Estrutura de Dados I

Análise Empírica e Assintótica

Prof. Rodrigo Minetto

Universidade Tecnológica Federal do Paraná

Sumário

Análise de algoritmos

A análise de algoritmos, descrita e difundida por Donald Knuth (autor da série de livros **The Art of Computer Programming**), pode ser definida como o estudo da **eficiência** ou **complexidade** de um algoritmo em função do tamanho do problema, do número de passos necessário (complexidade temporal), e da memória do sistema usado para executar o algoritmo (complexidade espacial).

Análise de algoritmos

O objetivo não é apenas fazer códigos corretos, mas que sejam também eficientes. **Pergunta**: qual a funcionalidade e complexidade do algoritmo a seguir?

Algoritmo-A (**int** n)

```
1. s = 0;
2. for (i = 1; i ≤ n; i++)
3.     for (j = 1; j ≤ i; j++)
4.         s = s + 1;
5. return s;
```

Análise de algoritmos

O objetivo não é apenas fazer códigos corretos, mas que sejam também eficientes. **Pergunta**: qual a funcionalidade e complexidade do algoritmo a seguir?

Algoritmo-B (**int** **n**)

```
1. s = 0;
2. for (i = 1; i ≤ n; i++)
4.       s = s + i;
5. return s;
```

Análise de algoritmos

O objetivo não é apenas fazer códigos corretos, mas que sejam também eficientes. **Pergunta**: qual a funcionalidade e complexidade do algoritmo a seguir?

Algoritmo-C (**int** **n**)

1. **s** $= (\mathbf{n} \times (\mathbf{n} + 1))/2;$
2. **return** **s**;

Sumário

Análise empírica de algoritmos

As técnicas de análise matemática podem ser aplicadas com sucesso em muitos algoritmos simples. Porém, a análise matemática pode ser muito difícil, principalmente a análise do caso médio. Uma alternativa à análise matemática da eficiência de um algoritmo é a **análise empírica**.

Análise empírica de algoritmos

Plano geral para a análise empírica:

- Selecionar um algoritmo (módulo ou função) a ser analisado;
- Escolher uma métrica de eficiência:
  - unidade de tempo;
  - número de operações básicas;
- Selecionar amostras para avaliar o desempenho do algoritmo:
  - faixa e distribuição de valores;
  - tamanho (n, $2n$, $4n$, ... ou n, $10n$, $100n$, ...);
- Executar o algoritmo sobre as amostras;
- Analisar os dados.

Análise empírica de algoritmos

Objetivos:

- Avaliar a corretude do algoritmo (identificar erros);
- Comparar a eficiência de diferentes algoritmos;
- Comparar implementações alternativas (otimizadas) do mesmo algoritmo;
- Estimar a complexidade de um algoritmo;

Análise empírica de algoritmos

Problemas:

- Tempo do sistema geralmente não é preciso (diferentes tempos para execuções iguais);
- Tempo registrado pode ser zero (velocidade dos computadores);
- Tempo de execução *vs* Tempo real (usertime);

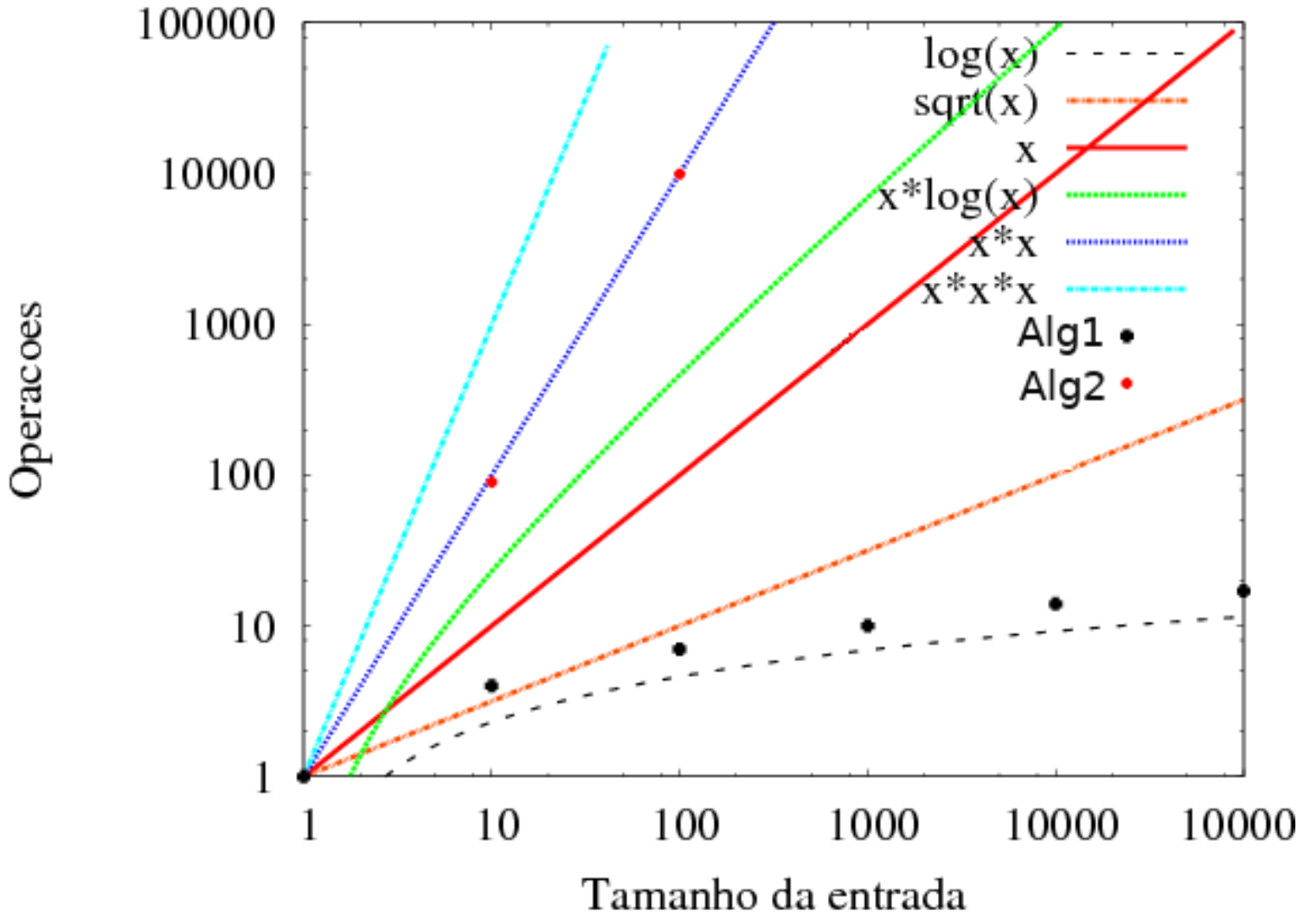
Análise empírica de algoritmos

Como analisar os dados?

- Tabular os dados para as diferentes entradas;
- Representar os dados graficamente;

Análise empírica de algoritmos

Como analisar os dados?



Análise empírica de algoritmos

Vantagens:

- Aplicável a qualquer algoritmo/função/módulo;
- Pode ser realizada dentro do próprio sistema;
- Fornece informação específica sobre a performance de um algoritmo em um ambiente particular;
- Pode revelar gargalos na performance do sistema;

Análise empírica de algoritmos

Desvantagens:

- Depende das características específicas do ambiente de teste;
- Pode não permitir comparações genéricas com algoritmos cujos testes foram realizados em outro ambiente;
- Geralmente buscamos utilizar uma entrada típica. Mas o que é uma entrada típica? Quais amostras devemos utilizar?

Sumário

Análise assintótica de algoritmos

Ao ver uma expressão como $n + 10$ ou $n^2 + 1$, a maioria das pessoas pensa automaticamente em valores pequenos de n, valores próximos de zero. A análise de algoritmos faz exatamente o contrário: ignora os valores pequenos e concentra-se nos valores enormes de n.

Análise assintótica de algoritmos

Para valores enormes de n, as funções

$$n^2 \qquad \frac{3n^2}{2} \qquad 9999n^2 \qquad \frac{n^2}{1000} \qquad n^2 + 100n$$

crescem todas com a mesma velocidade (**ordem**) e portanto são todas "equivalentes". Esse tipo de matemática, interessada somente em valores enormes de n, é chamado assintótico.

Análise assintótica de algoritmos

Quando observamos tamanhos de entradas grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução, estamos estudando a **eficiência assintótica** dos algoritmos — Algoritmos. 2 edição. Cormen et al.

Complexidade de tempo ou de espaço

Em análise de algoritmos conta-se o número de operações consideradas relevantes realizadas pelo algoritmo e expressa-se esse número como uma função de **n**.

Complexidade de tempo ou de espaço

Vamos expressar complexidade através de funções em variáveis que descrevam o tamanho de instâncias do problema. Exemplos:

- Problemas de aritmética de precisão arbitrária: número de bits (ou bytes) dos inteiros.
- Problemas em grafos: número de vértices e/ou arestas
- Problemas de ordenação de vetores: tamanho do vetor.
- Busca em textos: número de caracteres do texto ou padrão de busca.

Medida de complexidade e eficiência de algoritmos

Vamos supor que funções que expressam complexidade são **sempre positivas**, já que estamos medindo número de operações.

A complexidade de tempo (= eficiência) de um algoritmo é o número de instruções básicas que ele executa em função do tamanho da entrada.

Medida de complexidade e eficiência de algoritmos

O número de operações realizadas por um determinado algoritmo pode depender da particular instância da entrada. Em geral interessa-nos o pior caso, i.e., o maior número de operações usadas para qualquer entrada de tamanho n.

Análises também podem ser feitas para o melhor caso e o caso médio. Neste último, supõe-se conhecida uma certa distribuição da entrada.

Medida de complexidade e eficiência de algoritmos

Um algoritmo é chamado eficiente se a função que mede sua complexidade de tempo é limitada por um polinômio no tamanho da entrada. Por exemplo: n, $3n - 7$, $4n^2$, $143n^2 - 4n + 2$, n^5.

Mas por que polinômios?

Resposta: polinômios são funções bem "comportadas".

Algoritmos e tecnologia

Considere 5 algoritmos com diferentes complexidade de tempo. Suponha que uma operação leve 1 ms.

n	n	$nlogn$	n^2	n^3	2^n
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10137 séc.

Para uma máquina mais rápida onde uma operação leve 1 ps ao invés de 1 ms, precisariamos 10128 séculos ao invés de 10137 séculos :-)

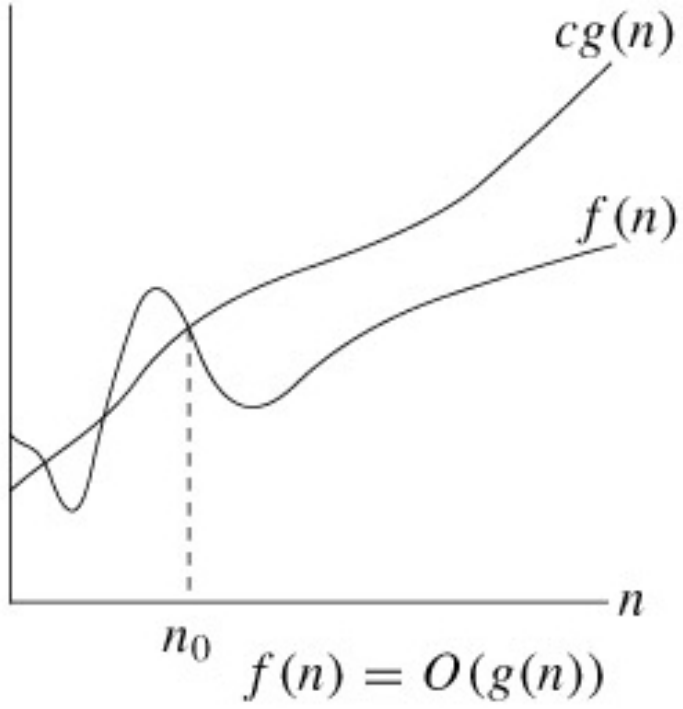
Sumário

Notação O

Quando afirmamos que "o tempo de execução do algoritmo é $O(n^2)$", equivale a dizer que o tempo de execução do pior caso é $O(n^2)$.

Exemplos de funções $O(n^2)$

- $f(n) = n^2$
- $f(n) = n^2 + n$
- $f(n) = n^2 - n$
- $f(n) = 1000n^2 + 1000n$
- $f(n) = n$
- $f(n) = n/1000$
- $f(n) = n^{1.999999}$
- $f(n) = logn$
- $f(n) = \sqrt{n}$

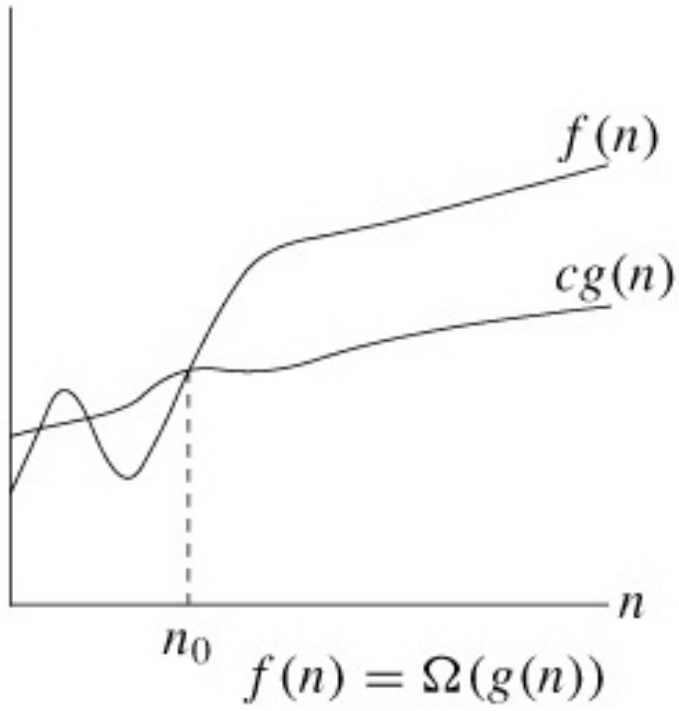


Notação Ω

Quando afirmamos que "o tempo de execução do algoritmo é $\Omega(n^2)$", equivale a dizer que o tempo de execução no melhor caso é $\Omega(n^2)$.

Exemplos de funções $\Omega(n^2)$

- $f(n) = n^2$
- $f(n) = n^2 + n$
- $f(n) = n^2 - n$
- $f(n) = 1000n^2 + 1000n$
- $f(n) = 1000n^2 - 1000n$
- $f(n) = n^3$
- $f(n) = n^{2.0000001}$
- $f(n) = n^2 log_2 log_2 log_2 n$
- $f(n) = 2^n$

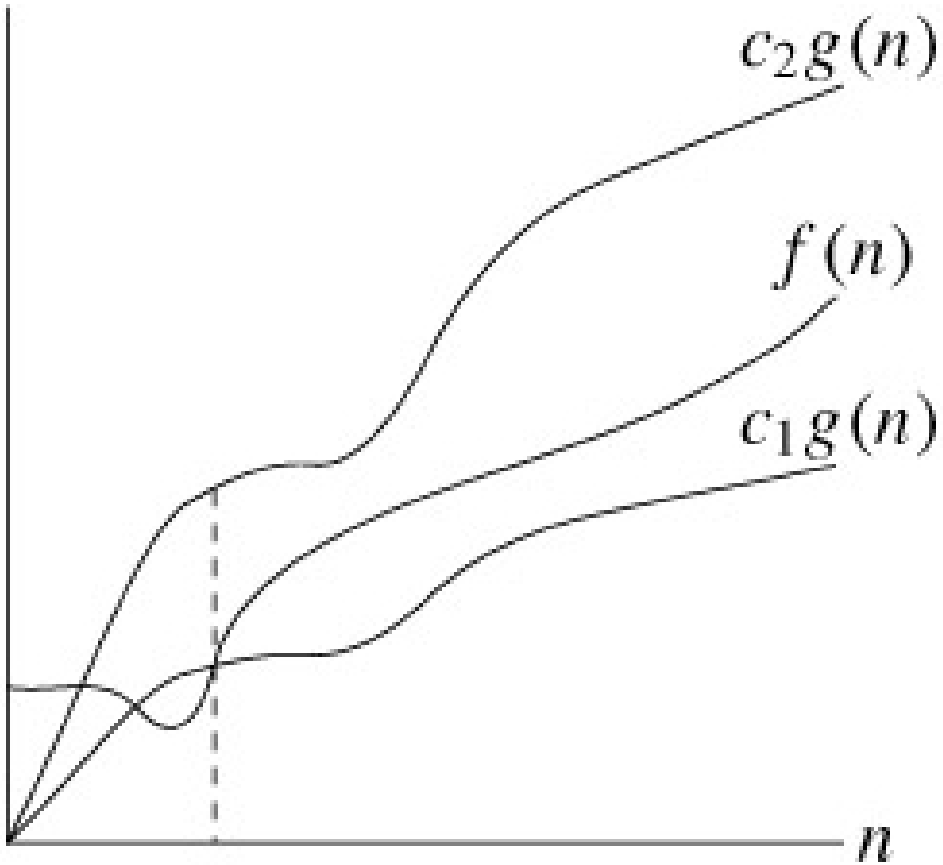


Notação Θ (limite assintótico restrito)

Dadas duas funções $f(n)$ e $g(n)$, temos $f(n) = \Theta(g(n))$ se e somente se

- $f(n) = O(g(n))$
- $f(n) = \Omega(g(n))$

Notação Θ



$$f(n) = \Theta(g(n))$$

Sumário

Notação O (limite assintótico superior)

Definição

Uma função $f(n)$ está em $O(g(n))$ se existem constantes positivas c e n_0 tais que
$0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0$

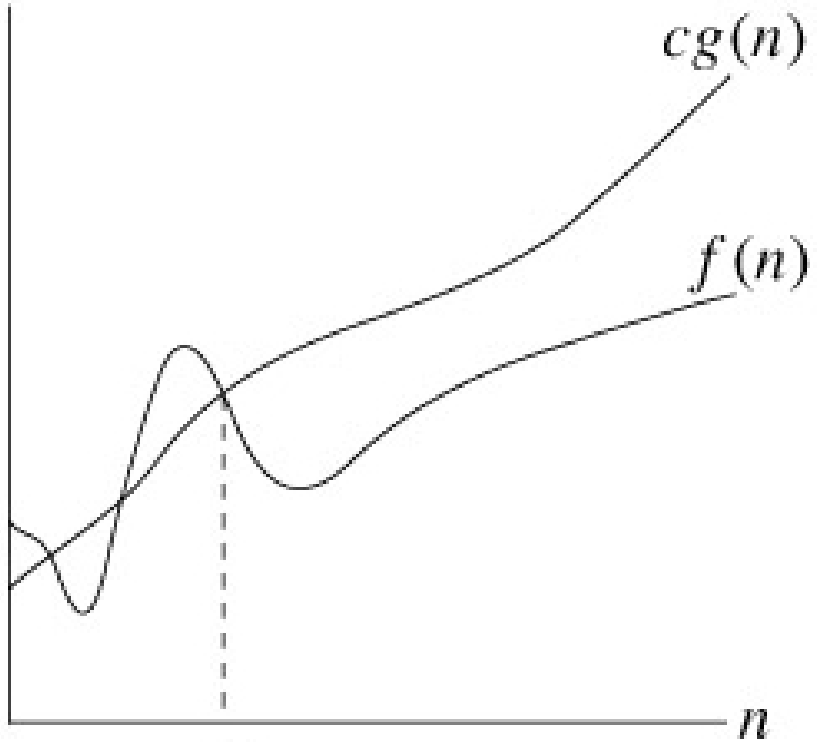
▷ "f está em $O(g)$" tem jeito de "$f \leq g$"

▷ tem jeito de "f não cresce mais que g"

▷ conceito sob medida para tratar consumo de tempo de algoritmos

Notação O

Isto é, para valores de n suficientemente grandes, $f(n)$ é igual ou menor que $g(n)$.



Notação O - Exemplo

Usando a definição mostre que $\frac{1}{2}n^2 - 3n = O(n^2)$, onde $f(n) = \frac{1}{2}n^2 - 3n$ e $g(n) = n^2$.

Definição: $f(n) = O(g(n))$ se $0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0$.

Para isso devemos definir constantes positivas c e n_0 tais que

$$0 \leq \frac{1}{2}n^2 - 3n \leq cn^2 \tag{1}$$

para todo $n \geq n_0$.

Notação O - Exemplo

$$0 \leq \frac{1}{2}n^2 - 3n \leq cn^2 \tag{2}$$

para todo $n \geq n_0$.

A divisão por n^2 produz

$$0 \leq \frac{1}{2} - \frac{3}{n} \leq c \tag{3}$$

Para $c = \frac{1}{2}$ e $n \geq 7$ $(n_0 = 7) \rightarrow \frac{1}{2}n^2 - 3n = O(n^2)$.

Sumário

Notação Ω (limite assintótico inferior)

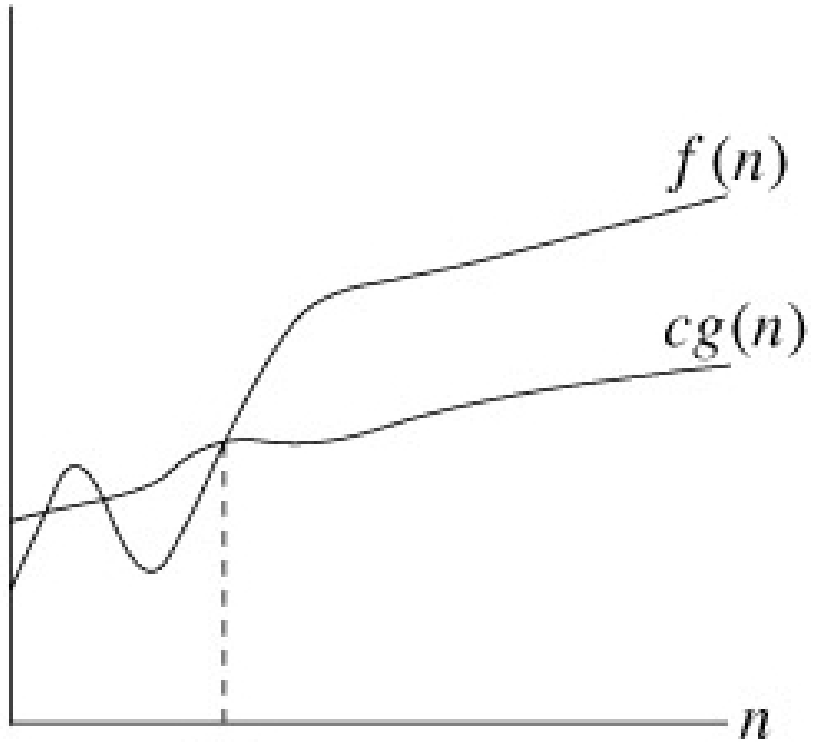
Definição

Uma função $f(n)$ está em $\Omega(g(n))$ se existem constantes positivas c e n_0 tais que
$0 \leq cg(n) \leq f(n)$ para todo $n \geq n_0$

$\triangleright$ "$f \in \Omega(g)$" tem jeito de "$f \geq g$"

Notação Ω

Isto é, para valores de n suficientemente grandes, $f(n)$ é igual ou **maior** que $g(n)$.



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

Notação Ω - Exemplo

Usando a definição mostre que $\frac{1}{2}n^2 - 3n = \Omega(n^2)$, onde $f(n) = \frac{1}{2}n^2 - 3n$ e $g(n) = n^2$.

Definição: $f(n) \in \Omega(g(n))$ se $0 \leq cg(n) \leq f(n)$ para todo $n \geq n_0$.

Para isso devemos definir constantes positivas c e n_0 tais que

$$0 \leq cn^2 \leq \frac{1}{2}n^2 - 3n \tag{4}$$

para todo $n \geq n_0$.

Notação Ω - Exemplo

$$0 \leq cn^2 \leq \frac{1}{2}n^2 - 3n \tag{5}$$

para todo $n \geq n_0$.

A divisão por n^2 produz

$$0 \leq c \leq \frac{1}{2} - \frac{3}{n} \tag{6}$$

Para $c = \frac{1}{14}$ e $n \geq 7$ $(n_0 = 7) \rightarrow \frac{1}{2}n^2 - 3n = \Omega(n^2)$.

Notação Ω

Considerando que a notação Ω descreve um limite inferior, quando a usamos para limitar o tempo de execução do melhor caso de um algoritmo, por implicação também limitamos o tempo de execução do algoritmo sobre entradas arbitrárias.

Notação Ω

Por exemplo, o tempo de execução do melhor caso da ordenação por inserção é $\Omega(n)$. Assim, o tempo de execução da ordenação por inserção recai entre $\Omega(n)$ e $O(n^2)$, pois ele fica em qualquer lugar entre uma função linear de n e uma função quadrática de n.

Sumário

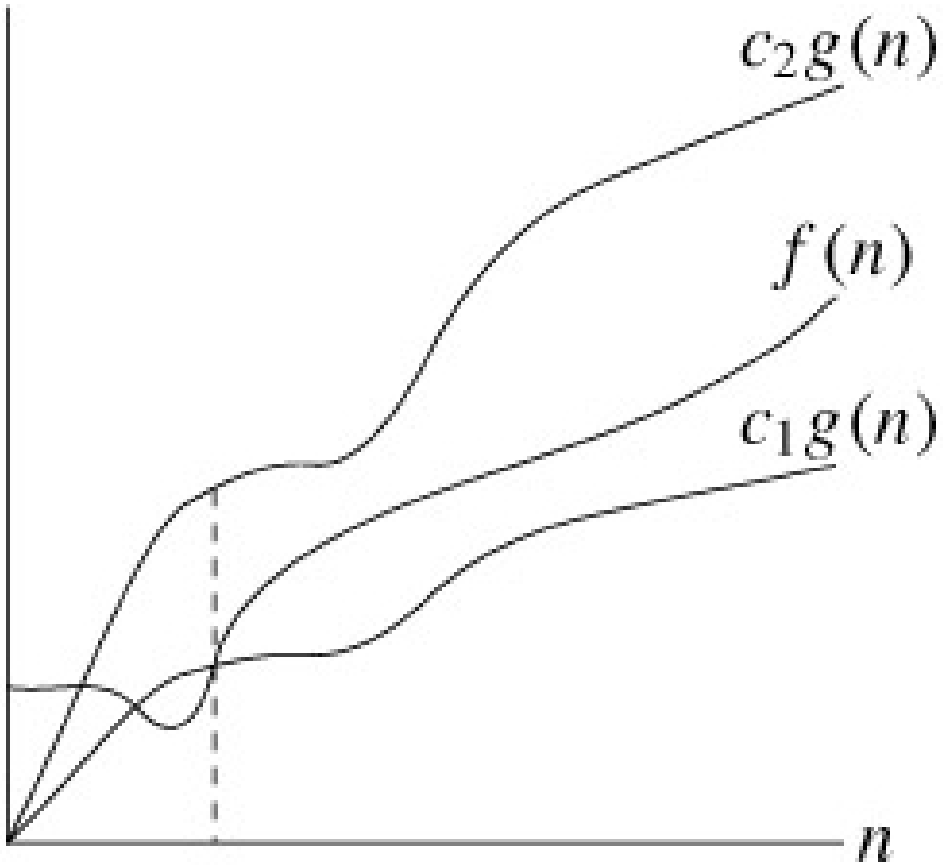
Notação Θ (limite assintótico restrito)

Dadas duas funções $f(n)$ e $g(n)$, temos $f(n) = \Theta(g(n))$ se e somente se

- $f(n) = O(g(n))$
- $f(n) = \Omega(g(n))$

Em outras palavras, existem números positivos c_1, c_2 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0$.

Notação Θ



$$f(n) = \Theta(g(n))$$