Prove que $6n^3$ **NÃO** está em $\Theta(n^2)$.

$c1\ n^2 <= 6n^3 <= c2\ n^2$

Lado esquerdo:

$c1\ n^2 <= 6n^3$

$c1 <= 6n$

Considerando c1 = 6, temos:
6 <= 6n
1 <= n, ou seja, n>=1

Logo, quando c1 = 6 e n>=1 temos o lado esquerdo válido.

Exemplo 3: Prove que $6n^3$ NÃO está em $\Theta(n^2)$.

$c1\ n^2 <= 6n^3 <= c2\ n^2$

Lado direito:

$6n^3 <= c2\ n^2$

$6n <= c2$

$n <= (c2)/6$

Veja que n é limitado por (c2)/6 para qualquer c2 positivo, onde temos uma contradição, uma vez que n teria que valer para todo n suficientemente grande, ou seja, não Conseguimos encontrar um c2 que satisfaça.

Portanto, $6n^3$ NÃO está em $\Theta(n^2)$.

- Solução de recorrências
  - Para analisar a função de custo de um algoritmo recursivo é necessário trabalhar com uma relação de recorrência, ou seja, uma expressão que fornece seu valor baseado em resultados “anteriores” da mesma função.
  - Por exemplo,
    - F(n) = F(n-1) + n, onde F(1) = 1.

n	1	2	3	4	5	6
F(n)	1	3	6	10	15	21

Caso base

- Exemplo 1: Considere a recorrência F(n) = F(n-1) + 3.
  - Suponha n pertencente ao conjunto {2,3,4,5,…}, onde n é o tamanho da instância, considerando o caso base F(1) = 1.
  - Deste modo, temos:

$$F(n) = F(n - 1) + 3$$
$$F(n) = (F(n - 2) + 3) + 3$$
$$F(n) = ((F(n - 3) + 3) + 3) + 3$$
$$\ldots$$

- Exemplo 1:
  - Podemos resumir essa expansão para:

$F(n) = F(n - k) + 3k$

  - Quero escrever tudo em função de n. Para tal, usamos o caso base com $F(1) = 1$. Deste modo, temos:

$n – k = 1$

$k = n - 1$

- Exemplo 1:
  - Logo, temos:

$$F(n) = F(n - (n-1)) + 3(n-1)$$
$$F(n) = F(n - n + 1) + 3n - 3$$
$$F(n) = F(1) + 3n - 3$$
$$F(n) = O(1) + 3n - 3$$
$$F(n) = 3n - 3 + O(1)$$

A saber, $F(1) = O(1)$

Portanto, assintoticamente temos um algoritmo com complexidade linear.

- Exemplo 2: Considere a recorrência F(n) = F(n/2) + 3.
  - Vamos assumir que n pertence ao conjunto $\{2^1, 2^2, 2^3, 2^4,\ldots\}$, ou seja, das potências inteiras de 2. Isto porque não faz sentido n estar no conjunto {2, 3, 4, 5,…}, pois n não pode pertencer a esse conjunto quando n é ímpar. Também não faz sentido n estar em {2, 4, 6, 8,10,…} pois pode haver n/2 sendo ímpar.
  - Deste modo, temos:

$$F(2^k) = F(2^k/2) + 3$$

$$F(2^k) = F(2^{k-1}) + 3$$

$$2^k/2 = 2^k.2^{-1} = 2^{k-1}$$

- Exemplo 2:
  - Expandindo a recorrência, temos:

$F(2^k) = (F(2^{k-2}) + 3) + 3$

$F(2^k) = ((F(2^{k-3}) + 3) + 3) + 3$

…

$F(2^k) = F(2^{k-k}) + 3k$

$F(2^k) = F(2^0) + 3k$

$F(2^k) = F(1) + 3k$

$F(2^k) = O(1) + 3k$

$F(n) = O(1) + 3\log n$

Portanto, assintoticamente temos um algoritmo com complexidade logarítmica.

$n = 2^k$

$\log n = k$