Estrutura de Dados II

Análise Empírica e Assintótica

Prof. Rodrigo Minetto
Profa. Juliana de Santi
Universidade Tecnológica Federal do Paraná
Material compilado de: Cormen, IC-UNICAMP e IME-USP

Sumário

Análise de algoritmos

Qual o custo do trecho de código abaixo?

```
...
for(i = 0; i < n; i++) {
   a[i] = b[i] + c[i];
}
for(i = 0; i < n; i++) {
   d[i] = a[i]*(e[i] + 1.0);
}
...
```

Análise de algoritmos

Podemos otimizar esse código?

```
...
for(i = 0; i < n; i++) {
   a[i] = b[i] + c[i];
}
for(i = 0; i < n; i++) {
   d[i] = a[i]*(e[i] + 1.0);
}
...
```

Análise de algoritmos

Qual o custo do trecho de código abaixo?

```
...
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i][j] = 0.0;
    }
}
for (i = 0; i < n; i++) {
    a[i][i] = 1.0;
}
...
```

Análise de algoritmos

Podemos otimizar esse código?

```
...
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i][j] = 0.0;
    }
}
for (i = 0; i < n; i++) {
    a[i][i] = 1.0;
}
...
```

Análise de algoritmos

Qual o custo do trecho de código abaixo?

```
...
for (i = 0; i < n; i++) {
    double sum = 0.0;
    for (j = 0; j < n; j++) {
        sum += cos(j);
    }
    a[i] *= sum;
}
...
```

Análise de algoritmos

Podemos otimizar esse código?

```
...
for (i = 0; i < n; i++) {
    double sum = 0.0;
    for (j = 0; j < n; j++) {
        sum += cos(j);
    }
    a[i] *= sum;
}
...
```

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

Modos de análise:

- Contador: contar o número de vezes que uma operação básica é realizada;
- Temporizador: medir o tempo de execução em um fragmento do código
  - comando *time* do UNIX;
  - funções: clock e System.currentTimeMillis;

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);

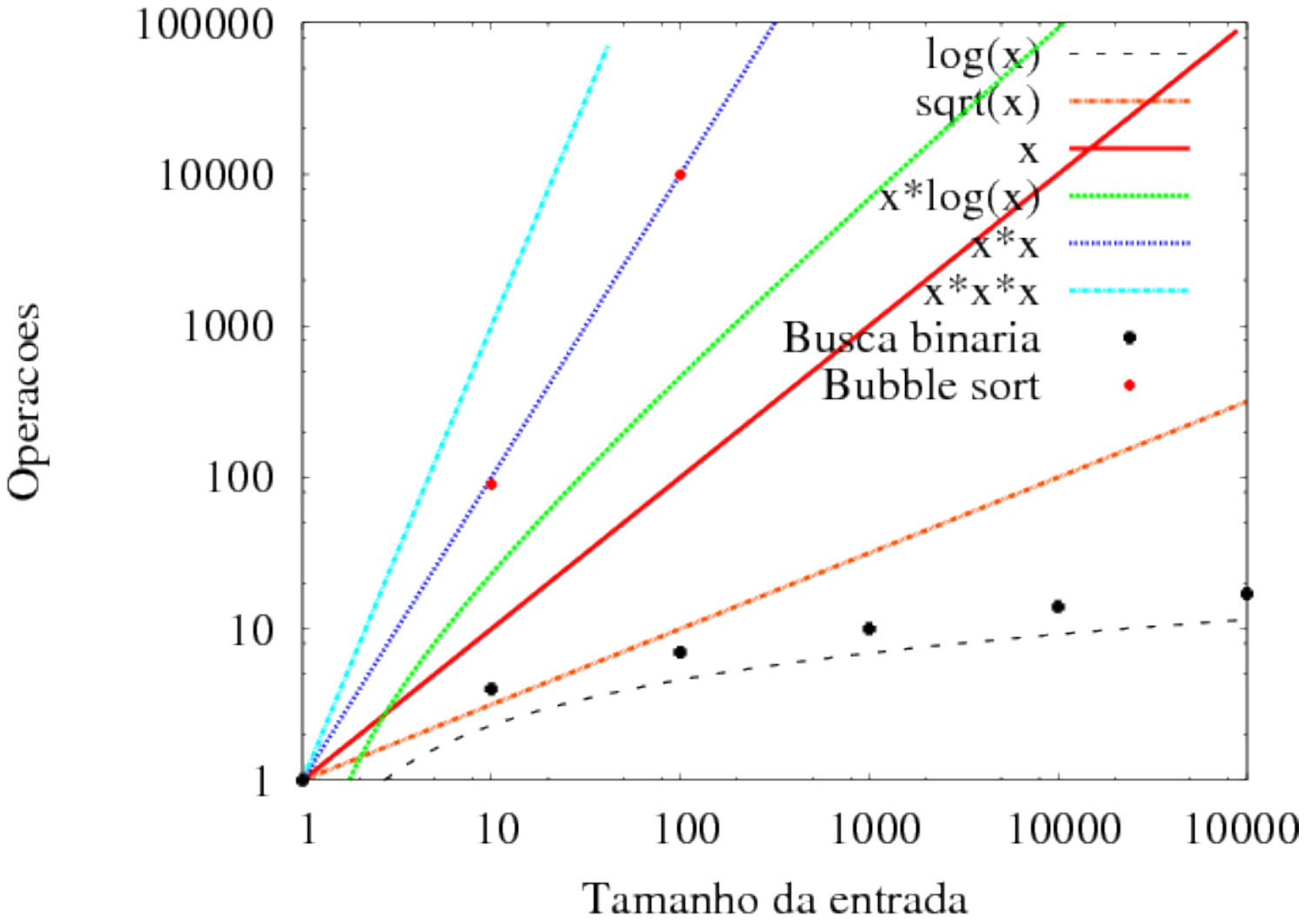
Análise empírica de algoritmos

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.

Algoritmos e tecnologia

Exemplo: **ordenação de n elementos**

- Como podemos comparar os dois algoritmos para escolher o melhor?
- Suponha que os computadores A e B executam 1G e 10M instruções por segundo, respectivamente. Ou seja, A é 100 vezes mais rápido que B. Para um dado problema considere dois algoritmos que o resolvem e seja n o tamanho da entrada do algoritmo.
- Algoritmo 1: implementado em A por um excelente programador em linguagem de máquina (ultra-rápida). Executa $2n^2$ instruções.
- Algoritmo 2: implementado em B por um programador mediano em linguagem de alto nível dispondo de um compilador "meia-boca". Executa $50nlogn$ instruções.

Algoritmos e tecnologia

O que acontece quando ordenamos um vetor de um milhão de elementos? Qual algoritmo é mais rápido?

- Algoritmo 1 na máquina A:

$$\frac{2.(10^6)^2 \ \text{instruções}}{10^9 \ \text{instruções/segundo}} = 2000 \ \text{segundos} \qquad (1)$$

- Algoritmo 2 na máquina B:

$$\frac{50.(10^6 log 10^6) \ \text{instruções}}{10^7 \ \text{instruções/segundo}} = 100 \ \text{segundos} \qquad (2)$$

- Ou seja, B foi VINTE VEZES mais rápido do que A!
- Se o vetor tiver 10 milhões de elementos, esta razão será de 2.3 dias para 20 minutos!

Algoritmos e tecnologia

Como dito anteriormente, geralmente nos concentramos na análise de pior caso e no comportamento assintótico dos algoritmos (instâncias de tamanho grande).

Algoritmos e tecnologia

O algoritmo Insertion-Sort tem como complexidade (de pior caso) uma função quadrática $an^2 + bn + c$, onde a, b e c são constantes absolutas.

O estudo assintótico nos permite "jogar para debaixo do tapete" os valores destas constantes, i.e., aquilo que independe do tamanho da entrada (neste caso os valores de a, b e c). **Por que podemos fazer isso?**

Algoritmos e tecnologia

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

n	$3n^2 + 10n + 50$	n^2
64	12.978	4.096
128	50.482	16.384
512	791.602	262.144
1024	3.156.018	1.048.576
2048	12.603.442	4.194.304
4096	50.372.658	16.777.216
8192	201.408.562	67.108.864
16384	805.470.258	268.435.456
32768	3.221.553.202	1.073.741.824

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

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 (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 $n \geq n_0$

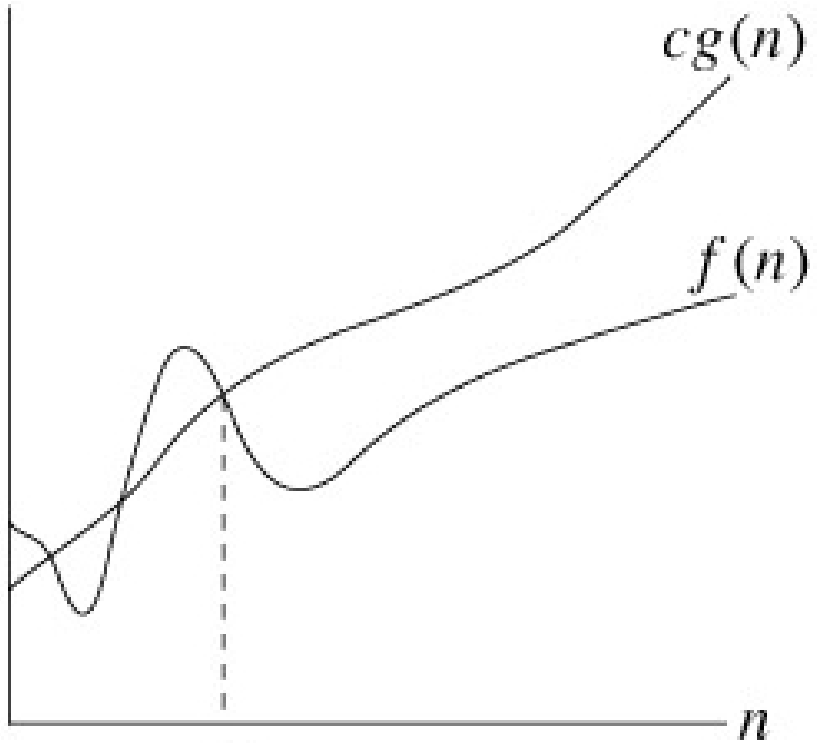
▷ "f está em $O(g)$" tem jeito de "$f \leq g$"

▷ 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

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

para todo $n \geq n_0$.

Notação O - Exemplo

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

para todo $n \geq n_0$.

A divisão por n^2 produz

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

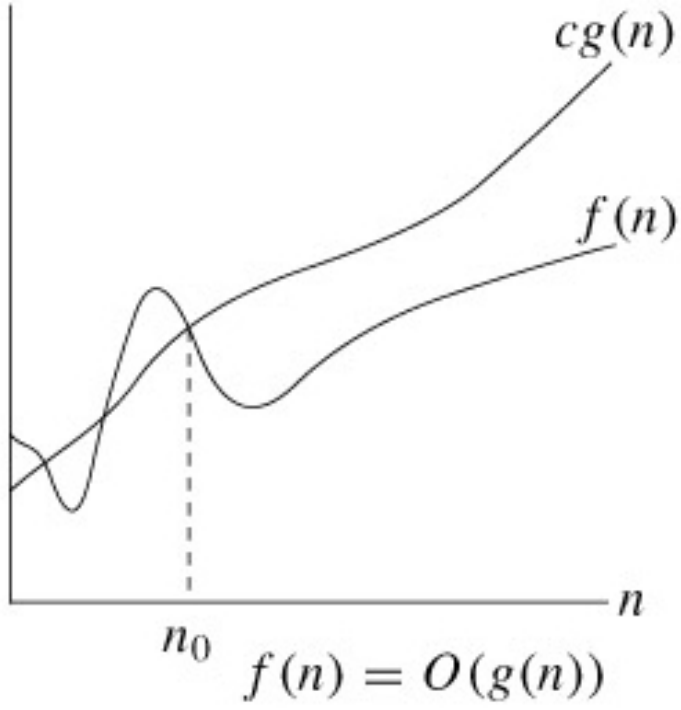
Para $c = \frac{1}{2}$ e $n \geq 7$ $(n_0 = 7) \rightarrow \frac{1}{2}n^2 - 3n = O(n^2)$.

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 (limite assintótico inferior)

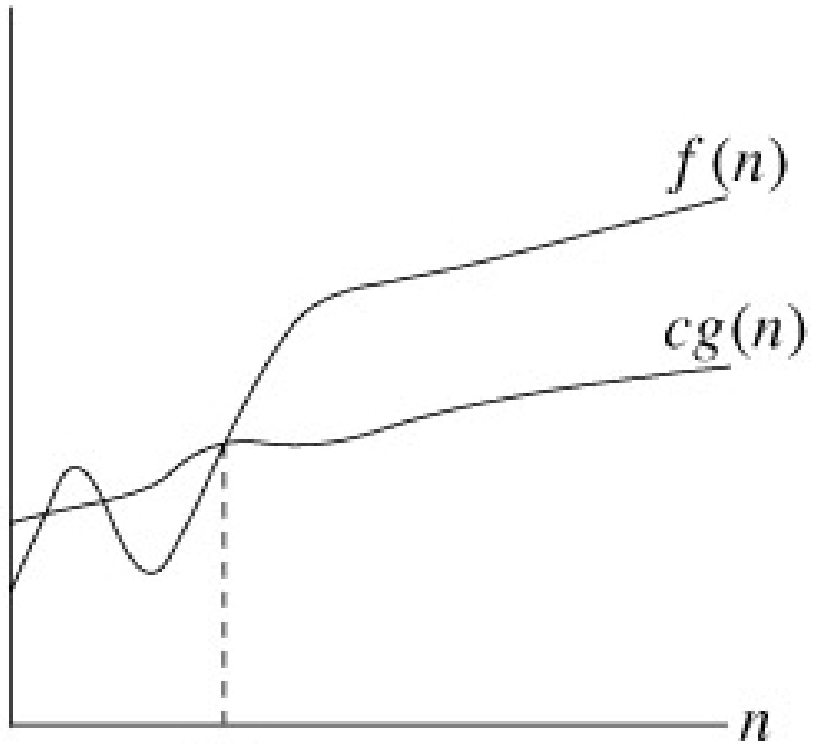
Definição

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

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

para todo $n \geq n_0$.

Notação Ω - Exemplo

$$0 \leq cn^2 \leq \frac{1}{2}n^2 - 3n \tag{7}$$

para todo $n \geq n_0$.

A divisão por n^2 produz

$$0 \leq c \leq \frac{1}{2} - \frac{3}{n} \tag{8}$$

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

Notação Ω

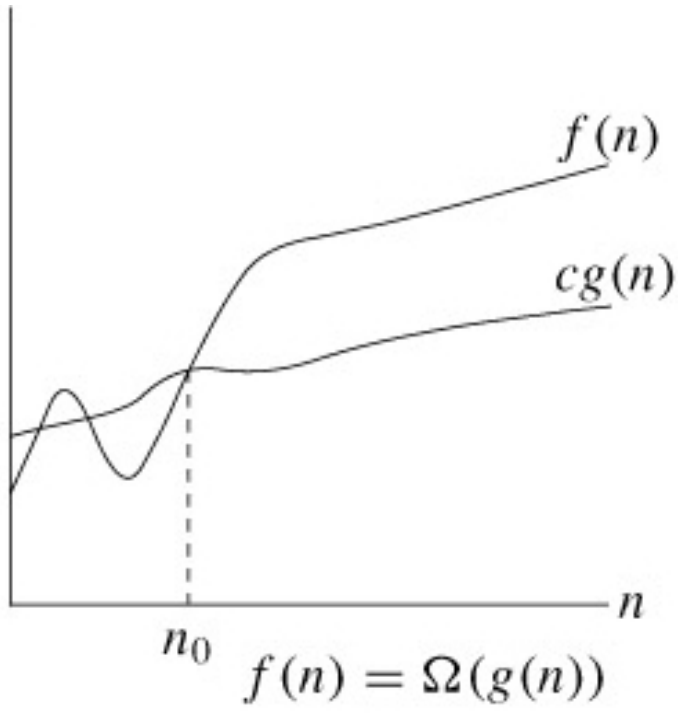
Considerando que a notação Ω descreve um limite inferior, quando a usamos para limitar o tempo de execução do melhor caso de um algoritmo, por implicação também limitamos o tempo de execução do algoritmo sobre entradas arbitrárias.

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.

Exemplos de funções $\Omega(n^2)$

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



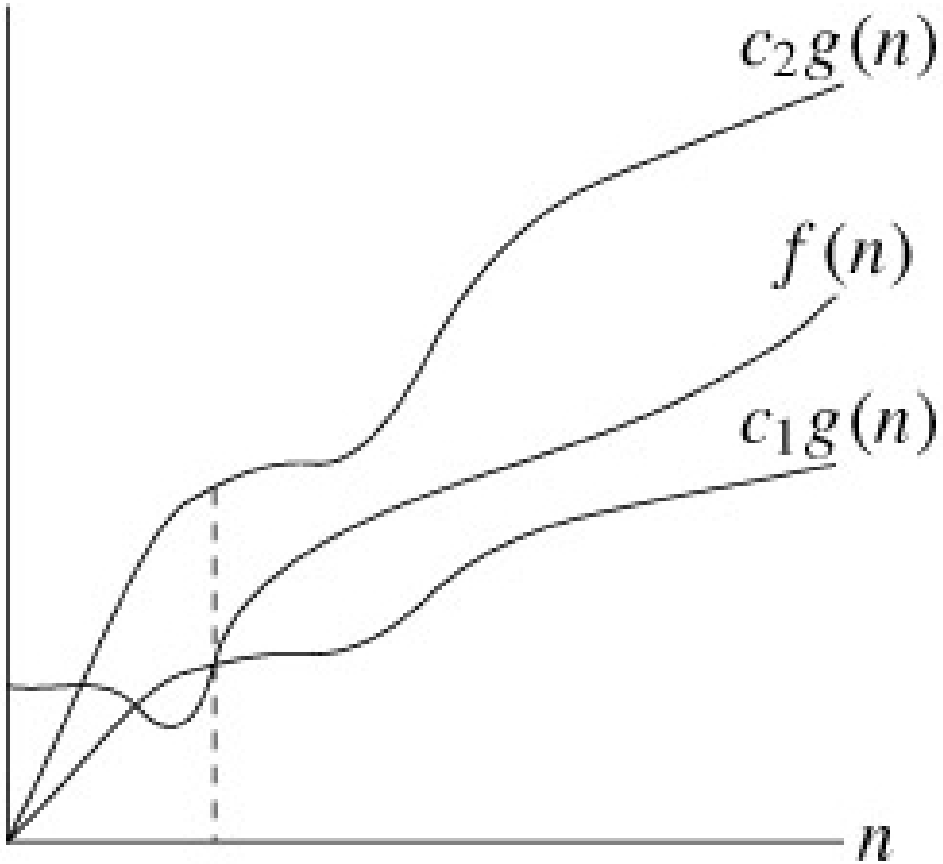
Notação Θ (limite assintótico restrito)

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

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

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



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

Notação Θ

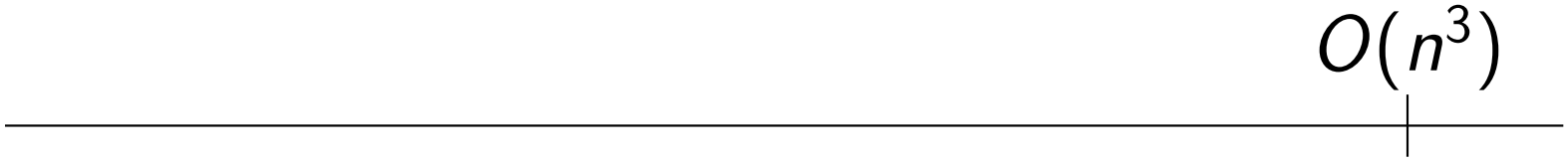
Para um polinômio $p(n) = \sum_{i=0}^{d} a_i n^i$, onde a_i são constantes e $a_d > 0$, temos que $p(n) = \Theta(n^d)$. Como uma constante pode ser considerada como um polinômio de grau 0, então dizemos que uma constante é $\Theta(n^0)$, ou seja, $\Theta(1)$.

Obs: escrever $f(n) = \Theta(g(n))$ é um abuso de notação o ideal seria escrever $f(n) \in \Theta(g(n))$ (o mesmo para O e Ω).

Sumário

Cota superior ou limite superior (upper bound)

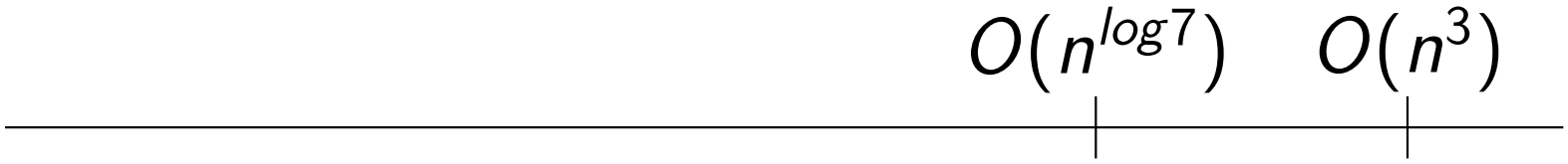
- Suponha um problema, por exemplo, multiplicação de duas matrizes quadradas $n \times n$ com n elementos.
- Conhecemos um algoritmo para resolver este problema (pelo método trivial) de complexidade $O(n^3)$.
- Sabemos assim que a complexidade deste problema não deve superar $O(n^3)$, uma vez que existe um algoritmo que o resolve com esta complexidade.
- Uma cota superior ou limite superior (upper bound) deste problema é $O(n^3)$.



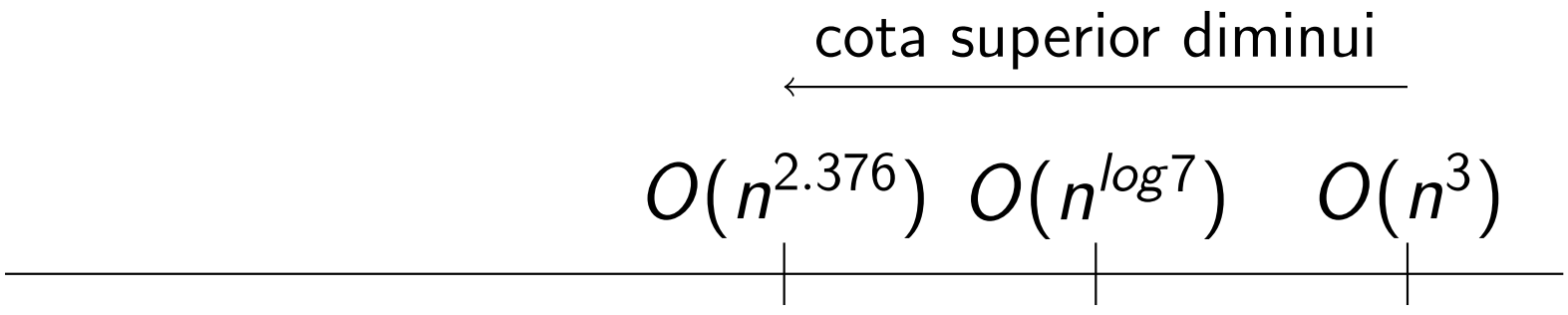
- No entanto, a cota superior de um problema pode mudar se alguém descobrir um outro algoritmo melhor.

Cota superior ou limite superior (upper bound)

- O algoritmo de Strassen reduziu a complexidade para $O(n^{log7})$. Assim a cota superior do problema de multiplicação de matrizes passou a ser $O(n^{log7})$.



- Coppersmith e Winograd melhoraram ainda para $O(n^{2.376})$



- Note que a cota superior de um problema depende do algoritmo. Pode diminuir quando aparece um algoritmo melhor.

Analogia com record mundial

A cota superior para resolver um problema é análoga ao record mundial de atletismo. Ele é estabelecido pelo melhor atleta (algoritmo) do momento. Como no record, a cota superior pode ser melhorada por um algoritmo (atleta) mais veloz.
"Cota superior" da corrida de 100 metros rasos:

Ano	Atleta	Tempo
1988	Carl Lewis	9s92
1993	Linford Christie	9s87
1993	Carl Lewis	9s86
1994	Leroy Burrell	9s84
1996	Donovan Bailey	9s84
1999	Maurice Greene	9s79
2002	Tim Montgomery	9s78
2007	Asafa Powell	9s74
2008	Usain Bolt	9s72
2008	Usain Bolt	9s69
2009	Usain Bolt	9s58

Cota inferior ou limite inferior (lower bound) Ω

- As vezes é possível demonstrar que, para um dado problema, **qualquer** que seja o algoritmo a ser usado, o problema requer pelo menos um certo número de operações.
- Essa complexidade é chamada **cota inferior** (**lower bound**) do problema.
- Note que a cota inferior depende do problema mas não do particular algoritmo.

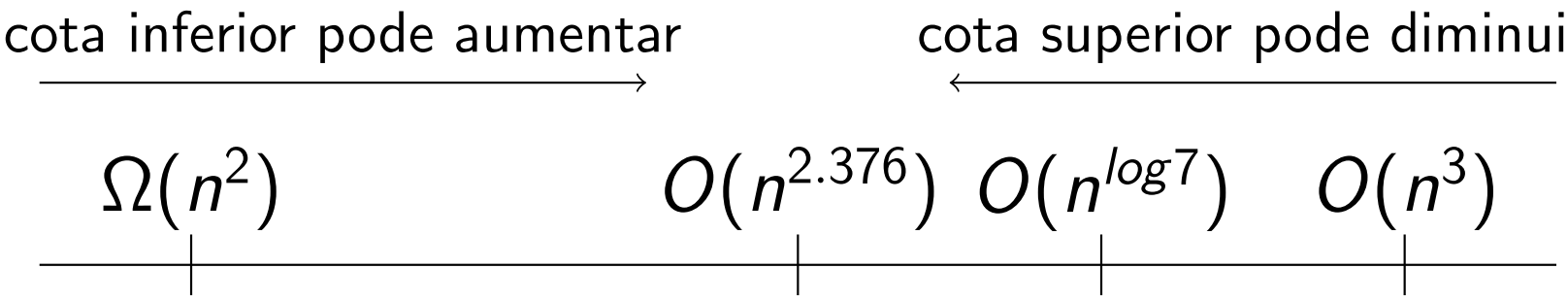
Cota inferior (lower bound) Ω

Para o problema de multiplicação de matrizes quadradas $n \times n$, apenas para ler os elementos das duas matrizes de entrada ou para produzir os elementos da matriz produto leva tempo n^2. Assim uma cota inferior trivial é $\Omega(n^2)$.

Cota inferior (lower bound) Ω

Na analogia anterior, uma cota inferior de uma modalidade de atletismo não dependeria mais do atleta. Seria algum tempo mínimo que a modalidade exige, qualquer que seja o atleta. Uma cota inferior trivial para os 100 metros rasos seria o tempo que a velocidade da luz leva para percorrer 100 metros no vácuo.

Cota inferior (lower bound)



- Se um algoritmo tem uma complexidade que é igual à cota inferior do problema, então ele é **assintoticamente ótimo**.

- Pesquisadores dedicam seu tempo tentando encurtar o intervalo ("gap") até encostar as duas cotas.

Conceitos relacionados

- Notação assintótica em equações e desigualdades:

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

- Propriedades: transitividade, reflexividade, simetria, etc;
- Notação o;
- Notação ω;