Universidade Federal de Goiás – Instituto de Informática

Análise e Projeto de Algoritmos

Crescimento de Funções

Diogo Stelle
com slides do Prof. Fábio H. V. Martinez
diogostelle@inf.ufg.br / diogo.stelle@gmail.com

2019

Conteúdo da Aula

- Introdução
- Notação assintótica
- Notação e funções comuns
- Classes de Comportamento Assintótico
- Exercícios

Crescimento de funções

Ordem de Crescimento

- a ordem ou taxa de crescimento do tempo de execução de um algoritmo, como vimos na aula passada, fornece uma caracterização simples da eficiência de um algoritmo
- também permite comparar o desempenho relativo de algoritmos alternativos
- o esforço para computar o tempo de execução exato de um algoritmo não compensa
- para entradas grandes o suficiente, as constantes multiplicativas e os termos de ordem menor de um tempo de execução exato são dominados pelos efeitos do tamanho da entrada

Comportamento Assintótico de Funções

- Na aula passada aprendemos como calcular a função de complexidade f(n)
- Algumas observações:
    - Para valores pequenos de *n*, praticamente qualquer algoritmo custa pouco para ser executado.
    - Logo: a escolha do algoritmo tem pouquíssima influência em problemas de tamanho pequeno.

Comportamento Assintótico de Funções

- A análise de algoritmos deve ser realizada para valores grandes de *n*
- Para isso, estuda-se o **comportamento assintótico** das funções de custo.
    - Comportamento de suas funções para valores grandes de *n*
- O comportamento assintótico de f(n) representa o limite do comportamento do custo quando n cresce

Comportamento Assintótico de Funções

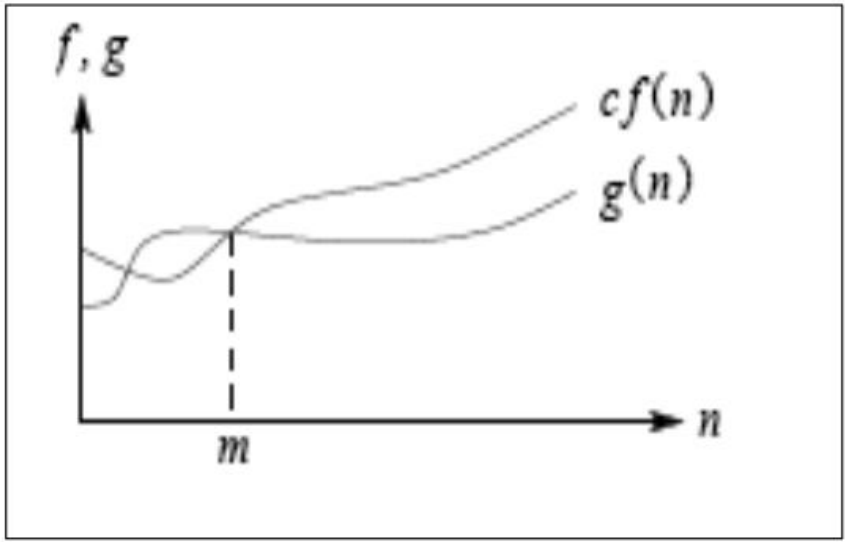
- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
- A medida de custo ou medida de complexidade relata o **crescimento assintótico** da operação considerada.

Definição: Uma função f(n) domina assintóticamente outra função g(n) se:

Existem duas constantes positivas **c** e **m** tais que, para **n ≥ m**, temos **|g(n)| ≤ c |f(n)|**

Dominação Assintótica

- f(n) domina assintoticamente (n) se:
  - Existem duas constantes positivas **c** e **m** tais que, para **n ≥ m**, temos |g(n)| ≤ c |f(n)|



Dominação Assintótica

- **Exemplo**
  - Sejam $g(n) = (n + 1)^2$ e $f(n) = n^2$.
  - As funções g(n) e f(n) dominam assintoticamente uma a outra, desde que
    - $|(n + 1)^2| \leq 4\ |n^2|$ para $n \geq 1$ e
    - $|n^2| \leq |(n + 1)^2|$ para $n \geq 0$.

Notação Assintótica

- Tempo de Execução
  - notação assintótica é aplicada a funções
  - funções que representam tempos de execução de algoritmos
  - tempo de execução: de pior caso, de melhor caso, do caso médio?
  - às vezes, mencionamos simplesmente “tempo de execução” e isso quer dizer que queremos saber o tempo de execução de um algoritmo, não importando qual entrada

Notação Assintótica

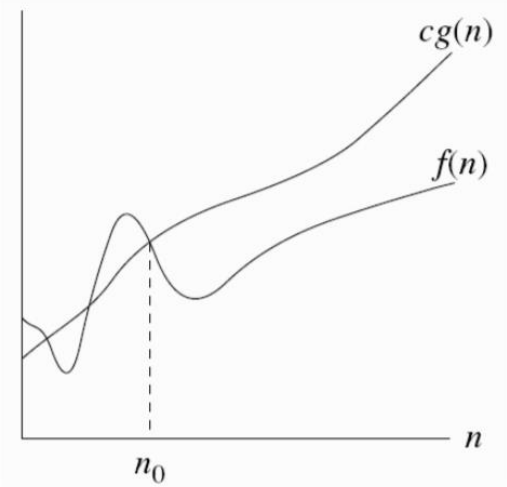
- Vamos expressar complexidade através de funções em variáveis que descrevam o tamanho de instâncias do problema. Exemplos:
  - Problemas de aritmética: número de bits (ou bytes) dos inteiros.
  - Problemas em grafos: número de vértices e/ou arestas
  - Problemas de ordenação de vetores: tamanho do vetor.
  - em textos: número de caracteres do texto ou padrão de busca.
- Vamos supor que funções que expressam complexidade são sempre positivas, já que estamos contando o número de operações.

Notação O

Notação O

- Para uma dada função g(n), denotamos por O(g(n)) o conjunto de funções

O(g(n)) = {**f(n)**: existem constantes positivas **c** e $\mathbf{n_0}$ tais que $\mathbf{0 \leq f(n) \leq cg(n)}$ para todo $\mathbf{n \geq n_0}$}.



Notação O

- Exemplo 1: $2n + 10$ é $O(n)$
  - podemos realizar uma manipulação para encontrar c e n_0:

$2n + 10 \leq c \cdot n$ **->>** $c \cdot n - 2n \geq 10$ **->>** $(c - 2)n \geq 10$ **->>** $n \geq 10 / (c-2)$

  - a afirmação é válida para $c = 3$ e $n_0 = 10$.
- Exemplo 2: n^2 é $O(n)$
  - é preciso encontrar *c* que seja sempre maior ou igual a *n* para todo valor de um n_0:

$$n^2 \leq c \cdot n \Rightarrow n \leq c$$

  - é impossível pois c deve ser constante.

Notação O

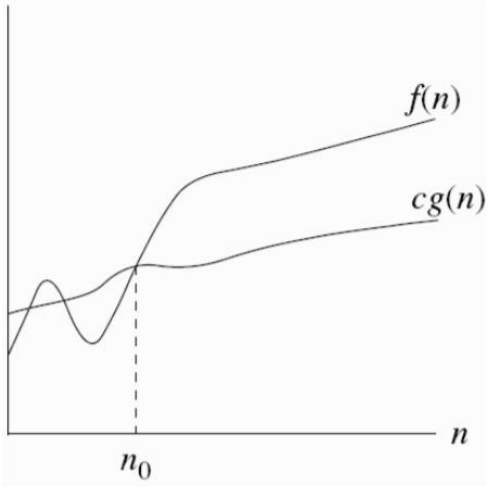
- Exemplo 3: 2^{n+2} é $O(2^n)$
  - é preciso $c > 0$ e $n_0 \geq 1$ tais que $2^{n+2} \leq c \cdot 2^n$ para todo $n \geq n_0$
  - note que $2^{n+2} = 2^2 \cdot 2^n = 4 \cdot 2^n$
  - assim, basta tomar, por exemplo, $c = 4$ e qualquer $n_0 \geq 1$
- Exemplo 4: $10n^3 - 3n^2 + 27$ é $O(n^3)$
  - $10n^3 - 3n^2 + 27 \leq c \cdot n^3$
  - $10n^3 - 3n^2 + 27 \leq 10 \cdot n^3$ se $(3n^2 + 27) \geq 0$ ou seja
  - $10n^3 - 3n^2 + 27 \leq 10 \cdot n^3$ para todo $n \geq 3$ ->> $c = 10$ e $n_0 = 10$

Notação Ω

Notação Ω

- Para uma dada função g(n), denotamos por Ω(g(n)) o conjunto de funções

Ω(g(n)) = {**f(n)**: existem constantes positivas **c** e $\mathbf{n_0}$ tais que $\mathbf{0 \leq cg(n) \leq f(n)}$ para todo $\mathbf{n > n_0}$}.



Notação Θ

- se **f(n)** ∈ **Θ(g(n))**, dizemos que **g(n)** é um limitante assintoticamente justo para **f(n)**
- além disso, toda **f(n)** ∈ **Θ(g(n))** é assintoticamente não negativa
- consequentemente, a função **g(n)** é assintoticamente não negativa

Notação Ω

- quando dizemos que o tempo de execução de um algoritmo é Ω(g(n)) queremos dizer que não importa qual entrada particular de tamanho n é escolhida, para cada valor de n, o tempo de execução para esta entrada é pelo menos uma constante vezes g(n), para n suficientemente grande
- equivalentemente, estamos fornecendo um limitante inferior sobre o tempo de execução de melhor caso de um algoritmo
- por exemplo, o tempo de execução do melhor caso da ordenação por inserção é Ω(n), o que implica que o tempo de execução da ordenação por inserção é Ω(n)

Notação Ω

- Exemplo 1: $3n^2 + n = \Omega(n)$
  - temos que encontrar constantes c e n tais que

$$3n^2 + n \geq cn$$

  - dividindo por n^2, temos

$$3 + (1/n) \geq c/n$$

  - considerando $c = 4$ e $n > 0$, temos que $3n^2 + n$ é $\Omega(n)$

Notação Ω

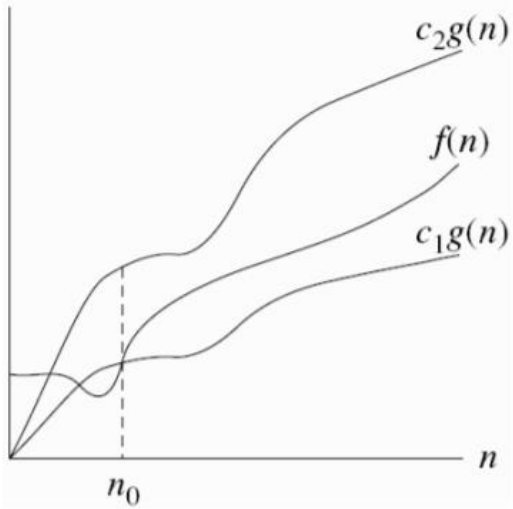
- Exemplo 2: $n^2 + 3n = \Omega(n^2)$

Notação Θ

Notação Θ

- Para uma dada função **g(n)**, denotamos por **Θ(g(n))** o conjunto de funções

  Θ(g(n)) = {**f(n)**: existem constantes positivas $\mathbf{c_1}$, $\mathbf{c_2}$ e $\mathbf{n_0}$ tais que $\mathbf{0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)}$ para todo $\mathbf{n \geq n_0}$}.



Notação Θ

- se f(n) ∈ Θ(g(n)), dizemos que g(n) é um limitante assintoticamente justo para f(n)
- além disso, toda f(n) ∈ Θ(g(n)) é assintoticamente não negativa
- consequentemente, a função g(n) é assintoticamente não negativa

Notação Θ

- **Exemplo 1** **$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0$**
  - para mostrar formalmente que, $n^2 + 3n = \Theta(n^2)$
  - definiremos constantes positivas c_1, c_2 e n_0 tais que:

$$c_1 n^2 \leq n^2 + 3n \leq c_2 n^2 ,$$

para todo $n \geq n_0$. Dividindo por n^2:

$$c_1 \leq 1 + 3/n \leq c_2,$$

  - para satisfazer a 1ª inequação, podemos quase automaticamente perceber que para qualquer $n \geq 1$, é válido $c_1 = 1$
  - para satisfazer a 2ª inequação, podemos perceber facilmente que para qualquer $n \geq 1$, é válido $c_2 = 4$

Notação Θ

- **Exemplo 2** **$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0$**
  - para mostrar formalmente que, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
  - definiremos constantes positivas c_1, c_2 e n_0 tais que:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 ,$$

para todo $n \geq n_0$. Dividindo por n^2:

$$c_1 \leq 1/2 - 3/n \leq c_2,$$

  - a desigualdade do lado direito pode ser considerada válida para $n \geq 1$ escolhendo $c_2 \geq 1/2$, e a do lado esquerdo pode ser considerada válida para $n \geq 7$ escolhendo $c_1 \geq 1/14$.
  - para $c_2 = 1/2$, $n = 7$ e $c_1 = 1/14$, temos: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Notação Θ

- **Exemplo 3** **$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0$**
  - para mostrar formalmente que, $10n^2 + 5n + 3$ é (n^2)
  - para mostrar que $f(n) = (n^2)$, vamos mostrar que $f(n) = O(n^2)$ e $f(n) = \Omega(n^2)$

Relação entre notações assintóticas

Teorema

Para quaisquer duas funções f(n) e g(n), temos $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Notação assintótica em equações e inequações

- $n = O(n^2)$ significa $n \in O(n^2)$
- $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ significa que $2n^2 + 3n + 1 = 2n^2 + f(n)$ e que $f(n) \in \Theta(n)$
- $2n^2 + \Theta(n) = \Theta(n^2)$ significa que não importa como as funções anônimas são escolhidas à esquerda da igualdade, existe uma forma de escolher as funções à direita da igualdade que tornam a equação verdadeira
- para qualquer função $f(n) \in \Theta(n)$, existe alguma função $g(n) \in \Theta(n^2)$ tal que $2n^2 + f(n) = g(n)$ para todo n

Notação o

Notação o

- Para uma dada função **g(n)**, denotamos por **o(g(n))** o conjunto de funções

  o(g(n)) = {**f(n)**: para qualquer constante **c > 0**, existe uma constante positiva $\mathbf{n_0}$ tal que **0 ≤ f(n) < cg(n)** para todo $\mathbf{n \geq n_0}$}.

Notação o

- $2n^2 = O(n^2)$ é um limitante assintoticamente justo, mas $2n = O(n^2)$ não é
- por outro lado, $2n = o(n^2)$, mas $2n^2 \neq o(n^2)$
- se $f(n) = O(g(n))$ então $0 \leq f(n) \leq cg(n)$ vale para alguma constante $c > 0$
- por outro lado, se $f(n) = o(g(n))$ então $0 \leq f(n) < cg(n)$ vale para todas as constantes $c > 0$
- intuitivamente, na notação o, a função f(n) torna-se insignificante comparada à g(n) quando n vai para o infinito; isto é

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Notação ω

Notação ω

- Para uma dada função **g(n)**, denotamos por **ω(g(n))** o conjunto de funções

  ω(g(n)) = {**f(n)**: para qualquer constante **c > 0**, existe uma constante positiva $\mathbf{n_0}$ tal que **0 ≤ cg(n) < f(n)** para todo **n ≥** $\mathbf{n_0}$}.

Notação ω

- Exemplo: $2n^2$ é $o(n^3)$
  - n^2 é sempre menor que n^3 para um n suficientemente grande, é preciso apenas determinar n_0 em função de c

$$2n^2 < c \cdot n^3$$

$$2n^2 / n^2 < (c \cdot n^3) / n^2$$

$$2n < c \cdot n$$

$$2 / n < c$$

  - a desigualdade vale para $n_0 \geq 2$ e $c = 2$

Notação ω

- f(n) ∈ ω(g(n)) se e somente se g(n) = o(f(n))
- $n^2/2$ = ω(n), mas $n^2/2$, ω(n^2)
- intuitivamente, na notação ω, a função f(n) torna-se arbitrariamente grande comparada à g(n) quando n vai para o infinito; isto é

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Comparação de funções

Comparação de funções

- Vamos comparar funções assintoticamente, ou seja, para valores grandes, desprezando constantes multiplicativas e termos de menor ordem.
- Por que queremos fazer isto?

$$n^2 , 2n^2 , 30n^2 , 100n^2 , \mathbf{c}n^2 .$$

- Obviamente, quanto maior a constante **c** associada, maior é o valor da função. Entretanto, todas elas tem a mesma velocidade de crescimento. Podemos ignorar o valor de **c**.

Comparação de funções

- Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

- Como se vê, **$3n^2$** é o termo dominante quando **n** é grande
- De um modo geral, podemos nos concentrar nos **termos dominantes** e esquecer os demais

Comparação de funções

- Transitividade
  - $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$ implica que $f(n) = \Theta(h(n))$
  - $f(n) = O(g(n))$ e $g(n) = O(h(n))$ implica que $f(n) = O(h(n))$
  - $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$ implica que $f(n) = \Omega(h(n))$
  - $f(n) = o(g(n))$ e $g(n) = o(h(n))$ implica que $f(n) = o(h(n))$
  - $f(n) = \omega(g(n))$ e $g(n) = \omega(h(n))$ implica que $f(n) = \omega(h(n))$

Comparação de funções

- Reflexividade
  - $f(n) = \Theta(f(n))$
  - $f(n) = O(f(n))$
  - $f(n) = \Omega(f(n)$

Comparação de funções

- Simetria
  - $f(n) = \Theta(g(n))$ se e somente se $g(n) = \Theta(f(n))$
- Simetria transposta
  - $f(n) = O(g(n))$ se e somente se $g(n) = \Omega(f(n))$
  - $f(n) = o(g(n))$ se e somente se $g(n) = \omega(f(n))$

Comparação de funções

$f(n) = O(g(n))$ $a \leq b$
$f(n) = \Omega(g(n))$ $a \geq b$
$f(n) = \Theta(g(n))$ $a = b$
$f(n) = o(g(n))$ $a < b$
$f(n) = \omega(g(n))$ $a > b$

Princípio da tricotomia

Para quaisquer dois números reais a e b, exatamente uma das comparações é verdadeira: $a < b$, $a = b$ ou $a > b$

Ops! Nem todas as funções são assintoticamente comparáveis: por exemplo, n e $n^{1+\text{sen } n}$

Monotonicidade

- uma função f(n) é **monotonicamente crescente** se m ≤ n implica que f(m) ≤ f(n)
- uma função f(n) é **monotonicamente decrescente** se m ≤ n implica que f(m) ≥ f(n)
- uma função f(n) é **estritamente crescente** se m < n implica que f(m) < f(n)
- uma função f(n) é **estritamente decrescente** se m < n implica que f(m) > f(n)

Algumas considerações sobre as notações

- O uso das notações permite comparar a taxa de crescimento das funções correspondentes aos algoritmos
    - **Não faz sentido comparar pontos isolados das funções**, já que podem não corresponder ao comportamento assintótico
    - Pode ser utilizado em **outros contextos** (não apenas para o tempo de execução)

Exemplo

- Para 2 algoritmos quaisquer, considere as funções de eficiência correspondentes 1000n e n^2
- A primeira é maior do que a segunda para valores pequenos de n
- A segunda cresce mais rapidamente e eventualmente será uma função maior, sendo que o ponto de mudança é n=1000
- Existe uma constante c e um ponto n_0 a partir do qual T(n) é menor ou igual a c*f(n)
    - No nosso caso, T(n)=1000n, f(n)= n^2, c=1 e n_0=1000 (ou, ainda, c=100 e n_0=10)
        - Dizemos que 1000n=O(n^2)

Classes de Comportamento Assintótico

Classes de Comportamento Assintótico

- As mais comuns

c	constante
$\log n$	logarítmica
$\log^2 n$	logarítmica ao quadrado
n	linear
$n \log n$	
n^2	quadrática
n^3	cúbica
2^n	
a^n	exponencial

Classes de Comportamento Assintótico

- Se **f** é uma função de complexidade para um algoritmo **F**, então **O(f)** é considerada a complexidade assintótica ou o comportamento assintótico de **F**.
    - **f(n) = O(1)**. Algoritmos de complexidade constante. O tempo de execução do algoritmo independe do tamanho do problema (valor n).
    - **f(n) = O(log n)**. Algoritmos de complexidade logarítmica. Comumente utilizamos base 2, mas pode-se também variar a base de acordo com o problema em questão.
    - **f(n) = O(n)**. Algoritmos de complexidade linear.

Classes de Comportamento Assintótico

- Se **f** é uma função de complexidade para um algoritmo **F**, então **O(f)** é considerada a complexidade assintótica ou o comportamento assintótico de **F**.
    - **f(n) = O(n log n)**. Comportamento de algoritmos que resolvem problemas dividindo-os em subproblemas. É bastante comum nos melhores algoritmos de ordenação por comparação.
    - **f(n) = O(n^2)**. Algoritmos de complexidade quadrática.
    - **f(n) = O(n^3)**. Algoritmos de complexidade cúbica.
    - **f(n) = O(n^k), k = 1, 2, ...** Algoritmos de complexidade polinomial. Englobam os lineares, quadráticos, cúbicos, etc.

Classes de Comportamento Assintótico

- Se **f** é uma função de complexidade para um algoritmo **F**, então **O(f)** é considerada a complexidade assintótica ou o comportamento assintótico de **F**.

    - **f(n) = O(c^n)**, **c** constante. Algoritmos de complexidade exponencial. Geralmente não são úteis do ponto de vista prático (não terminam em tempo hábil). Por exemplo, um algoritmo cuja função de complexidade seja 3^n , para n = 60 o algoritmo terminaria sua execução em aproximadamente 10^{13} séculos.

    - **f (n) = O(n!)**. Algoritmos de complexidade fatorial. Para ter uma ideia de quão ineficiente é um algoritmo de complexidade fatorial, para um problema de “tamanho” n = 40, o tempo de execução do algoritmo é proporcional a 40!, um número de 48 dígitos.

Relembrando um pouco de matemática...

Piso e teto

- para qualquer número real **x**, denotamos o maior número inteiro menor ou igual a **x** por $\lfloor x \rfloor$ e lemos “o piso de **x**”
- para qualquer número real **x**, denotamos o menor número inteiro maior ou igual a **x** por $\lceil x \rceil$ e lemos “o teto de **x**”

Exponenciação

- para todo número real a > 0, m e n, temos

$$a^0 = 1 ,$$

$$a^1 = a ,$$

$$a^{-1} = 1/a ,$$

$$(a^m)^n = a^{mn} ,$$

$$(a^m)^n = (a^n)^m ,$$

$$a^m a^n = a^{m+n} .$$

- para todo n e a ≥ 0, a função a n é monotonicamente crescente

Exponenciação

- para quaisquer constantes reais a e b, com a > 1, temos $a^0 = 1$,

$$\lim_{n \to \infty} \frac{n^b}{a^n} = 0,$$

$$n^b = o(a^n).$$

Logaritmos

- notação:

$$\lg n = \log_2 n$$
$$\ln n = \log_e n$$
$$\lg^k n = (\lg n)^k$$
$$\lg \lg n = \lg(\lg n)$$

- lg n + k significa (lg n) + k
- se b > 1 é uma constante, então para todo n > 0, a função $\log_b$ n é estritamente crescente

Logaritmos

- para todo número real a > 0,b > 0, c > 0 e n, temos

$$a = b^{\log_b a}$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b(1/a) = -\log_b a$$
$$\log_b a = \frac{1}{\log_a b}$$
$$a^{\log_b c} = c^{\log_b a}$$

em cada equação a base do logaritmo é diferente de 1

Exercícios

Exercícios

- Prove que
  - $10n^2 + 5n + 3 = \ominus(n^2)$
  - $7n^2 \neq O(n)$
  - $5 \log n + \sqrt{n} = O(\sqrt{n})$
  - $10n^3 - 3n^2 + 27 = O(n^3)$

Notação assintótica - resumo

- **f(n) = O(g(n))** se houver constantes positivas n_0 e **c** tal que **f(n) ≤ c g(n)** para todo **n ≥** n_0
- **f(n) = Ω(g(n))** se houver constantes positivas n_0 e c tal que **f(n) ≥ c g(n)** para todo **n ≥** n_0
- **f(n) = Θ(g(n))** se houver constantes positivas n_0, c_1 e c_2 tal que c_1 **g(n) ≤ f(n) ≤** c_2 **g(n)** para todo **n ≥** n_0
- **f(n) = o(g(n))** se, para qualquer constante positiva **c**, existe n_0 tal que **f(n) < c g(n)** para todo **n ≥** n_0
- **f(n) = ω(g(n))** se, para qualquer constante positiva **c**, existe n_0 tal que **f(n) > c g(n)** para todo **n ≥** n_0