Revisão: O que é uma **função matemática** ?

Sendo A e B dois conjuntos não vazios em uma relação f de A em B, essa relação f é uma função de A em B quando a cada elemento de x do conjunto A está associado um e um só elemento y do conjunto B.

Crescimento assintótico de funções

Chamamos de comportamento assintótico o comportamento a ser observado em uma função f(n), quando n tende ao infinito.

Em geral, o custo aumenta com o tamanho n do problema.
Para valores pequenos de n, mesmo um algoritmo ineficiente não custa muito para ser executado, por isso interessa-nos algoritmos de n entradas com n>>1.

Há uma notação especial que permite comparar classes de funções quanto ao crescimento assintótico.

Essa notação é extremamente útil porque permitirá comparar algoritmos diferentes para um mesmo problema e dá indícios de como um novo algoritmo se comporta em relação a problemas conhecidos (basicamente problemas de busca e ordenação).

Obs: Se f é uma função de complexidade para um algoritmo F, então 0(f) é considerado a complexidade assintótica do algoritmo F.

Notação Ω (ômega)

$g(n) = \Omega\ f(n)$: g(n) é de ordem, no mínimo, f(n)
f(n) é um limite assintótico inferior
g(n) cresce, pelo menos, tão rápido quanfo f

Ω define um limite assintótico inferior para g(n)

Se $g(n) \in \Omega\ f(n)$ então $g(n) = \Omega\ f(n)$

$g(n) \in \Omega\ f(n)$ então existe $c \geq 0$ e n' tal que $0 \leq cf(n) \leq g(n)$, para todo $n \geq n'$.

A notação Ω é usada para expressar o limite inferior do tempo de execução de qualquer algoritmo para resolver um dado problema.

Exemplos: (1) $n^2 \in \Omega\ (n)$ (2) $n \in \Omega\ (\log n)$

(3) Se $f(n) = 7n^3 + 5$ e $g(n) = 2^n$, então $g(n) \in \Omega\ f(n)$

Observação: o limite inferior para qualquer algoritmo de ordenação que utilize comparações entre elementos é $\Omega\ (n \log n)$

Notação 0

$g(n) = 0\ f(n)$: g(n) é de ordem, no máximo, f(n)
g(n) cresce, no máximo, tão rápido quanfo f
f(n) domina assintoticamente g(n)

Se $g(n) \in 0\ f(n)$ então $g(n) = 0\ f(n)$

$g(n) \in 0\ f(n)$ então existe $c \geq 0$ e n' tal que $g(n) \leq c\ f(n)$,
para todo $n \geq n'$.

A notação 0 é usada para expressar o limite superior do tempo de execução de um algoritmo para resolver um dado problema.

Exemplos: (1) $n \in 0\ (n^2)$ (2) $\log n \in 0\ (n)$
(3) $3n^3 \in 0\ (n^3)$

Obs 1: Se $f(n)= 0(1)$ o algoritmo independe do tamanho de n, as instruções são executadas um número fixo de vezes.

Obs 2: a notação 0 usamos para nos referirmos a algoritmos enquanto a notação Ω usamos para nos referirmos a problemas.

Notação θ (teta)

$g(n) = \theta f(n)$: g(n) é da mesma ordem que f(n)

$g(n) \in \theta f(n)$ então existe $c_1 \geq 0$ e , $c_2 \geq 0$ n' tal que
$0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$, para todo $n \geq n'$

A notação 0 é usada para expressar funções que crescem com a mesma rapidez para resolver um dado problema.

Exemplos: (1) Verifica-se facilmente que $2n^2 - 3n = \theta(n^2)$

Observação: a notação **0** é usada para algoritmos enquanto a notação Ω usamos para problemas e a notação θ depende de ambos.

Observações:

Sejam dois algoritmos F e G. Se o algoritmo de F leva 3 vezes mais tempo que G para ser executado:

- $f(n) = 3\ g(n)$, sendo que $f(n) = 0\ g(n)$ e ao mesmo tempo $g(n) = 0\ f(n)$, ou ainda podemos escrever: $0\ f(n) = 0\ g(n)$, ambos tem complexidade equivalente.

- Se $f(n)= 0(1)$ o algoritmo independe do tamanho de n, as instruções são executadas um número fixo de vezes.

- $f(n) = 0\ (\log n)$ ocorre tipicamente em algoritmo que resolvem um problema transformando-o em problemas menores.

- $f(n) = 0\ (n \log n)$ ocorre tipicamente em algoritmos que resolvem um problema quebrando-o em problemas menores, resolvendo cada um deles independentemente e depois juntando as soluções.

- $f(n) = 0(n^2)$ ocorre tipicamente quando os itens de dados são processados aos pares, em um, loop dentro de outro.

- $f(n) = 0\ (2^n)$ são os algoritmos do tipo “força-bruta”, nada práticos para processar ordenação.