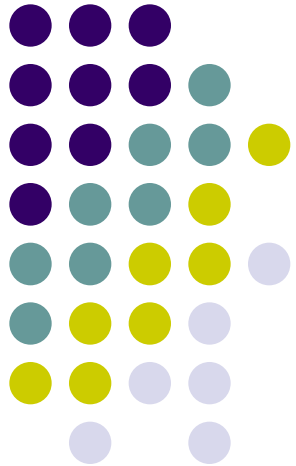
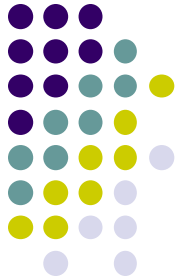
Análise de algoritmos

Introdução à Ciência da Computação 2

Baseado nos slides do Prof. Thiago A. S. Pardo



Algoritmo



- Noção geral: conjunto de instruções que devem ser seguidas para solucionar um determinado problema

- Cormen et al. (2002)

  - Qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores de **entrada** e produz algum valor ou conjunto de valores de **saída**

  - Ferramenta para resolver um problema computacional bem especificado

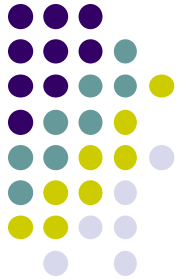
  - Assim como o hardware de um computador, constitui uma tecnologia, pois o desempenho total do sistema depende da escolha de um algoritmo eficiente tanto quanto da escolha de um hardware rápido

Algoritmo

- Comen et al. (2002)
  - Deseja-se que um algoritmo termine e seja correto

- Perguntas
  - Mas um algoritmo correto vai terminar, não vai?
  -

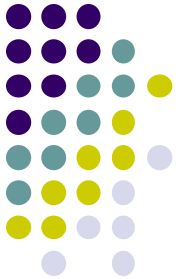
Exemplo



- Versão recursiva vs. iterativa do algoritmo de Fibonacci
  - Estimativa de tempo (Brassard e Bradley, 1996)

n	***10***	***20***	***30***	***50***	***100***
Recursão	8 ms	1 s	2 min	21 dias	10^9 anos
Iteração	1/6 ms	1/3 ms	1/2 ms	3/4 ms	1,5 ms

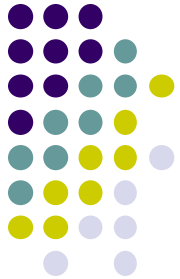
Recursos de um algoritmo



- Uma vez que um algoritmo está pronto/disponível, é importante determinar os recursos necessários para sua execução
  - Tempo
  - Memória

- Qual o principal quesito? Por que?

Análise de algoritmos

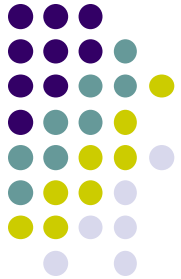


- Um algoritmo que soluciona um determinado problema, mas requer o processamento de um ano, não deve ser usado

- O que dizer de uma afirmação como a abaixo?
  - "Desenvolvi um novo algoritmo chamado TripleX que leva 14,2 segundos para processar 1.000 números, enquanto o método SimpleX leva 42,1 segundos"

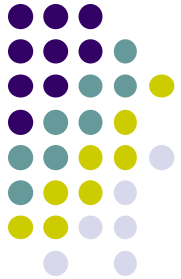
    - Você trocaria o SimpleX que roda em sua empresa pelo TripleX?

Análise de algoritmos



- A afirmação tem que ser examinada, pois há diversos fatores envolvidos
  - Características da máquina em que o algoritmo foi testado
    - Quantidade de memória
  - Linguagem de programação
  - Implementação pouco cuidadosa do algoritmo SimpleX vs. "super" implementação do algoritmo TripleX
  - Quantidade de dados processados
    - Se o TripleX é mais rápido para processar 1.000 números, ele também é mais rápido para processar quantidades maiores de números, certo?

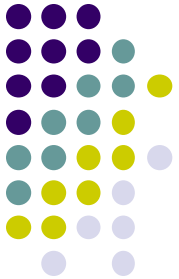
Análise de algoritmos



- A comunidade de computação começou a pesquisar formas de comparar algoritmos de forma independente de
  - Hardware
  - Linguagem de programação
  - Habilidade do programador

- Portanto, deseja-se comparar algoritmos e não programas
  - Área conhecida como "análise/complexidade de algoritmos"

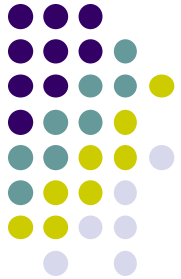
Eficiência de algoritmos



- Sabe-se que
  - Processar 10.000 números leva mais tempo do que 1000 números
  - Cadastrar 10 pessoas em um sistema leva mais tempo do que cadastrar 5
  - Etc.

- Então, pode ser uma boa ideia estimar a eficiência de um algoritmo em função do tamanho do problema
  - Em geral, assume-se que "n" é o tamanho do problema, ou número de elementos que serão processados
  - E calcula-se o número de operações que serão realizadas sobre os n elementos

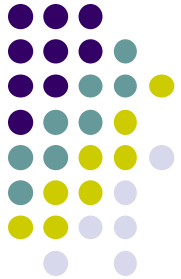
Eficiência de algoritmos



- O melhor algoritmo é aquele que requer menos operações sobre a entrada, pois é o mais rápido
  - O tempo de execução do algoritmo pode variar em diferentes máquinas, mas o número de operações é uma boa medida de desempenho de um algoritmo

- De que operações estamos falando?

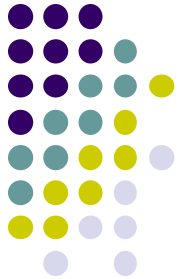
Exemplo: TripleX vs. SimpleX



- TripleX: para uma entrada de tamanho n, o algoritmo realiza n^2+n operações
  - Pensando em termos de função: $f(n)=n^2+n$

- SimpleX: para uma entrada de tamanho n, o algoritmo realiza 1.000n operações
  - $g(n)=1.000n$

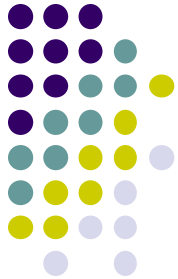
Exemplo: TripleX vs. SimpleX



- Faça os cálculos do desempenho de cada algoritmo para cada tamanho de entrada

Tamanho da entrada (n)	1	10	100	1.000	10.000
$f(n)=n^2+n$					
$g(n)=1.000n$					

Exemplo: TripleX vs. SimpleX

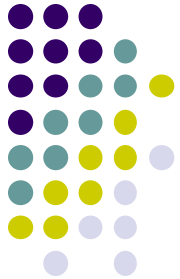


- Faça os cálculos do desempenho de cada algoritmo para cada tamanho de entrada

Tamanho da entrada (n)	1	10	100	1.000	10.000
$f(n)=n^2+n$	2	110	10.100	1.001.000	100.010.000
g(n)=1.000n	1.000	10.000	100.000	1.000.000	10.000.000

- A partir de n=1.000, f(n) mantém-se maior e cada vez mais distante de g(n)
  - Diz-se que f(n) cresce mais rápido do que g(n)

Análise assintótica

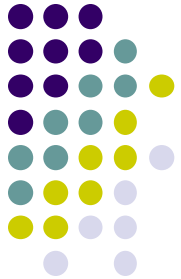


- Deve-se preocupar com a eficiência de algoritmos quando o **tamanho de n for grande**

- Definição: *a eficiência assintótica de um algoritmo descreve a eficiência relativa dele quando n torna-se grande*
  - *Pode não valer para quando a entrada for pequena*

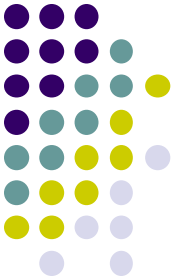
- Portanto, para comparar 2 algoritmos, determinam-se as taxas de crescimento de cada um: o algoritmo com menor taxa de crescimento rodará mais rápido quando o tamanho do problema for grande

Análise assintótica



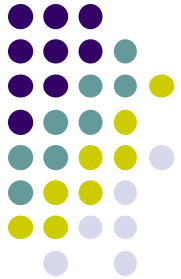
- Atenção
  - Algumas funções podem não crescer com o valor de n
    - Quais? Exemplo de um problema?
  - Também se pode aplicar os conceitos de análise assintótica para a quantidade de memória usada por um algoritmo
    - Mas não é tão útil, pois é difícil estimar os detalhes exatos do uso de memória e o impacto disso

Relembrando um pouco de matemática...



- Expoentes
  - $x^a x^b = x^{a+b}$
  - $x^a / x^b = x^{a-b}$
  - $(x^a)^b = x^{ab}$
  - $x^n + x^n = 2x^n$ (diferente de x^{2n})
  - $2^n + 2^n = 2^{n+1}$

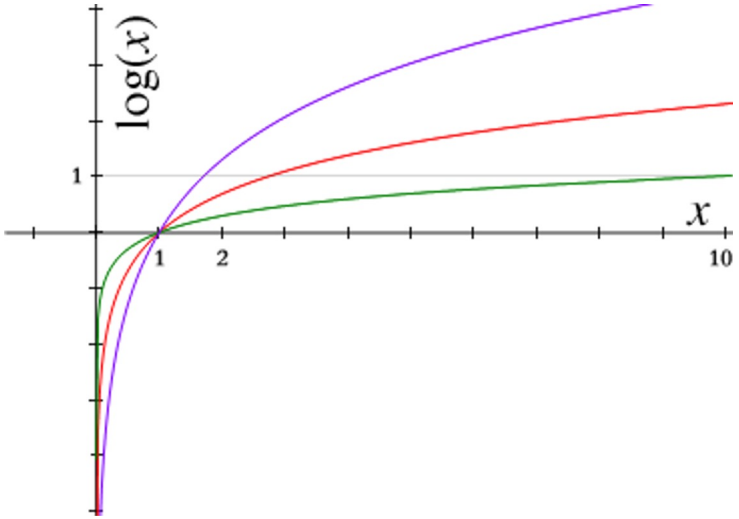
Relembrando um pouco de matemática...



- Logaritmos (usaremos a base 2, a menos que seja dito o contrário)
  - $x^a=b \Leftrightarrow \log_x b=a$
  - $\log_a b = \log_c b / \log_c a$, se $c>0$
  - $\log ab = \log a + \log b$
  - $\log a/b = \log a - \log b$
  - $\log(a^b) = b \log a$

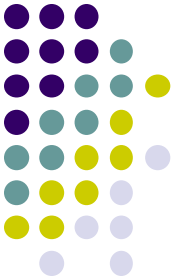
Relembrando um pouco de matemática...

- Logaritmos (usaremos a base 2, a menos que seja dito o contrário)
  - E o mais importante
    - log x < x para todo x>0
  - Alguns valores
    - log 1=0, log 2=1,
      log 1.024=10,
      log 1.048.576=20



Exemplo para várias bases

Relembrando um pouco de matemática...



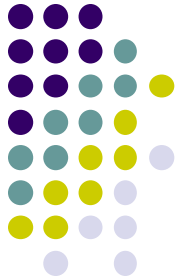
- Séries

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

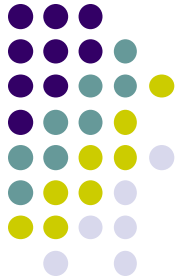
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$$

Algumas notações



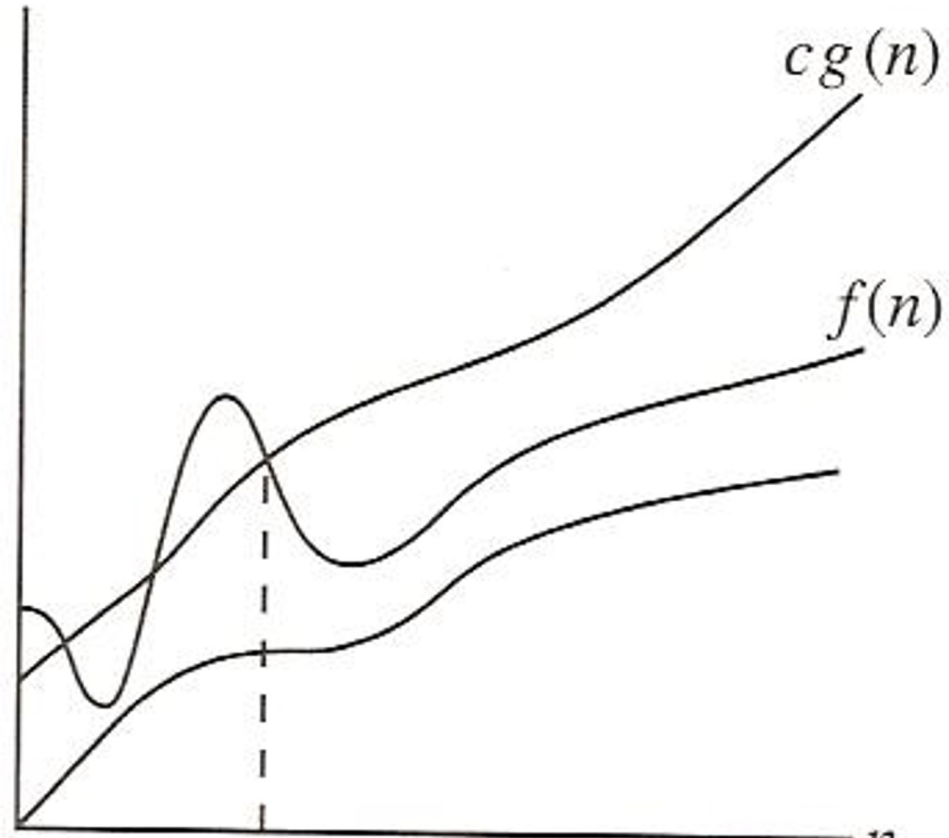
- Notações que usaremos na análise de algoritmos
  - T(n) = O(f(n)) (lê-se *big-oh*, *big-o* ou "da ordem de") se existirem constantes c e n_0 tal que T(n) ≤ c * f(n) quando n ≥ n_0
    - A taxa de crescimento de T(n) é menor ou igual à taxa de f(n)
  - T(n) = Ω(f(n)) (lê-se "ômega") se existirem constantes c e n_0 tal que T(n) ≥ c * f(n) quando n ≥ n_0
    - A taxa de crescimento de T(n) é maior ou igual à taxa de f(n)

Algumas notações



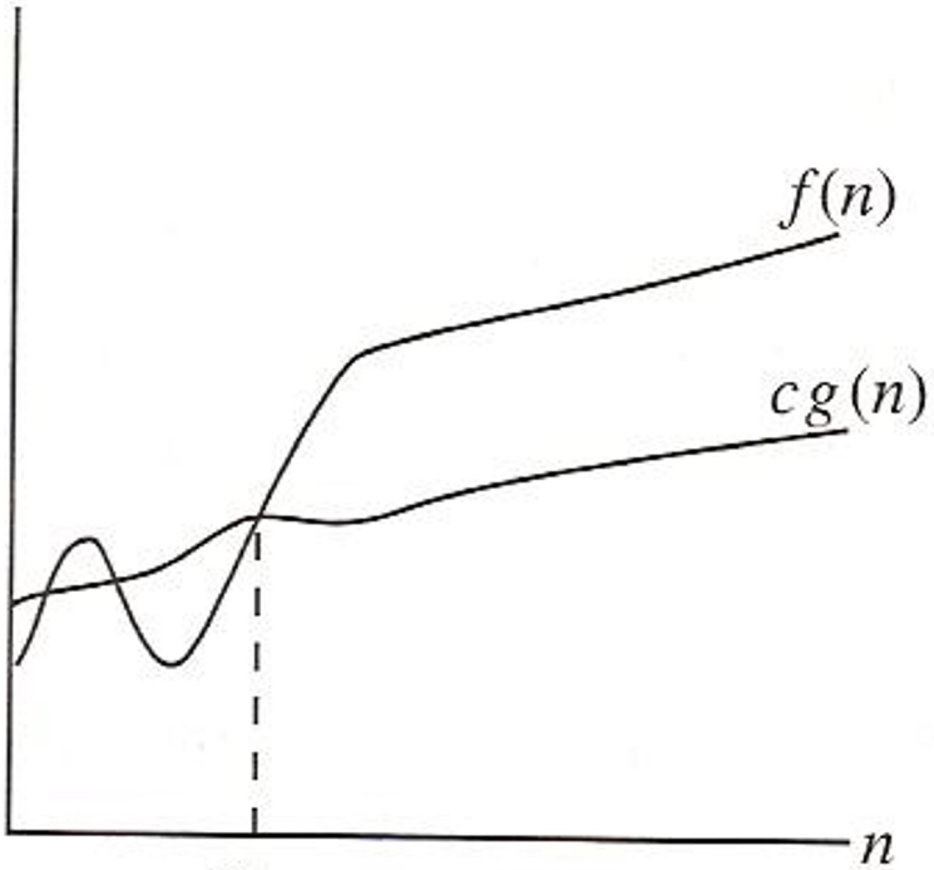
- Notações que usaremos na análise de algoritmos
  - $T(n) = \Theta(f(n))$ (lê-se "theta") se e somente se $T(n) = O(f(n))$ e $T(n) = \Omega(f(n))$
    - A taxa de crescimento de T(n) é igual à taxa de f(n)
  - $T(n) = o(f(n))$ (lê-se *little-oh* ou *little-o*) se e somente se $T(n) = O(f(n))$ e $T(n) \neq \Theta(f(n))$
    - A taxa de crescimento de T(n) é menor do que a taxa de f(n)

Algumas notações



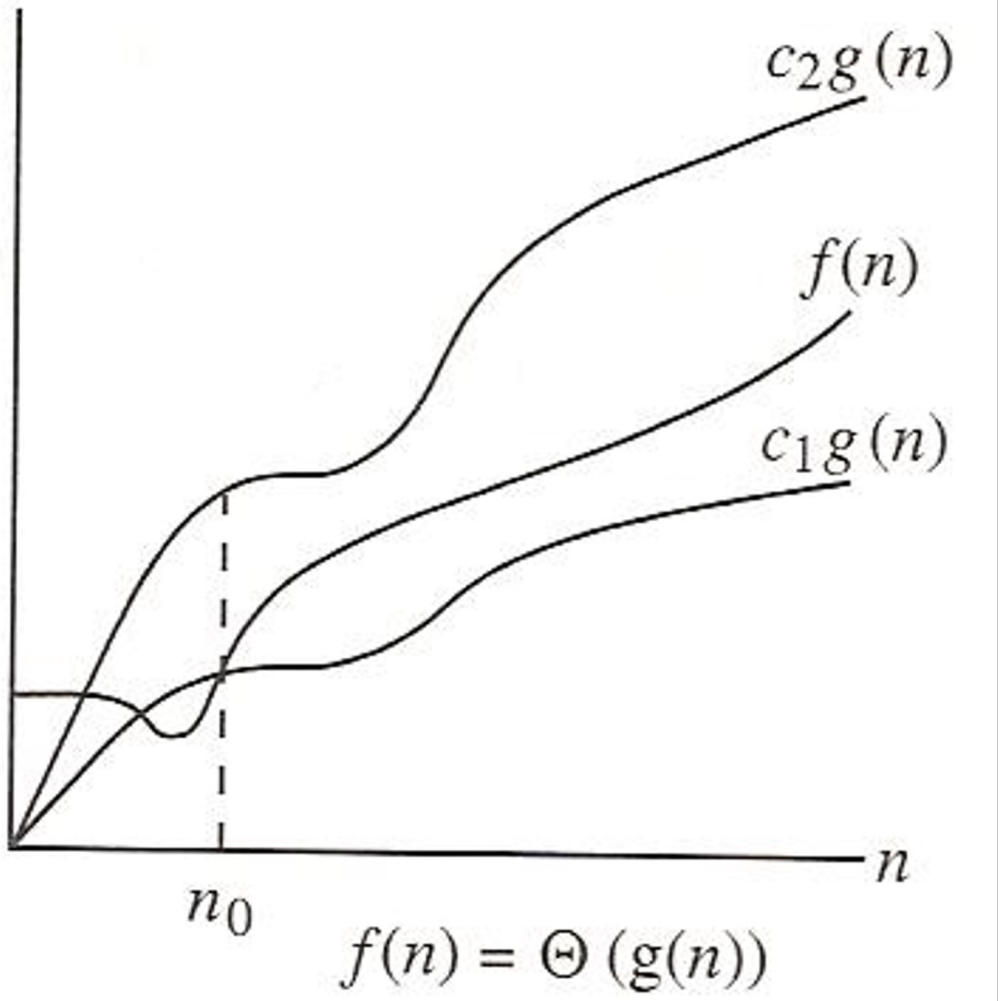
$f(n) = O(g(n))$

Algumas notações

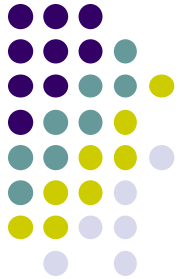


$f(n) = \Omega(g(n))$

Algumas notações

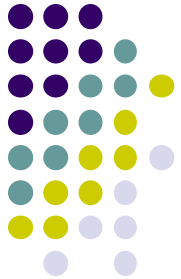


Algumas notações



- 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)

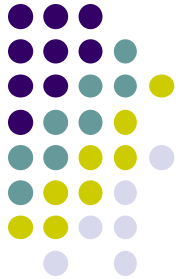
Mais algumas considerações



- Ao dizer que $T(n) = O(f(n))$, garante-se que T(n) cresce numa taxa não maior do que f(n), ou seja, f(n) é seu limite superior

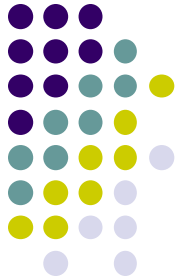
- Ao dizer que $f(n) = \Omega(T(n))$, tem-se que T(n) é o limite inferior de f(n)

Exercícios conceituais



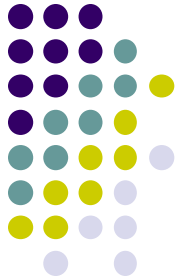
- Suponha que um dos algoritmo de ordenação mais utilizados, possui no melhor caso O(n). Podemos dizer que nenhum outro algoritmo poderá atingir uma complexidade melhor do que esta?

Exercícios conceituais



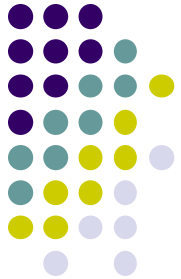
- Suponha que um dos algoritmo de ordenação mais utilizados, possui no melhor caso O(n). Podemos dizer que nenhum outro algoritmo poderá atingir uma complexidade melhor do que esta?
  - Para resolver o problema é necessária ler todos os valores. Portanto, qualquer algoritmo será no mínimo Ω(n).

Exercícios conceituais



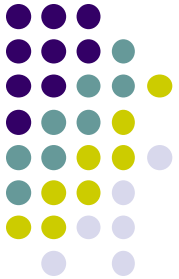
- Dois algoritmos A e B possuem complexidade n^2 e 200n, respectivamente. Você utilizaria o algoritmo A ao invés do B?

Exercícios conceituais



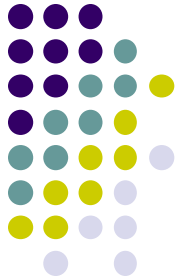
- Dois algoritmos A e B possuem complexidade n^2 e 200n, respectivamente. Você utilizaria o algoritmo A ao invés do B?
  - Sim, para entrada pequenas.

Para pensar (em casa)



- Considere o problema de buscar um elemento em um conjunto ordenada do dados: $a_1 < a_2 < ... < a_n$
  - Mostre que o algoritmo é $\Omega(\log n)$

Exercício

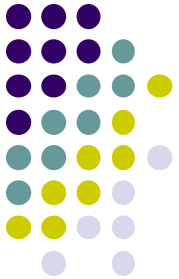


- Mostrar que
  - $n^2 = O(n^3)$
  - $n^3 = \Omega(n^2)$

- Se $f(n)=n^2$ e $g(n)=2n^2$, então essas duas funções têm taxas de crescimento iguais
  - Provar que $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Exercício

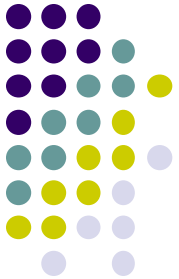
- Provar que $6n^3 \neq \Theta(n^2)$



Exercício

- Provar que $f(n) = \frac{1}{2} n^2 - 3n = \Theta(n^2)$
  - Intuitivamente os termos de menor grau são desconsiderados na análise assintótica

Exercício



- Seja f(x) = O(g(x)) e g(x) = O(h(x)).
  - Mostre que f(x) = O(h(x))