PCC104 - Projeto e Análise de Algoritmos

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

Conteúdo

Fonte

Este material é baseado nos livros

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- S. Halim. *Competitive Programming*. 3rd Edition, 2013.
- Ian Parberry and William Gasarch. *Problems on Algorithms*. Second Edition, 2002.
- Ian Parberry *Lecture Notes on Algorithm Analysis and Complexity Theory*. Fourth Edition, 2001.

Licença

Este material está licenciado sob a Creative Commons BY-NC-SA 4.0. Isto significa que o material pode ser compartilhado e adaptado, desde que seja atribuído o devido crédito, que o material não seja utilizado de forma comercial e que o material resultante seja distribuído de acordo com a mesma licença.

Algoritmo

Definição Informal

É qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como **entrada** e produz algum valor ou conjunto de valores como **saída**.

Portanto, é uma sequência de passos computacionais que transformam uma entrada na saída.

Definição Informal 2

Uma ferramenta para resolver um **problema computacional** bem especificado.

O enunciado do problema computacional especifica, em termos gerais, o relacionamento entre entrada e saída.

O algoritmo descreve um procedimento computacional para alcançar esse relacionamento da entrada com a saída.

Exemplo

Problema de Ordenação

Ordenar uma sequência de números de maneira não decrescente.

Entrada

Uma sequência de n números $<a_1, a_2, \ldots, a_n>$.

Saída

Uma permutação $<a'_1, a'_2, \ldots, a'_n>$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Instância de um Problema

Uma **instância** de um problema consiste em uma entrada específica para qual se deseja calcular uma solução para o problema, por exemplo <31, 41, 59, 25, 41>.

Algoritmo

Definição Formal

Conjunto das regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número de etapas (Dicionário Houaiss da Língua Portuguesa, 2001, 1a edição).

Definição Formal 2

The term algorithm is used in computer science to describe a problem-solving method suitable for implementation as a computer program (Algorithms in C, Sedgewick, 1998, 3rd edition).

Observações

1. O número de etapas deve ser finito: se um algoritmo levar décadas, séculos ou milênios para executar, ele é impraticável;
2. Existem diferentes modelos computacionais nos quais os algoritmos podem ser implementados.

Algoritmo vs. Problema vs. Modelo Computacional

Um programa pode ser entendido como um algoritmo implementado em uma determinada linguagem para solucionar um problema computacional específico em um modelo computacional em particular.
Observações importantes:

- Nem todo problema computacional é solucionável;
- Diferentes problemas possuem diferentes níveis de dificuldade;
- Diferentes algoritmos possuem diferentes níveis de complexidade;
- Dependendo do modelo computacional utilizado, pode não haver algoritmo possível para determinado problema;
- Cada instrução executada em um modelo computacional possui um **custo de tempo**;
- Cada dado armazenado em um modelo computacional possui um **custo de espaço**.

Modelo Computacional

Modelo Computacional Genérico - Arquitetura de von Neumann

- Um único processador (ou unidade lógico aritmética);
- Memória RAM;
- Operações sequenciais, não há paralelismo;
- Instruções
  - Aritméticas (soma, subtração, multiplicação, divisão, resto, piso e teto);
  - Movimentação de dados (carregar, armazenar e copiar);
  - Controle (desvio condicional e incondicional, chamada e retorno de rotinas).
- Cada instrução tem tempo de execução constante, embora possam ser diferentes.

Com base nas operações definidas, outras operações mais complexas podem ser derivadas.

Algumas áreas "cinzas" como exponenciação são tratadas como tempo constante também.

Ian Parberry e William Gasarch, *Problems on Algorithms*

"*Algorithm analysis usually means 'give a big-O figure for the running time of an algorithm' (Of course, a big-Θ would be even better). This can be done by getting a big-O figure for parts of the algorithm and then combining these figures using the sum and product rules for big-O.*

Another useful technique is to pick an elementary operation, such as additions, multiplications or comparisons, and observing that the running time of the algorithm is big-O of the number of elementary operations. Then, you can analyze the exact number of operations as function of n in the worst case. This is easier to deal with because it is an exact function of n and you don't have the messy big-O symbols to carry through your analysis."

Análise de Algoritmos

Definição

Analisar um algoritmo significa prever os recursos que ele necessitará para sua execução. Os mais importantes são **tempo** e **espaço**.

Tempo de Execução

É o número de instruções primitivas executadas pelo algoritmo.

Espaço

É de fato o espaço necessário para armazenar dados durante a execução do algoritmo.

Utilidade

Com base na análise, podemos identificar a eficiência de cada algoritmo para um determinado problema. A análise é realizada levando em consideração um determinado modelo computacional.

Perspectivas

Definição

Além do ambiente computacional, o comportamento de um algoritmo pode variar de acordo com o comportamento da entrada (tamanho, estrutura, etc.), o que gera diferentes **perspectivas**.

Melhor Caso

A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema.

Pior Caso

A entrada está organizada de maneira que o algoritmo levará o tempo máximo para resolver o problema.

Caso Médio

A entrada está organizada de maneira que o algoritmo levará um tempo médio para resolver o problema.

Análise e Perspectivas

As análises se concentram geralmente no pior caso e no caso médio:

- O pior caso nos dá uma idéia de quão ruim pode ser o comportamento do algoritmo – cálculo razoavelmente simples, nos dá uma garantia de que o algoritmo não poderá ser mais lento. Pode ser crucial para aplicações críticas;
- O caso médio nos dá uma idéia de como o algoritmo se comportará em boa parte dos casos: determinar qual é o comportamento médio pode ser mais complexo, envolvendo probabilidades.

Funções Tipicamente Utilizadas

Representação

O comportamento do algoritmo é expressado como uma função matemática definida sobre o tamanho da entrada, denotada por $T(n)$.

Por exemplo, ordenar 3 números é mais rápido que ordenar 1000 usando o mesmo algoritmo, porém, ambos seguem uma mesma função de crescimento do tempo.

Geralmente, o tempo de execução aumenta com o aumento da entrada.

Análise Assintótica

A análise deve ser simplificada, e para a expressão da complexidade em cada perspectiva utilizamos a **análise assintótica**.

Estamos interessados mais no **comportamento**, na taxa de crescimento do tempo de execução do que de fato na precisão da função utilizada para expressar a complexidade.

Análise Assintótica

Ian Parberry e William Gasarch, *Problems on Algorithms*

"*It is useful for the analysis of algorithms since it captures the asymptotic growth pattern of functions and ignores the constant multiple (which is out of our control anyway when algorithms are translated into programs).*"

Funções Tipicamente Utilizadas

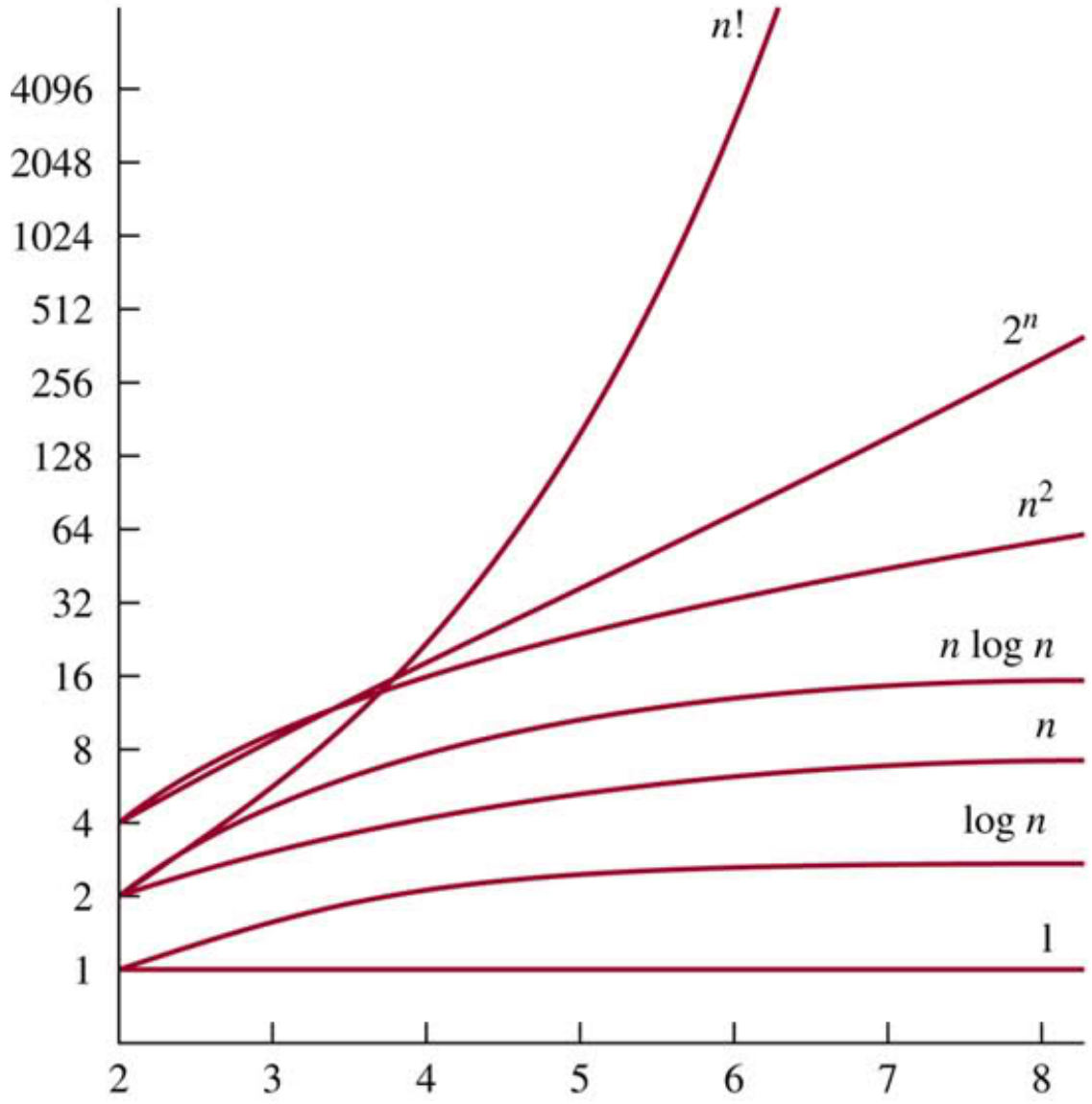
- k: constante em k, geralmente denotado como 1
  - Independe do tamanho de n.
- $log\ n$: logarítmico (geralmente a base é 2)
  - Menor do que uma constante grande;
  - n=1000, $log\ n$=10;
  - n=1.000.000, $log\ n$=20.
- n: linear
  - Aumenta no mesmo ritmo que n.
- $n\ log\ n$: linear logarítmico, ou apenas $n\ log\ n$
  - n=1000, $n\ log\ n$=10.000;
  - n=1.000.000, $n\ log\ n$=20.000.000.

Funções Tipicamente Utilizadas

- n^2: quadrático
  - n=1000, n^2=1.000.000.
- n^k: polinomial, proporcional à n elevado a k (constante);
- 2^n: exponencial, proporcional à 2 elevado a n
  - n=20, 2^n >1.000.000.
  - Quando n dobra, 2^n se eleva ao quadrado.
- $n!$: fatorial
  - n=10, $n!$= 3.628.800.
- n^n: sem nome formal, apenas n^n.

Comparação da Taxa de Crescimento de Funções

© The McGraw-Hill Companies, Inc. all rights reserved.



Taxa de Crescimento de Funções

Não afetam a taxa de crescimento

- Fatores constantes;
- Fatores de ordem mais baixa.

Exemplos

- $10^2n + 10^5$: é uma função linear;
- $10^5n^2 + 10^8n$: é uma função quadrática;
- $10^{-9}n^3 + 10^{20}n^2$: é uma função cúbica.

Descrição

Descreve como o tempo de execução cresce à medida em que a entrada aumenta indefinidamente, o comportamento de **limite**.

É uma análise teórica, independente de *hardware* e que utiliza funções cujos domínios são o conjunto dos números naturais.

Os termos de mais baixa ordem e constantes são ignorados.

Utiliza três notações: O, Θ e Ω.

Observação: Assumimos que as funções utilizadas nos slides a seguir são assintoticamente positivas, ou seja, são positivas para todo n suficientemente grande.

Notação Θ

Definição

Formalmente, para uma função $g(n)$ denotamos por $\Theta(g(n))$ o conjunto de funções $\Theta(g(n)) = \{f(n)$: existam constantes positivas c_1, c_2 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para $n \geq n_0\}$.

Significado

Significa dizer que uma função $f(n)$ pertence a $g(n)$ caso existam constantes c_1 e c_2 tal que ela seja "imprensada" entre $c_1 g(n)$ e $c_2 g(n)$.

Poderíamos escrever $f(n) \in \Theta(g(n))$, porém, em um abuso de notação escrevemos $f(n) = \Theta(g(n))$.

$f(n)$ é a função que expressa a complexidade de tempo, $T(n)$, e $g(n)$ geralmente é uma das funções vistas anteriormente.

Notação Θ

Limite Assintoticamente Restrito

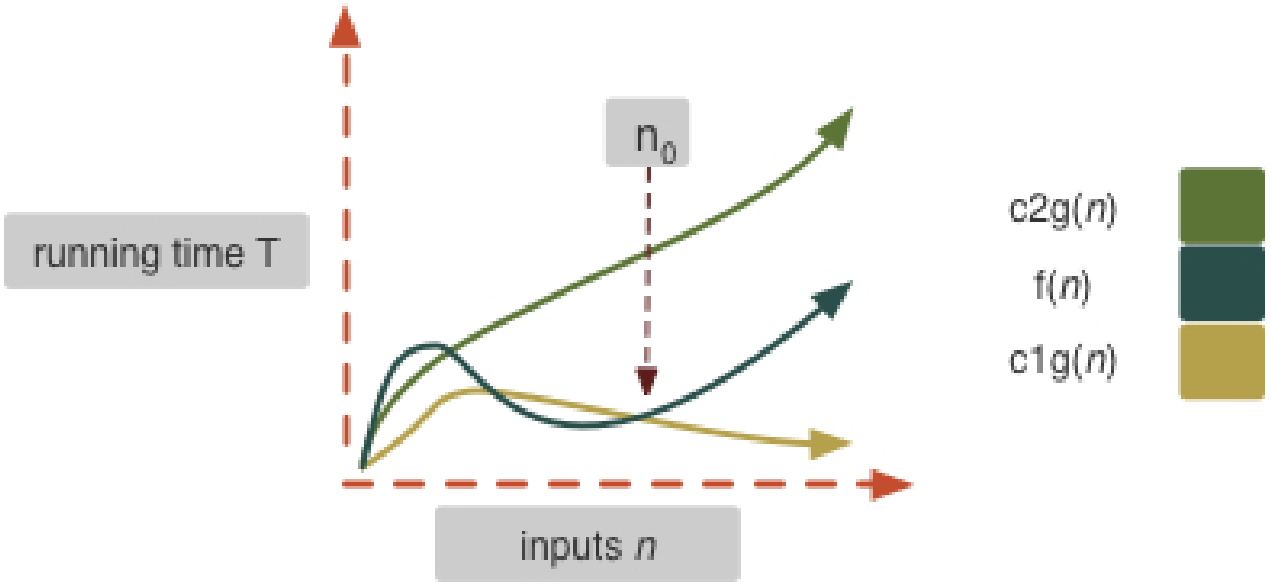
A função $g(n)$ denota o **limite assintoticamente restrito** para $f(n)$.

Para $n \geq n_0$, $f(n)$ estará "presa" entre duas inclinações de $g(n)$, ou seja, não será maior nem menor: a taxa de crescimento é igual.

Notação Θ

Theta-notation

$\Theta(g(n))$ = { $f(n)$: there are positive constant values c1, c2 and n0 where $0 \leq c1g(n) \leq f(n) \leq c2g(n)$ for all $n \geq n0$ }



Notação Θ

Exemplo

Vamos provar que $T(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Precisamos definir constantes 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 $n \geq n_0$.

Para simplificar, dividimos por n^2

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

A desigualdade da esquerda será válida se n=7 e $c_1 = \frac{1}{14}$.

A desigualdade da direita será válida se n=1 e $c_2 = \frac{1}{2}$.

Logo, se n_0=7, $c_1 = \frac{1}{14}$ e $c_2 = \frac{1}{2}$, temos que $T(n) = \Theta(n^2)$.

Notação O

Definição

Formalmente, para uma função $g(n)$ denotamos por $O(g(n))$ o conjunto de funções $O(g(n)) = \{f(n)$: existam constantes positivas c e n_0 tais que $0 \leq f(n) \leq cg(n)$ para $n \geq n_0\}$.

Significado

Significa dizer que uma função $f(n)$ pertence a $g(n)$ caso exista uma constante c tal que ela seja limitada superiormente por $cg(n)$.

Novamente, em um abuso de notação escrevemos $f(n) = O(g(n))$.

$f(n)$ é a função que expressa a complexidade de tempo, T(n), e $g(n)$ geralmente é uma das funções vistas anteriormente.

Notação O

Limite Assintoticamente Superior

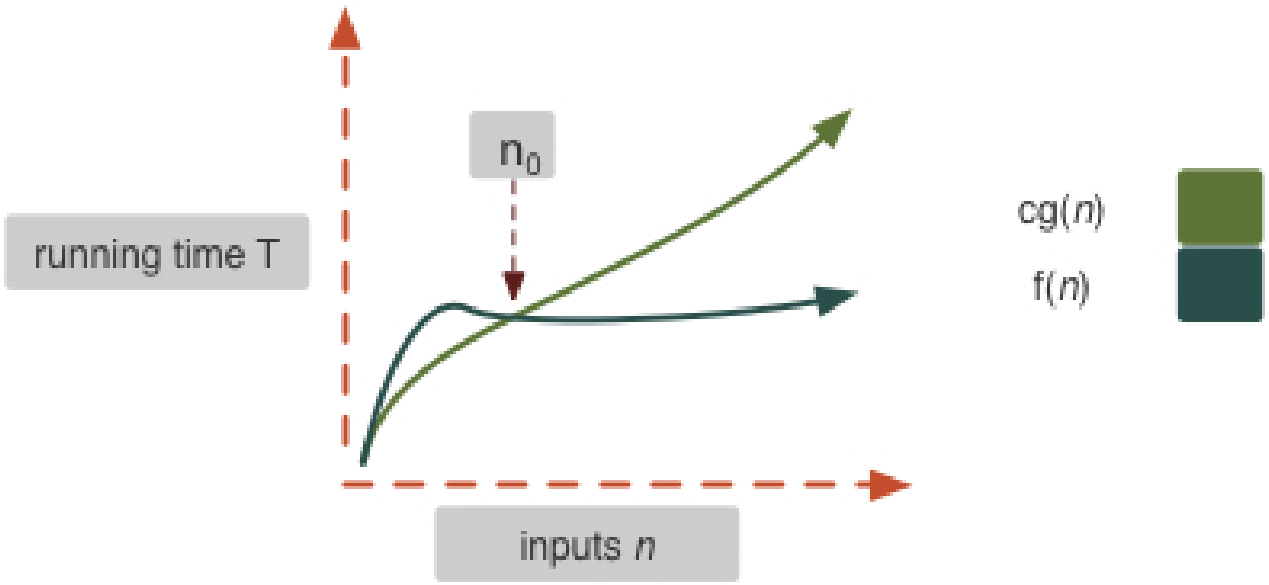
A função $g(n)$ denota o **limite assintoticamente superior** para $f(n)$.

Para $n \geq n_0$, $f(n)$ estará limitada por um fator constante de $g(n)$, ou seja, não será maior: a taxa de crescimento é no máximo igual.

Notação O

O-notation

O(g(*n*)) = { f(*n*) : there is a positive constant value c and n0 where 0 ≤ f(*n*) ≤ cg(*n*) for all *n* ≥ n0 }



Notação O

O e Θ

Note que se $\mathrm{T}(n) = \Theta(n)$, então não é incorreto dizer que $\mathrm{T}(n) = O(n^2)$. De fato, qualquer função linear é $O(n^2)$, porém, é impreciso.

Utilização

Geralmente a notação O é utilizada para descrever o pior caso.

Para maior precisão, combinamos notações e perspectivas da execução do algoritmo.

Exercício

Sendo $\mathrm{T}(n) = 6n^3+2n^2+72n+1$, qual é a complexidade utilizando a notação O?

Notação O

Operações

$$f(n) = O(f(n))$$

$$c \times f(n) = O(f(n)) \text{ } (c \text{ constante})$$

$$O(f(n)) + O(f(n)) = O(f(n))$$

$$O(O(f(n))) = O(f(n))$$

$$O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$$

$$O(f(n))O(g(n)) = O(f(n)g(n))$$

$$f(n)O(g(n)) = O(f(n)g(n))$$

Notação Ω

Definição

Formalmente, para uma função $g(n)$ denotamos por $\Omega(g(n))$ o conjunto de funções $\Omega(g(n)) = \{f(n)$: existam constantes positivas c e n_0 tais que $0 \leq cg(n) \leq f(n)$ para $n \geq n_0\}$.

Significado

Significa dizer que uma função $f(n)$ pertence a $g(n)$ caso existam constantes c e n_0 tal que ela seja limitada inferiormente por $cg(n)$ para $n \geq n_0$.

Novamente, em um abuso de notação escrevemos $f(n) = \Omega(g(n))$.

Notação Ω

Limite Assintoticamente Inferior

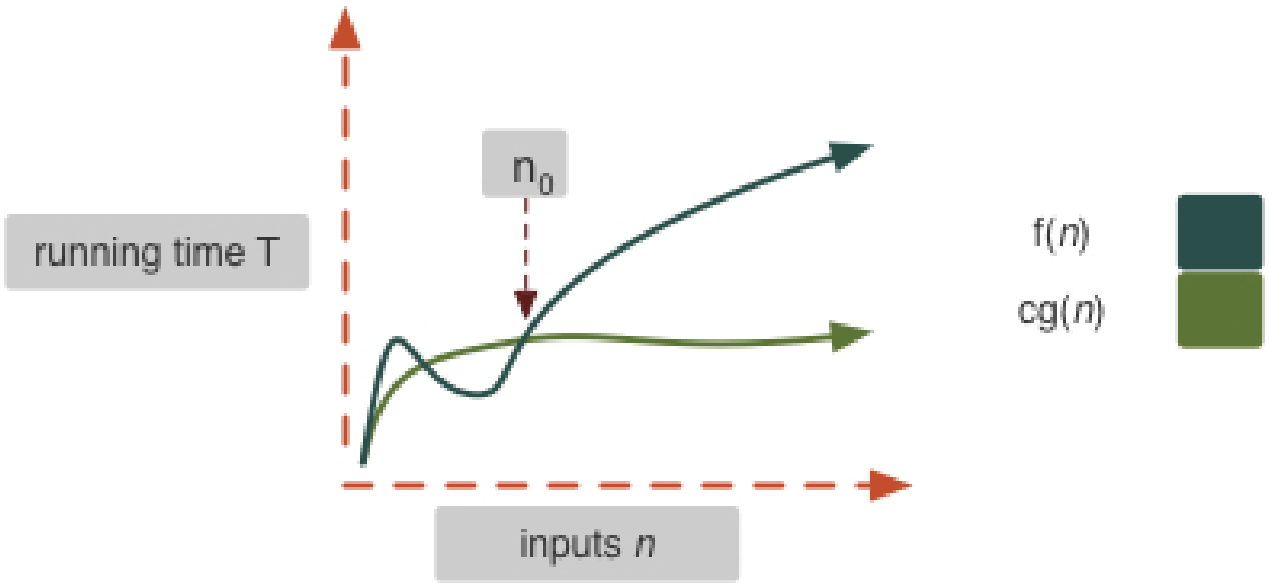
$f(n)$ é a função que expressa a complexidade de tempo, T(n), e $g(n)$ geralmente é uma das funções vistas anteriormente, chamada de **limite assintoticamente inferior** para $f(n)$.

Para $n \geq n_0$, $f(n)$ estará limitada por um fator constante de $g(n)$, ou seja, não será menor: a taxa de crescimento é no mínimo igual.

Notação Ω

Omega-notation

$\Omega(g(n)) = \{ f(n)$: there is a positive constant value c and n0 where $0 \leq cg(n) \leq f(n)$ for all $n \geq n0 \}$



Notação Ω

Exemplo 1

Vamos provar que T(n)=n^3+2n^2 é $\Omega(n^3)$.

Sejam c=1 e $n_0 = 0$, então T(n) $\geq n^3$ para $n \geq n_0$.

Exemplo 2

Vamos provar que, considerando T(n)=n para n ímpar ($n \geq 1$) e T(n) = $\frac{n^2}{10}$ para n par ($n \geq 0$), então T(n)= $\Omega(n^2)$.

Sejam c=$\frac{1}{10}$ e $n = 0, 2, 4, 6$, então T(n) $\geq n^2$.

Relação Entre Notações

Θ, O e Ω

Para um dado T(n), se T(n) = $\Omega(n^2)$ e T(n) = $O(n^2)$, então T(n) = $\Theta(n^2)$, pois T(n) é limitado inferiormente e superiormente por n^2.

Conclusões

- Na análise de complexidade de um algoritmo estamos mais interessados no seu comportamento geral do que em outros detalhes que podem depender da máquina, do sistema operacional, da linguagem, dos compiladores, etc.
- Procura-se medir a complexidade de um algoritmo em função de um parâmetro do problema, geralmente, o tamanho da entrada;
- Alguma operação (ou conjunto de operações) devem balizar a análise de complexidade de um algoritmo;
- Considera-se a eficiência assintótica dos algoritmos executados em máquinas que operam em um determinado modelo computacional;
- Dois algoritmos para o mesmo problema com análises assintóticas iguais precisam ser comparados em experimentos computacionais.

Dúvidas?