Notação
Projeto e Análise de Algoritmos

Daniel Capanema

Pontifícia Universidade Católica de Minas Gerais

2022

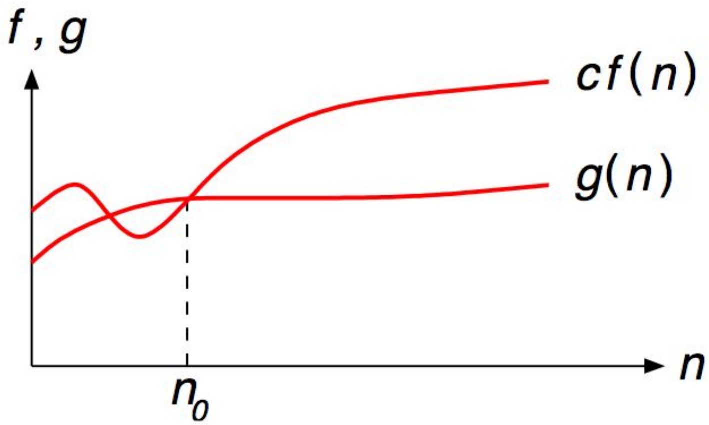
Comportamento Assintótico de Funções

- O parâmetro n fornece uma medida da dificuldade para se resolver o problema
- Para valores suficientemente pequenos de n, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes
- A escolha do algoritmo não é um problema crítico para problemas de tamanho pequeno
- Logo, a análise de algoritmos é realizada para valores grandes de n
- Estuda-se o comportamento assintótico das funções de custo (comportamento de suas funções de custo para valores grandes de n)
- Para entradas grandes o bastante, as constantes multiplicativas e os termos de mais baixa ordem de um tempo de execução podem ser ignorados

Dominação Assintótica

- 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 assintoticamente** outra função $g(n)$ se existem duas constantes positivas c e n_0 tais que, para $n \geq n_0$, temos

$$|g(n)| \leq c \times |f(n)|$$



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

Como Medir o Custo de Execução de um Algoritmo?

- **Função de Custo** ou **Função de Complexidade**
  - T(n) = medida de custo necessário para executar um algoritmo para um problema de tamanho n
  - Se T(n) é uma medida da quantidade de tempo necessário para executar um algoritmo para um problema de tamanho n, então T é chamada função de complexidade de tempo de algoritmo
  - Se T(n) é uma medida da quantidade de memória necessária para executar um algoritmo para um problema de tamanho n, então T é chamada função de complexidade de espaço de algoritmo

Observação: TEMPO NÃO É TEMPO!

É importante ressaltar que a complexidade de tempo na realidade não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.

Custo Assintótico de Funções

- É interessante comparar algoritmos para valores grandes de n
- O *custo assintótico* de uma função T(n) representa o limite do comportamento de custo quando n cresce
- Em geral, o custo aumenta com o tamanho n do problema

Observação:

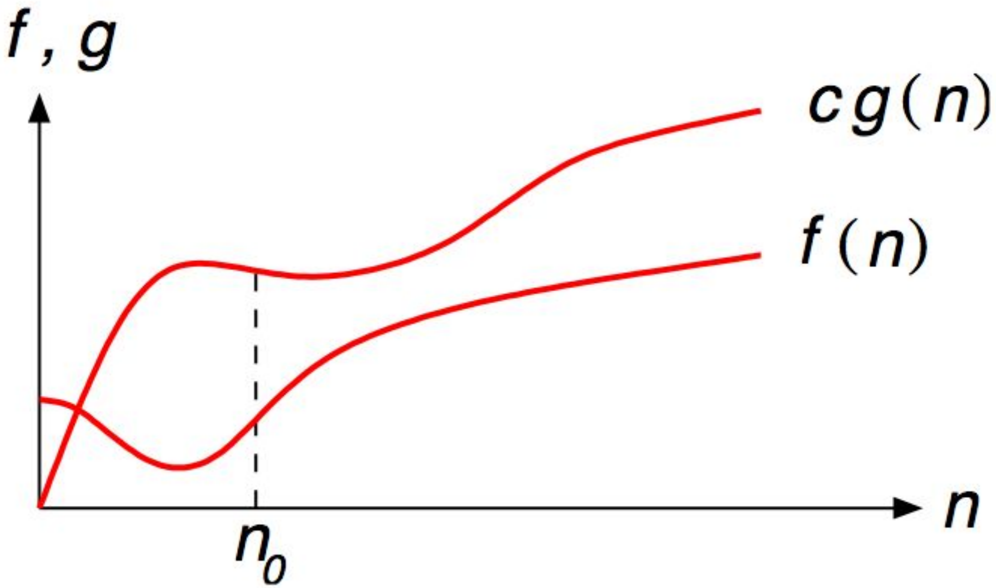
Para valores pequenos de n, mesmo um algoritmo ineficiente não custa muito para ser executado

Notação assintótica de funções

- Existem três notações principais na análise de assintótica de funções:
  - Notação O (“O” grande)
  - Notação Ω
  - Notação Θ

Notação O

- f(n) = O(g(n))



Notação O

- A notação O define um limite superior para a função, por um fator constante
- Escreve-se *f(n) = O(g(n))*, se existirem constantes positivas *c* e n_0 tais que para $n \geq n_0$, o valor de *f(n)* é menor ou igual a *cg(n)*.
  - Pode-se dizer que *g(n)* é um limite assintótico superior (em inglês, *asymptotically upper bound*) para *f(n)*

$$f(n) = \mathrm{O}(g(n)), \exists\, c > 0 \text{ e } n_0 \mid 0 \leq f(n) \leq cg(n), \forall n \geq n_0$$

- Escrevemos f(n) = O(g(n)) para expressar que g(n) domina assintoticamente f(n). Lê-se f(n) é da ordem no máximo g(n).

- Observe que a notação O define um conjunto de funções:

$$O(g(n)) = \{f : \aleph \rightarrow \Re^{+} \mid \exists c > 0, n_0, 0 \leq f(n) \leq cg(n),\ \ \forall n \geq n_0\}$$

Notação O: Exemplos

- Seja $f(n) = (n + 1)2$
  - Logo $f(n)$ é $O(n^2)$, quando no = 1 e c = 4, já que

$$(n+1)^2 \leq 4n^2 \quad \text{para} \quad n \geq 1$$

- Seja $f(n) = n$ e $g(n) = n^2$. Mostre que $g(n)$ não é $O(n)$.
  - Sabemos que $f(n)$ é $O(n^2)$, pois para $n \geq 0,\ n \leq n^2$.
  - Suponha que existam constantes c e no tais que para todo $n \geq n_0$, $n^2 \leq cn$.
  - Assim, $c \geq n$ para qualquer $n \geq n_0$.
  - No entanto, não existe uma constante *c* que possa ser maior ou igual a *n* para todo *n*.

Notação O: Exemplos

- Mostre que $g(n) = 3n^3 + 2n^2 + n$ é *O*(*n*³)
  - Basta mostrar que $3n^3 + 2n^2 + n \leq 6n^3$ para $n \geq 0$
  - A função $g(n) = 3n^3 + 2n^2 + n$ é também *O*(n^4), entretanto esta afirmação é mais fraca que dizer que *g*(*n*) é *O*(n^3)

- Mostre que $h(n) = \log_5 n$ é *O*(*log n*)
  - O $\log_b n$ difere do $\log_c n$ por uma constante que no caso é $\log_b c$
  - Como $n = c^{\log_c n}$, tomando o logaritmo base *b* em ambos os lados da igualdade, temos que $\log_b n = \log_b c^{\log_c n} = \log_c n \times \log_b c$

Notação O

- Quando a notação O é usada para expressar o tempo de execução de um algoritmo no pior caso, está se definindo também o limite superior do tempo de execução desse algoritmo para *todas* as entradas

- Por exemplo, o algoritmo de ordenação por inserção é $O(n^2)$ no pior caso
  - Este limite se *aplica* para *qualquer* entrada

Notação O

- Tecnicamente é um abuso dizer que o tempo de execução do algoritmo de ordenação por inserção é $O(n^2)$ (i.e., sem especificar se é para o pior caso, melhor caso, ou caso médio)
  - O tempo de execução desse algoritmo depende de como os dados de entrada estão arranjados.
  - Se os dados de entrada já estiverem ordenados, este algoritmo tem um tempo de execução de $O(n)$, ou seja, o tempo de execução do algoritmo de ordenação por inserção no *melhor caso* é $O(n)$.

- O que se quer dizer quando se fala que "o tempo de execução é $O(n^2)$" é que no pior caso o tempo de execução é $O(n^2)$
  - ou seja, não importa como os dados de entrada estão arranjados, o tempo de execução em qualquer entrada é $O(n^2)$

Operações com a notação O

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times O(f(n)) &= O(f(n)) \quad c = constante \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Operações com a notação O: Exemplos

- Regra da soma $O(f(n)) + O(g(n))$
  - Suponha três trechos cujos tempos de execução sejam

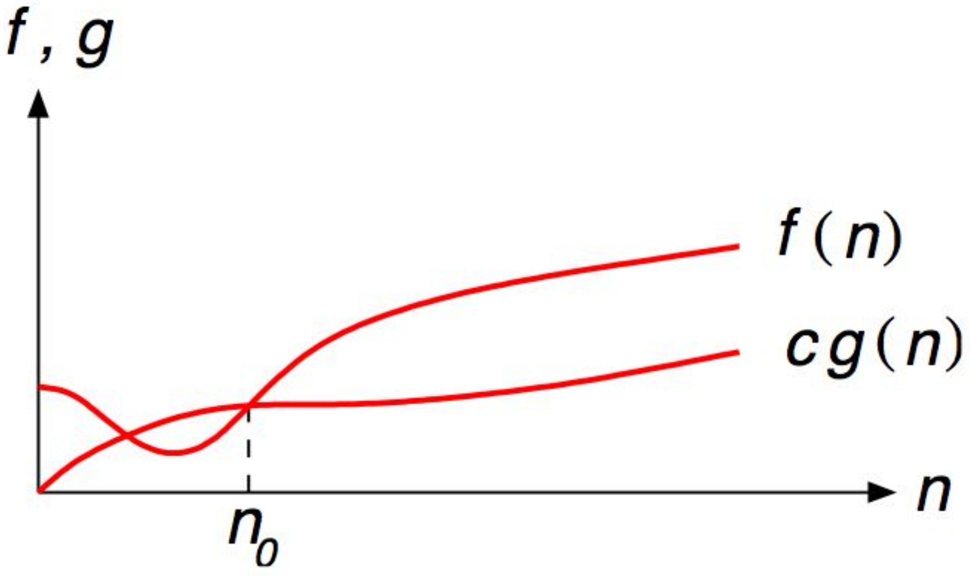
    $$O(n), O(n^2) \text{ e } O(n \log n)$$
  - O tempo de execução dos dois primeiros trechos é $O(\max(n, n^2))$, que é $O(n^2)$
  - O tempo de execução de todos os três trechos é então

$$O(\max(n^2, \log n))$$

que é $O(n^2)$

Notação Ω

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



Notação Ω

- A notação Ω define um limite inferior para a função, por um fator constante

- Escreve-se *f(n)* = *Ω(g(n))*, se existirem constantes positivas c e n0 tais que para *n* ≥ n^0, o valor de *f(n)* é maior ou igual a *cg(n)*
  - Pode-se dizer que *g(n)* é um limite assintótico inferior (em inglês, *asymptotically lower bound*) para *f(n)*

$$f(n) = \Omega(g(n)), \exists c > 0 \text{ e } n_0 \mid 0 \le cg(n) \le f(n), \forall n \ge n_0$$

- Observe que a notação Ω define um conjunto de funções:

$$\Omega(g(n)) = \{f : \aleph \rightarrow \Re^+ \mid \exists c > 0, n_0, 0 \le cg(n) \le f(n), \ \forall n \ge n_0\}$$

Notação Ω

- Quando a notação Ω é usada para expressar o tempo de execução de um algoritmo no melhor caso, está se definindo também o limite (inferior) do tempo de execução desse algoritmo para todas as entradas

- Por exemplo, o algoritmo de ordenação por inserção é Ω(n) no melhor caso
  - O tempo de execução do algoritmo de ordenação por inserção é Ω(n)

- O que significa dizer que "o tempo de execução" (i.e., sem especificar se é para o pior caso, melhor caso, ou caso médio) é Ω(g(n))?
  - O tempo de execução desse algoritmo é pelo menos uma constante vezes g(n) para valores suficientemente grandes de n

Notação Ω: Exemplos

- Para mostrar que $f(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$ basta fazer *c* = 1, e então $3n^3 + 2n^2 \geq n^3$ para $n \geq 0$

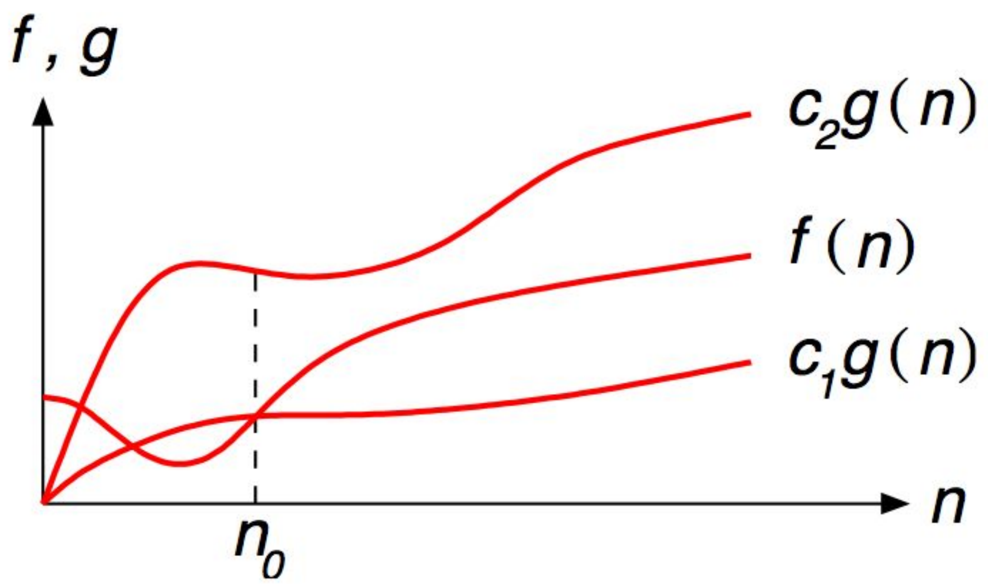
Notação Ω: Exemplos

- Para mostrar que $f(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$ basta fazer *c* = 1, e então $3n^3 + 2n^2 \geq n^3$ para $n \geq 0$

- Seja *f(n)* = *n* para *n* ímpar (*n* ≥ 1) e *f(n)* = $n^2/10$ para *n* par (*n* ≥ 0).
  - Neste caso *f(n)* é Ω (n^2), bastando considerar *c* = 1/10 e *n* = 0,2,4,6,..

Notação Θ

- f(n) = Θ(g(n))

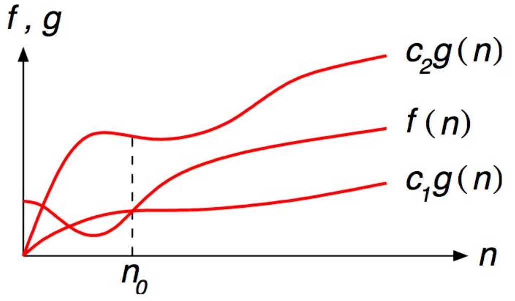


Notação Θ

- A notação Θ limita a função por fatores constantes
- Escreve-se f(n) = Θ(g(n)), se existirem constantes positivas c1, c2 e n0 tais que para n ≥ n0, o valor de f(n) está sempre entre c1g(n) e c2g(n) inclusive
- Neste caso, pode-se dizer que g(n) é um limite assintótico firme (em inglês, *asymptotically tight bound*) para f(n)

$$f(n) = \Theta(g(n)), \exists c_1 > 0, c_2 > 0 \text{ e } n_0 \mid$$

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n), \ \forall n \ge n_0$$



- Observe que a notação Θ define um conjunto de funções:

$$\Theta(g(n)) = \{ f : \aleph \to \Re^+ \mid \exists c_1 > 0, c_2 > 0, \ n_0, 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \ \forall n \ge n_0 \}$$

Notação Θ: Exemplo

- Mostre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

- Para provar esta afirmação, devemos achar constantes $c_1 > 0$, $c_2 > 0$, $n_0 > 0$, tais que:

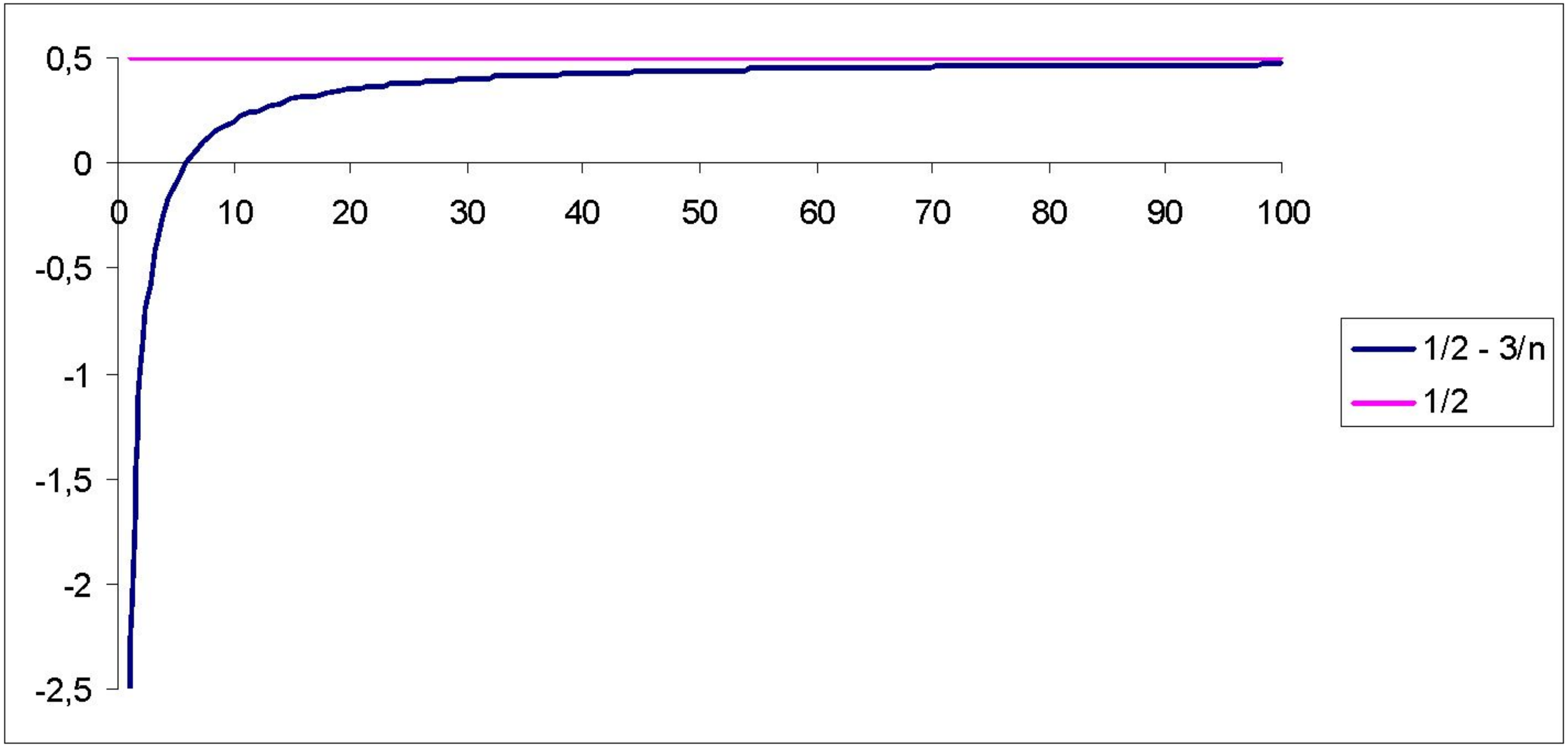
$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

- para todo $n \geq n_0$

- Se dividirmos a expressão acima por n^2 temos:

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

Notação Θ: Exemplo



Notação Θ: Exemplo

- A inequação mais a direita será sempre válida para qualquer valor de $n \geq 1$ ao escolhermos $c_2 \geq 1/2$
- Da mesma forma, a inequação mais a esquerda será sempre válida para qualquer valor de $n \geq 7$ ao escolhermos $c_1 \leq 1/14$
- Assim, ao escolhermos $c_1 = 1/14$, $c_2 = 1/2$ e $n_0 = 7$, podemos verificar que

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

- Note que existem outras escolhas para as constantes c_1 e c_2, mas o fato importante é que a escolha existe
- Note também que a escolha destas constantes depende da função $\frac{1}{2}n^2 - 3n$
- Uma função diferente pertencente a $\Theta(n^2)$ irá provavelmente requerer outras constantes

Exercício

- Usando a definição formal de Θ, prove que $6n^3 \neq \Theta(n^2)$.

Notações: Propriedades

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

- 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, $g(n) = \Omega(f(n))$.

- Transitividade:
  - Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, então $f(n) = O(h(n))$.
  - Se $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$, então $f(n) = \Omega(h(n))$.
  - Se $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n))$, então $f(n) = \Theta(h(n))$.

Notações: Relações Úteis

Quais as notações mais indicadas para expressar a complexidade de casos específicos de um algoritmo, do algoritmo de modo geral e da classe de algoritmos para o problema?

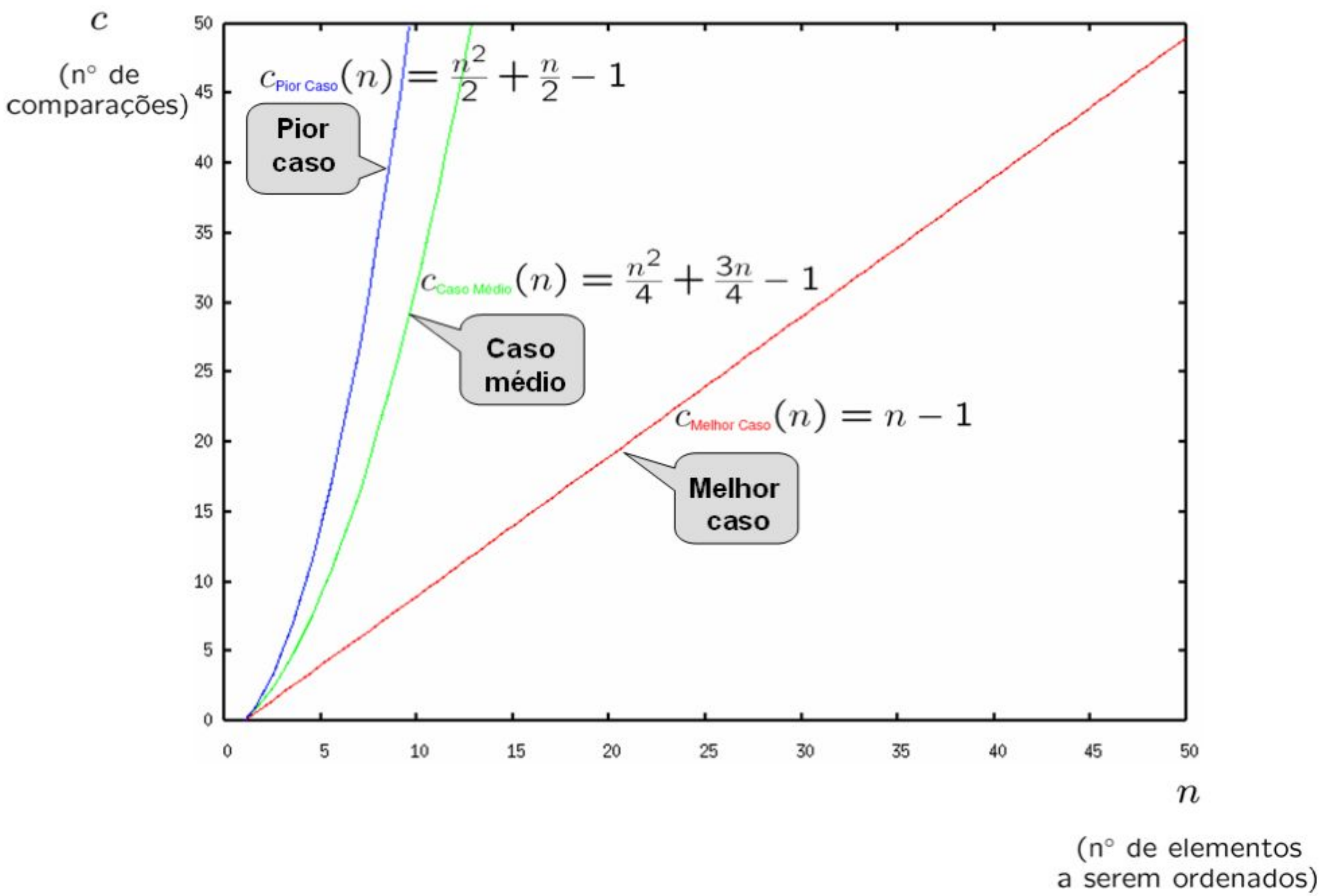
- **Casos específicos:**
  - o ideal é a notação **Θ**, por ser um limite assintótico firme.
  - A notação O também é aceitável e bastante comum na literatura.
  - Embora possa teoricamente ser usada, a notação Ω é mais fraca neste caso e deve ser evitada para casos específicos.
- **Algoritmo de forma geral:**
  - Se o algoritmo comporta-se de forma idêntica para qualquer entrada, a notação Θ é a mais precisa (lembre-se que $f(n)=\Theta(g(n)) \Rightarrow f(n)=O(g(n))$).
  - Se os casos melhor e pior são diferentes, a notação mais indicada é a O, já que estaremos interessados em um limite assintótico superior.
  - **O pior caso do algoritmo deve ser a base da análise.**
- **Para uma classe de algoritmos:**
  - Neste caso estamos interessados no limite inferior para o problema e a notação deve ser a Ω.

Limites do Algoritmo de Ordenação por Inserção

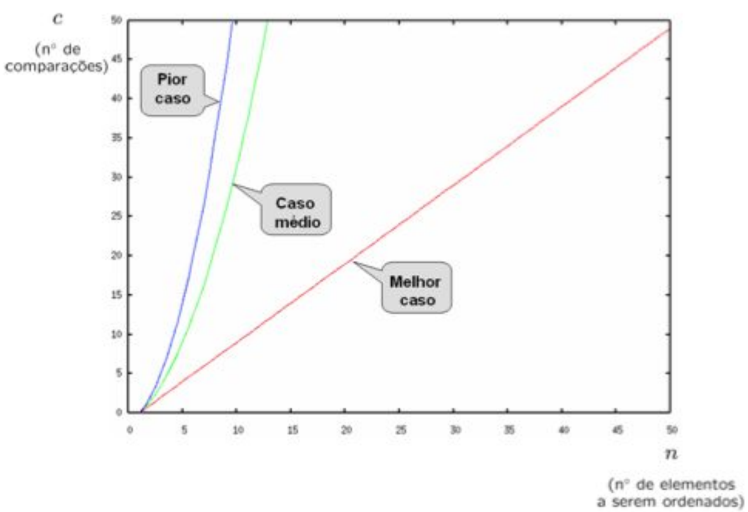
- O tempo de execução do algoritmo de ordenação por inserção está entre $\Omega(n)$ e $O(n^2)$

- Estes limites são assintoticamente os mais firmes possíveis
  - Por exemplo, o tempo de execução deste algoritmo não é $\Omega(n^2)$, pois o algoritmo executa em tempo $\Theta(n)$ quando a entrada já está ordenada

Funções de Custo (nº de comparações): Algoritmo de Ordenação por Inserção



Funções de Custo e Notações Assintóticas: Algoritmo de Ordenação por Inserção



Pior Caso:

$$c_{\text{Pior Caso}}(n) = \frac{n^2}{2} + \frac{n}{2} - 1 = \begin{matrix} \boxed{O} \\ \Theta \\ \Omega \end{matrix} \left(n^2 \right)$$

Caso Médio:

$$c_{\text{Caso Médio}}(n) = \frac{n^2}{4} + \frac{3n}{4} - 1 = \begin{matrix} O \\ \Theta \\ \Omega \end{matrix} \left(n^2 \right)$$

Melhor caso:

$$c_{\text{Melhor Caso}}(n) = n - 1 = \begin{matrix} O \\ \Theta \\ \boxed{\Omega} \end{matrix} \left(n \right)$$

indica a notação normalmente usada para esse caso.

Teorema

- Para quaisquer funções $f(n)$ e $g(n)$,

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

se e somente se, $f(n) = O(g(n))$, e

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

Mais sobre notação assintótica de funções

- Existem duas outras notações na análise assintótica de funções:
  - Notação o (“O” pequeno)
  - Notação ω
- Estas duas notações não são usadas normalmente, mas é importante saber seus conceitos e diferenças em relação às notações O e Ω, respectivamente

Notação o

- O limite assintótico superior definido pela notação O pode ser assintoticamente firme ou não
  - Por exemplo, o limite $2n^2 = O(n^2)$ é assintoticamente firme, mas o limite não é $2n = O(n^2)$

- A notação *o* é usada para definir um limite superior que não é assintoticamente firme

- Formalmente a notação *o* é definida como:

$$f(n) = o(g(n)), \text{ para qualquer } c > 0 \; e \; n_0 \mid 0 \leq f(n) < cg(n), \forall n \geq n_0$$

- Exemplo, $2n = o(n^2)$ mas $2n^2 \neq o(n^2)$

Notação o

- As definições das notações O e o são similares
  - A diferença principal é que em $f(n) = o(g(n))$, a expressão $0 \leq f(n) < cg(n)$ é válida para todas constantes $c > 0$

- Intuitivamente, a função $f(n)$ tem um crescimento muito menor que $g(n)$ quando n tende para infinito.
  - Isto pode ser expresso da seguinte forma:

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

- Alguns autores usam este limite como a definição de o

Notação ω

- Por analogia, a notação ω está relacionada com a notação Ω da mesma forma que a notação o está relacionada com a notação O

- Formalmente a notação ω é definida como:

$$f(n) = \omega(g(n)), \text{ para qualquer } c > 0 \ e \ n_0 \mid 0 \le cg(n) < f(n), \forall n \ge n_0$$

- Por exemplo, $\frac{n^2}{2} = \omega(n)$, mas $\frac{n^2}{2} \neq \omega(n^2)$

- A relação $f(n) = \omega(g(n))$ implica em:

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

se o limite existir!

Exercícios

1. Prove que $2^{n+1} = O(2^n)$.

Exercícios

1. Prove que $2^{n+1} = O(2^n)$.

$2^{n+1} = O(2^n)$, mas $2^{2n} \neq O(2^n)$

Para mostrar que $2^{n+1} = O(2^n)$, precisamos achar constantes c, $n_0 > 0$ tais que $0 <= 2^{n+1} <= c.2^n$ para todo $n >= n_0$

Já que $2^{n+1} = 2.2^n$ para todo n, podemos satisfazer a definição com $c = 2$ e $n_0 = 1$.

Exercícios

e para: $2^{2^n} = O(2^n)$.

Exercícios

Para mostrar que 2^{2n} != $O(2^n)$, assuma que exista constantes c, n_0 > 0 tais que 0 <= 2^{2n} <= $c.2^n$ para todo n >= n_0.

Considere que $2^{2n} = 2^n.2^n$ <= $c.2^n$ □ 2^n <= c.

Mas nenhuma constante é maior que 2^n.

Então essa consideração nos leva a uma contradição.