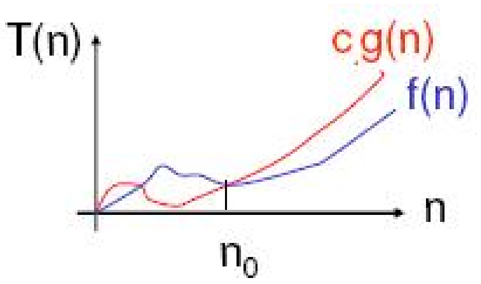
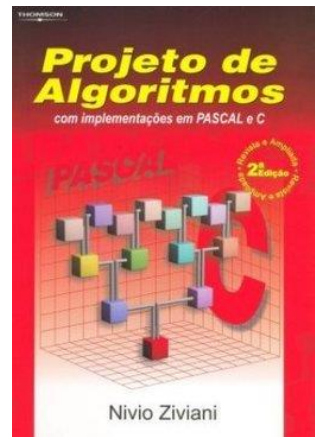
Unidade 2 - Crescimento Assintótico de Funções

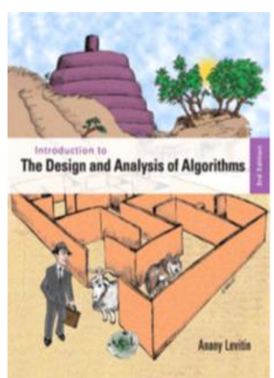


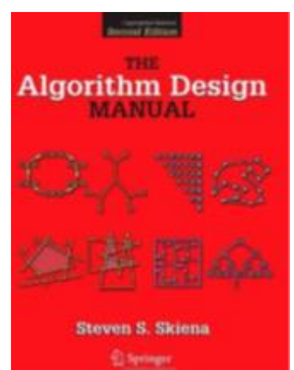
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

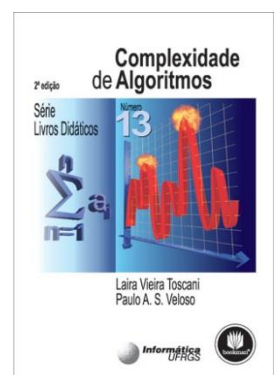
Bibliografia

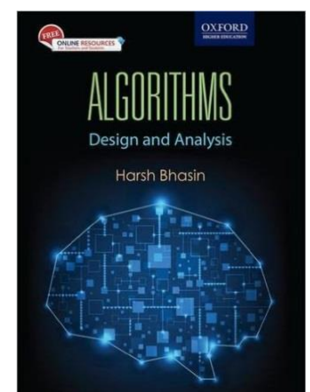
- Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002
- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS
- Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press – 2015
- Projeto de Algoritmos – Nivio Ziviani – Pioneira Informática - 1993









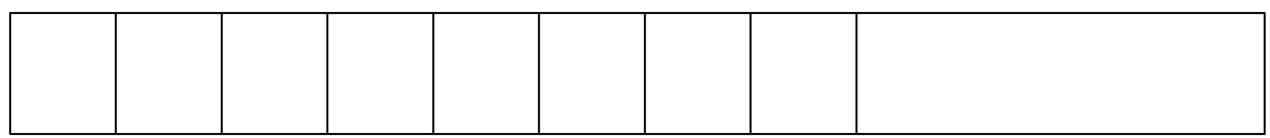


Introdução

- Para a execução de uma tarefa computacional, talvez o mais importante seja projetar um **algoritmo correto**.
- Um algoritmo é dito correto se ele atende à especificação da **tarefa requerida**;
- Entretanto, a despeito de ser correto, um algoritmo pode ter execução **impraticável**.
- Por exemplo, a pesquisa linear em um array é correta, mas **impraticável** se o array tiver $\mathbf{10^{10}}$ elementos.

Exemplo

- Pesquisa Linear em um array com $\mathbf{10^{10}}$ elementos;
- Tempo para se processar um elemento do array: $\mathbf{10^{-6}}$ segundos;
- No pior caso, serão necessários 10.000 segundos ou cerca de 3 horas para se buscar o elemento arbitrário no array.



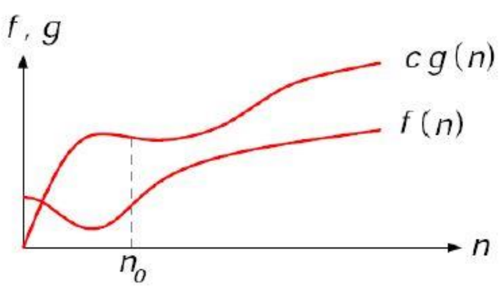
n = 10^{10} elementos

No pior caso, 3 horas de processamento!

Assim, algoritmos devem ser corretos mas também eficientes . . .

Introdução

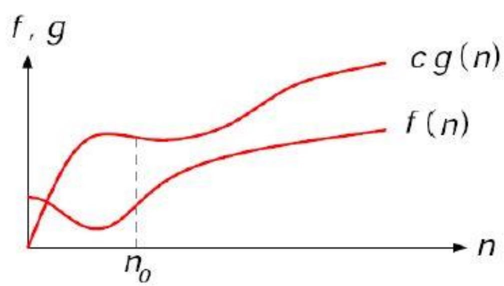
- É difícil determinar-se - de forma exata - o tempo de execução de um algoritmo!
- Em geral, cada passo em um pseudocódigo e cada **statement** em uma implementação HLL corresponde a um pequeno número de **operações primitivas** que não depende do tamanho da entrada;
- Assim, pode-se executar uma análise simplificada que estima o número de operações primitivas, por meio da contagem dessas operações;
- Felizmente, há uma notação que permite caracterizar os principais fatores que afetam o desempenho de um algoritmo, sem levar em conta os detalhes dessas operações primitivas.



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

Análise Assintótica

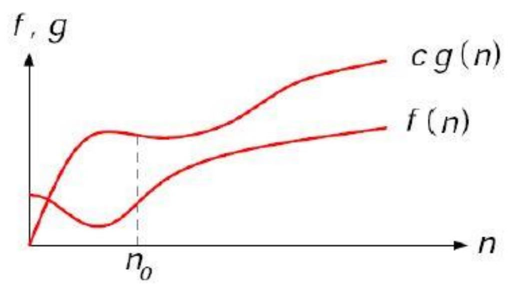
- Ao se deparar com uma Função de Complexidade definida por **F(n) = n+10** ou au **F(n) n^2 + 1**, geralmente pensa-se em valores não muito grandes de **n**, ou ainda valores próximos de zero;
- Na Análise de Algoritmos, por outro lado, atua-se de forma exatamente ao contrário;
- Ignora-se valores pequenos de **n** e foca-se em valores grandes (suficientemente grandes) de **n**;
- Esse tipo de Análise é denominada Análise Assintótica.



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

Análise Assintótica - Exemplo

- Consideremos o número de operações de dois Algoritmos que resolvem um mesmo problema, em função de **n**, onde **n** corresponde ao tamanho da entrada.
- **Algoritmo A**: **F1(n) = $2n^2$ + 50** operações
- **Algoritmo B**: **F2(n) = 500 n + 4000** operações
- Dependendo do valor de n, o **Algoritmo A** pode requerer mais ou menos operações que o **Algoritmo B**.

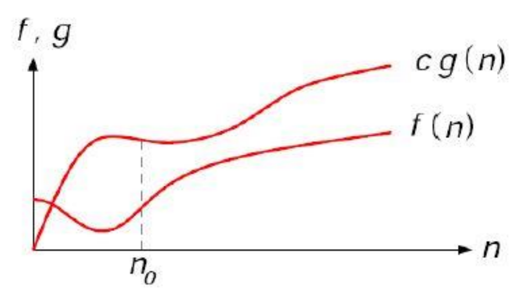


$$f(n) = O(g(n))$$

Análise Assintótica - Exemplo

- **Algoritmo A**: $\mathbf{F_1(n) = 2n^2 + 50}$ operações
  - ✓ Para n=10 => Serão necessárias $10.10^2 + 50 = 1.050$ operações
  - ✓ Para n= 100 => Serão necessárias $10.100^2 = 100.000 + 500 = 100.500$ operações

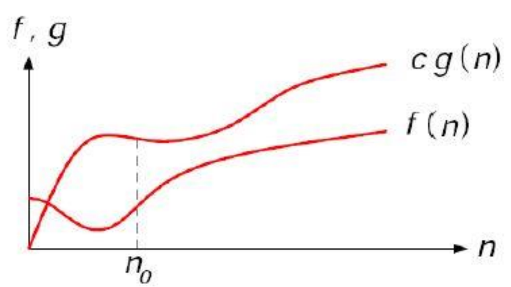
- **Algoritmo B**: $\mathbf{F_2(n) = 500\,n + 4000}$ operações
  - ✓ Para n=10 => Serão necessárias $500.10 + 4.000 = 9.000$ operações
  - ✓ Para n= 100 => Serão necessárias $500.100 + 4000 = 50.000 + 500 = 50.500$ operações



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

Análise Assintótica

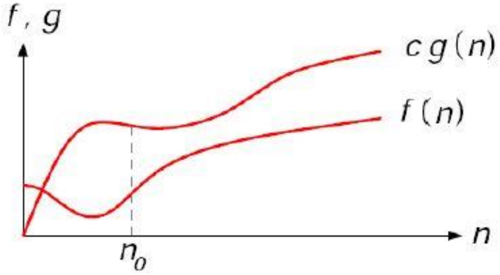
- Assim, o importante é observar-se que **F_1(n)** tem crescimento proporcional a n^2 (quadrático);
- Ao passo que **F_2(n)** tem crescimento proporcional a n (Linear);
- Um crescimento quadrático é **PIOR** que um crescimento linear;
- Portanto, na comparação das Funções **F_1** e **F_2**, deve-se preferir o **Algoritmo B**.



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

Análise Assintótica

- ✓ Considerando que é muito difícil levantar-se a quantidade exata de operações executadas por um algoritmo, em Análise de Algoritmos concentra-se, portanto, no comportamento assintótico das funções de complexidade, ou seja, deve-se observar a taxa de crescimento da função quando n é suficientemente grande;
- ✓ Em geral, os termos inferiores e as constantes multiplicativas pouco contribuem na análise e podem, dessa forma, serem descartadas.



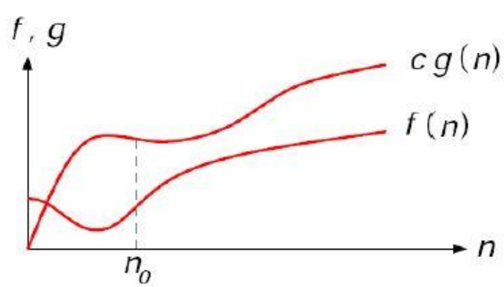
$$f(n) = O(g(n))$$

Análise Assintótica

- ✓ Para valores suficientemente grandes de n, as funções:

  n^2 , $7/2\ n^2$, $555555n^2$, $n^2/8888$, $7n^2 + 300n + 4$

- ✓ Crescem todas com a mesma velocidade e, portanto, do ponto de vista assintótico, são "equivalentes";
- ✓ Na Área de Análise de Algoritmos, as funções de Complexidade são classificadas em "ordens";
- ✓ Todas as funções de uma mesma ordem são "equivalentes";
- ✓ As cinco funções acima pertencem, portanto, à mesma ordem (quadráticas);

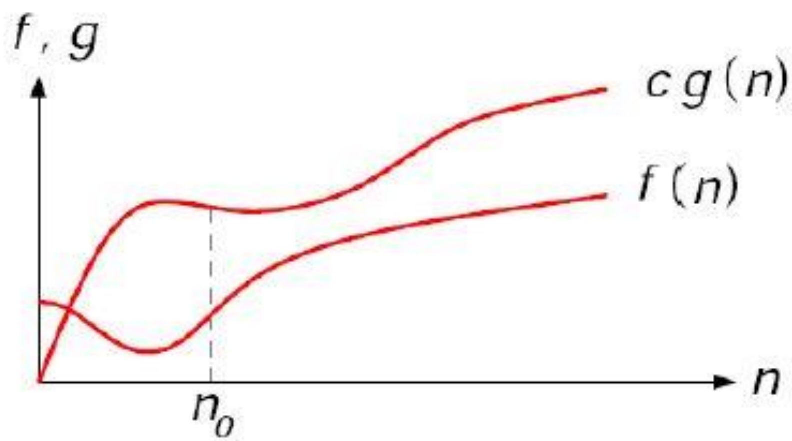


$$f(n) = O(g(n))$$

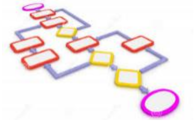
Funções Assintoticamente Não Negativas

- ✓ Na **Análise de Algoritmos**, restringem-se o estudo para funções assintoticamente não-negativas, ou seja, uma função **f** tal que **f(n) >= 0**, para todo **n** suficientemente grande;

- ✓ Mais explicitamente, **f** é assintoticamente não-negativa se existe um n_0 tal que **f(n) >= 0**, para todo n > n_0.

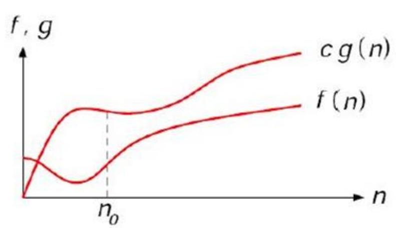


$$f(n) = O(g(n))$$

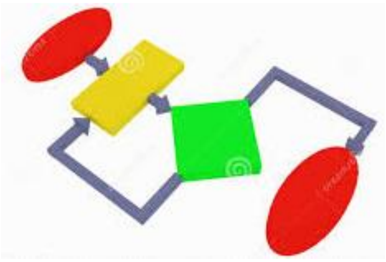


Ordem de grandeza de execução

- Por exemplo, o tempo exato de execução de um algoritmo pode ser dado pela função polinomial $f(n) = 3n^2 + 2n + 3$.
- Neste caso, o tempo aproximado de execução será uma função de n^2, ou seja $f(n^2)$. (mais alta potência de n)
- Dessa forma, pode-se desprezar o coeficiente de n^2, bem como os outros termos da função polinomial que define a complexidade do algoritmo;
- Assim, para efeito de análise de algoritmos, utiliza-se uma notação que seja capaz de exprimir a ordem de grandeza do tempo de execução.
- Essa notação é assintótica, ou seja, representa uma linha que se aproxima da função de complexidade do algoritmo.

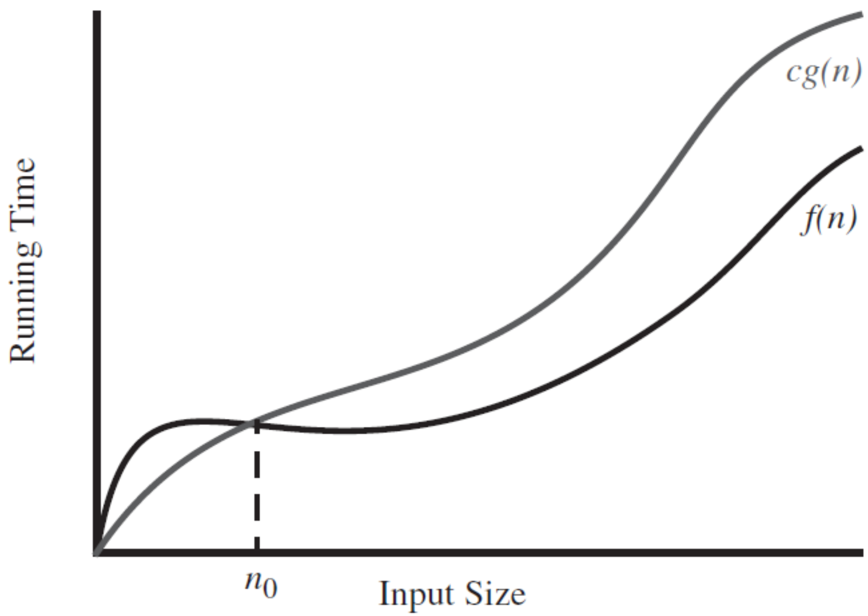


$$f(n) = O(g(n))$$



A notação Big-Oh

- Seja f(n) e g(n) funções que mapeiam inteiros não negativos para números reais;
- Diz-se que f(n) é O(g(n)) se existir uma constante real $c > 0$ e uma constante inteira $n_o \geq 1$ tal que $f(n) \leq cg(n)$ para todo inteiro $n \geq n_0$;
- Essa definição é frequentemente dita "**f(n) é big-Oh de g(n)**" ou **"f(n) é ordem g(n)"**.



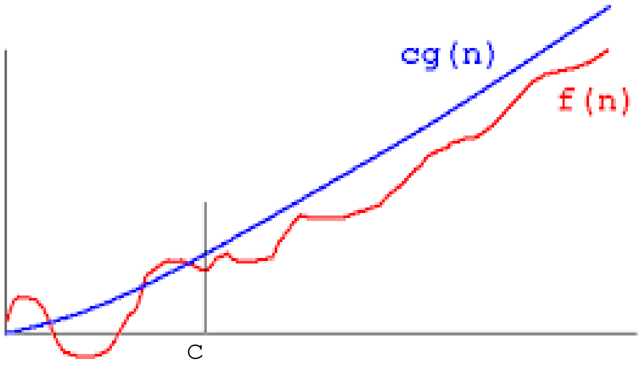
The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Ordem de Complexidade O(n)

- A notação **big-Oh** permite que se diga que uma função de n é “menor ou igual” a outra função, por um fator constante (c na definição);

- A notação **big-Oh** é largamente empregada para caracterizar limites de tempo e de espaço do algoritmo em termos de um parâmetro, n, o qual representa o tamanho do problema;

- A notação **big-Oh** fornece **limites superiores** de funções que, por sua vez, correspondem ao tempo de execução de algoritmos.



A função de complexidade $F(n) = 3n + 8$ é $O(n)$?

Ordem de Complexidade
F(n) = 3n + 8

- Necessita-se de uma constante real **c > 0** e uma constante inteira $\mathbf{n_o} \geq$ **1** tal que **3n + 8 ≤ c n** para todo inteiro $\mathbf{n} \geq \mathbf{n_0}$.

- Majorando-se a função F(n) , tem-se:

Constante c

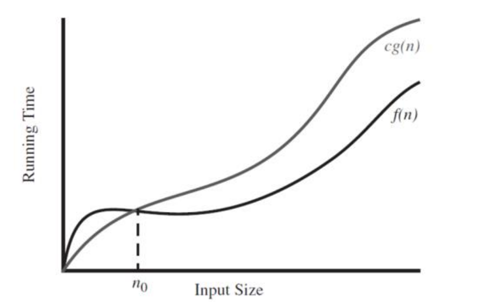
- F(n) = 3n + 8 ≤ **3n + 8n**

- **Portanto, F(n) = 3n + 8n ≤ 11.n**

- A expressão acima é verdadeira para todo n > 0;

- Logo, existe **c = 11**, tal que F(n) = 3n + 8 **≤ 11.n, para todo n > 0;**

- **Assim, 3n+8 é O(n)**



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

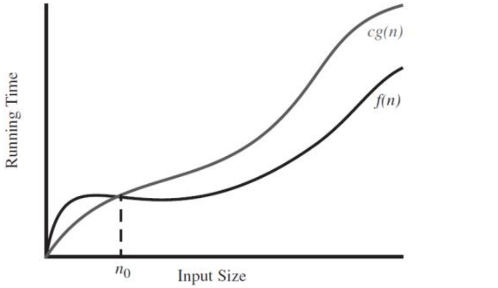
A função de complexidade $F(n) = 3n + 8$ é $O(n^2)$?

Ordem de Complexidade
$F(n) = 3n + 8$

- Necessita-se de uma constante real $\mathbf{c > 0}$ e uma constante inteira $\mathbf{n_o \geq 1}$ tal que $\mathbf{3n + 8 \leq c\, n^2}$ para todo inteiro $\mathbf{n \geq n_0}$.
- Majorando-se a função F(n) , tem-se:
- $F(n) = 3n + 8 \leq \mathbf{3n^2 + 8n^2}$

Constante c

- **Portanto,** $\mathbf{F(n) = 3n + 8 \leq 11.n^2}$
- A expressão acima é verdadeira para todo $n > 0$;
- Logo, existe **c = 11**, tal que $F(n) = 3n + 8 \leq \mathbf{11.n^2}$**, para todo** $\mathbf{n > 0}$**;**
- **Assim, 3n+8 é** $\mathbf{O(n^2)}$



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

A função de complexidade $F(n) = 3n + 8$ é $O(n^3)$?

Ordem de Complexidade
F(n) = 3n + 8

- Necessita-se de uma constante real **c > 0** e uma constante inteira $\mathbf{n_o \geq 1}$ tal que $\mathbf{3n + 8 \leq c\, n^3}$ para todo inteiro $\mathbf{n \geq n_0}$.

- Majorando-se a função F(n) , tem-se:

Constante c

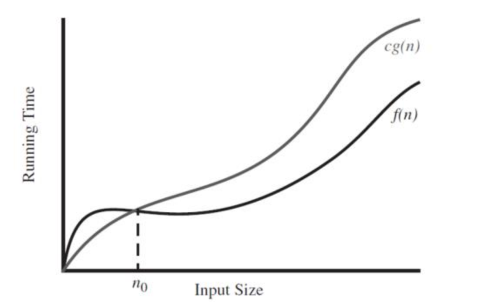
- $F(n) = 3n + 8 \leq \mathbf{3n^3 + 8n^3}$

- **Portanto,** $\mathbf{F(n) = 3n + 8 \leq 11.n^3}$

- A expressão acima é verdadeira para todo n > 0;

- Logo, existe **c = 11**, tal que $F(n) = 3n + 8$ $\mathbf{\leq 11.n^3}$**, para todo n > 0;**

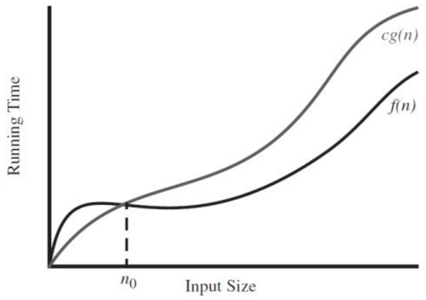
- **Assim, 3n+8 é** $\mathbf{O(n^3)}$



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Observação

- $3n + 8$ é $O(\mathbf{n})$.
- $3n + 8$ é $O(\mathbf{n^2})$.
- $3n + 8$ é $O(\mathbf{n^3})$.
- Portanto, 3n+8 ε a um conjunto de funções que atendem à definição de O(f(n)) ;



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

A função de complexidade $F(n) = 2n^2 + 3n + 4$ é $O(n^2)$?

Ordem de Complexidade
$F(n) = 2n^2 + 3n + 4$ é $O(n^2)$?

- Necessita-se de uma constante real **c > 0** e uma constante inteira **$n_o \geq$ 1** tal que **$2n^2 + 3n + 4 \leq c\, n^2$** para todo inteiro **$n \geq n_0$** .

- Majorando-se a função **F(n)** , tem-se:

- $F(n) = \mathbf{2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2}$

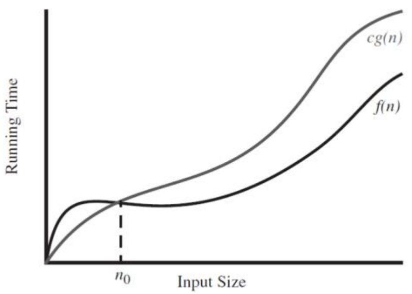
Constante c

- **Portanto, $F(n) = 2n^2 + 3n + 4 \leq 9.n^2$**

- A expressão acima é verdadeira para todo $n > 0$;

- Logo, existe **c = 9**, tal que $F(n) =$ **$2n^2 + 3n + 4 \leq 9.n^2$, para todo $n > 0$;**

- **Assim, $2n^2 + 3n + 4$ é $O(n^2)$**



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

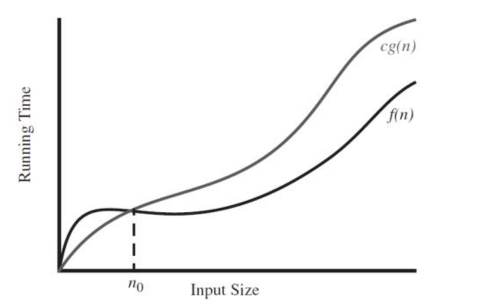
A função de complexidade $F(n) = 2n^2 + 3n + 4$ é $O(n^3)$?

Ordem de Complexidade
$F(n) = 2n^2 + 3n + 4$ é $O(n^3)$?

- Necessita-se de uma constante real **c > 0** e uma constante inteira $\mathbf{n_0 \geq 1}$ tal que $\mathbf{2n^2 + 3n + 4 \leq c\, n^3}$ para todo inteiro $\mathbf{n \geq n_0}$.
- Majorando-se a função **F(n)** , tem-se:
- $F(n) = \mathbf{2n^2 + 3n + 4 \leq 2n^3 + 3n^3 + 4n^3}$
- **Portanto,** $\mathbf{F(n) = 2n^2 + 3n + 4 \leq 9.n^3}$

Constante c

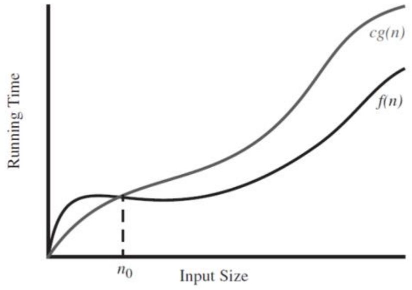
- A expressão acima é verdadeira para todo $n > 0$;
- Logo, existe **c = 9**, tal que $F(n) = \mathbf{2n^2 + 3n + 4 \leq 9.n^3}$ **, para todo n > 0;**
- **Assim,** $\mathbf{2n^2 + 3n + 4}$ **é** $\mathbf{O(n^3)}$



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Conclusão

- **$2n^2 + 3n + 4$ é $O(n^2)$**
- **$2n^2 + 3n + 4$ é $O(n^3)$**
- Porém, por razões de ordem prática, prefere-se dizer que **$2n^2 + 3n + 4$ é $O(n^2)$**

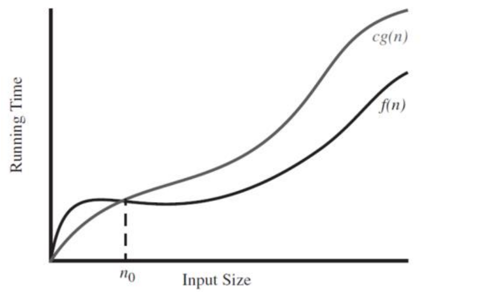


The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

A função de complexidade $F(n) = 2n^2 - 3n - 4$ é $O(n^2)$?

Ordem de Complexidade
$F(n) = 2n^2 - 3n - 4$ é $O(n^2)$?

- Necessita-se de uma constante real **$c > 0$** e uma constante inteira **$n_o \geq 1$** tal que **$2n^2 - 3n - 4 \leq c.n^2$** para todo inteiro **$n \geq n_0$** .
- **$F(n) = 2n^2 \cancel{- 3n} \cancel{- 4} \leq 2n^2$** (pode-se desprezar -3n e -4)
- **$F(n) = 2n^2 - 3n - 4 <= 2n^2$**
- Logo, existe uma constante c=2 (c>0), tal que $2n^2 - 3n - 4 <= 2n^2$
- **Assim,** $2n^2 - 3n - 4$ é **$O(n^2)$**

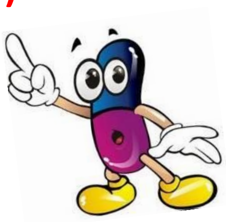


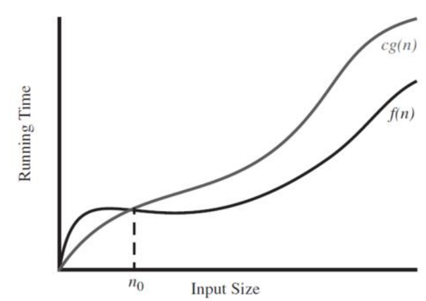
The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

A função de complexidade $F(n) = 8n^2 + 5n - 1000$ é $O(n^2)$?

Ordem de Complexidade
$F(n) = 8n^2 + 5n - 1000$ é $O(n^2)$?

- Necessita-se de uma constante real **$c > 0$** e uma constante inteira **$n_o \geq 1$** tal que **$8n^2 + 5n - 1000 \leq c.n^2$** para todo inteiro **$n \geq n_0$** .
- **$F(n) = 8n^2 + 5n - \cancel{1000} \leq 8n^2 + 5n$** (pode-se desprezar 1000)
- **$F(n) = 8n^2 + 5n - 1000 <= 8n^2 + 5n^2$** (majorando-se o termo 5n)
- **$F(n) = 8n^2 + 5n - 1000 <= 13n^2$**
- Logo, existe uma constante c=13 (c>0), tal que $8n^2 + 5n - 1000 <= 13n^2$
- **Assim,** $8n^2 + 5n - 1000$ é **$O(n^2)$**





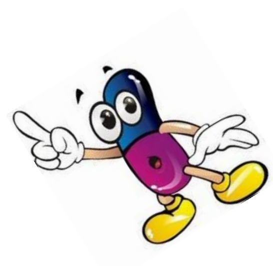
The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

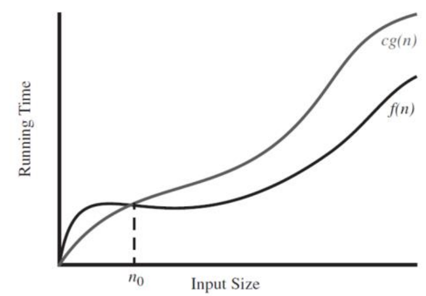
A função de complexidade $F(n) = 2n^2 - 3n + 4$ é $O(n)$?

QualitSys

Ordem de Complexidade
$F(n) = 2n^2 - 3n + 4$ é $O(n)$?

- Necessita-se de uma constante real **$c > 0$** e uma constante inteira **$n_o \geq 1$** tal que **$2n^2 - 3n + 4 \leq c\,n$** para todo inteiro **$n \geq n_0$** .
- **$2n^2 - 3n + 4 \leq cn$**
- **$2n^2 + 4 \leq cn + 3n$**
- **$2n^2 + 4 \leq n.(c + 3)$** **(c+3) também é uma constante k>0**
- **$2n^2 + 4 \leq n.k$**
- **$2n^2 + 4 \leq n.k$** **=> ABSURDO!**
- n é muito grande e, portanto, o primeiro termo da inequação nunca será inferior ao segundo termo, para qualquer **k>0** e $n \geq$ **n_0 , sendo $n_o > 1$**
- **Logo,** $2n^2 - 3n + 4$ NÃO é $O(n)$!

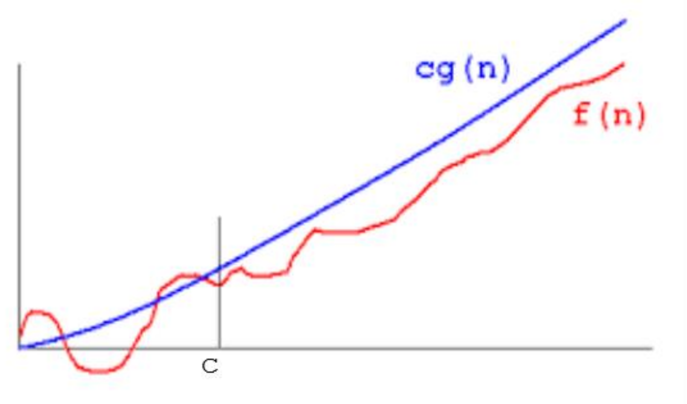




The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Notação Big Oh - Observações

- A ordem de complexidade $O(n)$ é melhor que $O(n^2)$ ou $O(n^3)$;
- Embora seja verdade dizer que $f(n) = 4n^3 + 3n^{4/3}$ seja $O(n^5)$, é mais informativo e prático dizer que seja $O(n^3)$;
- Algumas funções frequentemente aparecem na análise de Algoritmos, tais como: $O(\log n)$, $O(n)$, $O(n^2)$, $O(n^3)$, $O(n^k)$ $(K \geq 1)$ e $O(n^a)$ $(a > 1)$

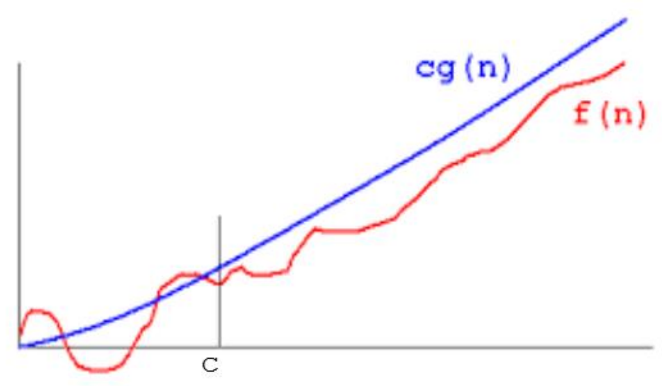


Notação Big Oh – Mais Observações

- É desaconselhável dizer que f(n) ≤ g(n), uma vez que o conceito de Big-Oh já denota a desigualdade “menor ou igual”;

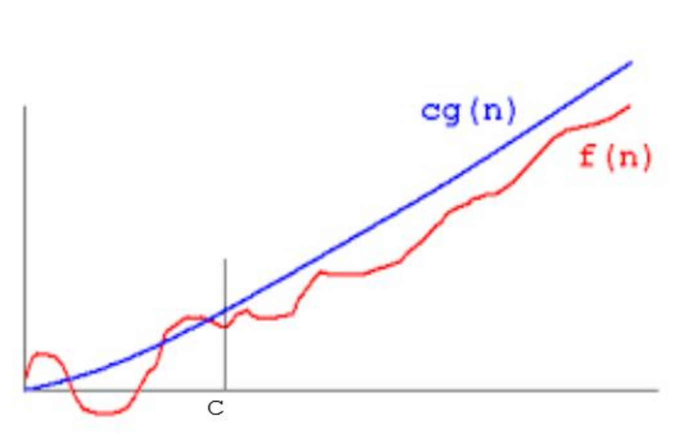
- Assim, embora comumente usado, não é completamente correto dizer-se que f(n) = g(n);

- É melhor dizer-se que f(n) ϵ g(n), uma vez que BigOh denota uma coleção de funções.



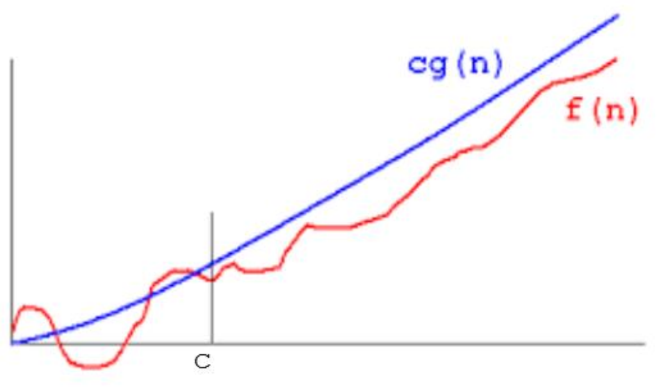
Limite Superior

- A notação **O(n)** é utilizada para indicar Limites Superiores para Problemas;
- Dado um problema, por exemplo, o de multiplicação de duas matrizes quadradas de ordem n ($n_x n$);
- Conhece-se um algoritmo para se resolver este problema (pelo método trivial) de complexidade **$O(n^3)$**.



Limite Superior

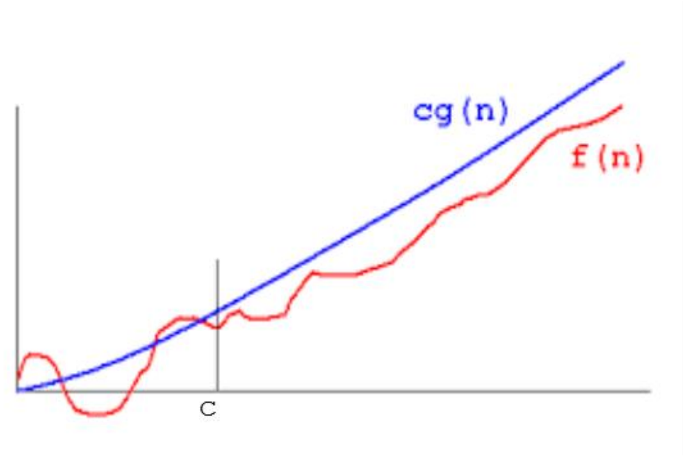
- Assim, sabe-se que a ordem de complexidade deste problema (multiplicação de matrizes quadradas de ordem n) não deve superar **$O(n^3)$,** uma vez que existe um algoritmo que o resolve com esta complexidade;
- Portanto, diz-se que uma COTA SUPERIOR ou LIMITE SUPERIOR para este problema é **$O(n^3)$;**
- A cota superior de um problema pode mudar se alguém descobrir um outro algoritmo melhor.



Limite Superior

- **V. Strassen** apresentou em **1969** um algoritmo para Multiplicação de Matrizes Quadradas com Complexidade $\mathbf{O\ (n^{\log 7}) = 0\ (n^{2.807})}$**;**

- Assim, a cota superior ou limite superior para o problema de multiplicação de matrizes passou a ser $\mathbf{0\ (n^{2.807})}$**;**



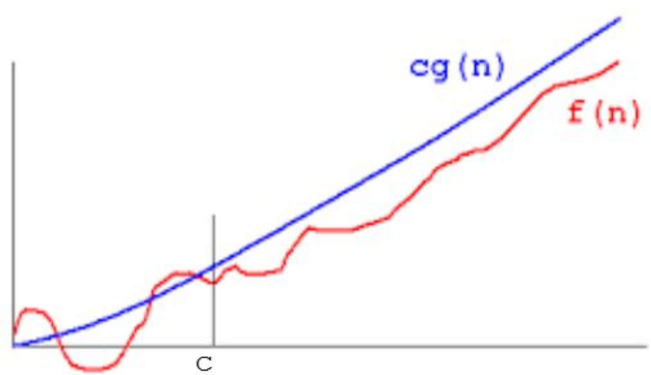
Limite Superior

- Em **1990, Coppersmith e Winograd** melhorarm esta marca para = **0 (n $^{2.376}$);**

- Em 2010, A. Stothers apresentou um algoritmo de complexidade **0 (n $^{2.373}$);**

- Em 2011, V. Willians melhorou ainda mais a cota superior do algoritmo com uma complexidade **0 (n $^{2.372}$);**

- **Portanto, a Cota Superior atual para o problema de multiplicação de matrizes é 0 (n $^{2.372}$);**



Analogia com Record Mundial

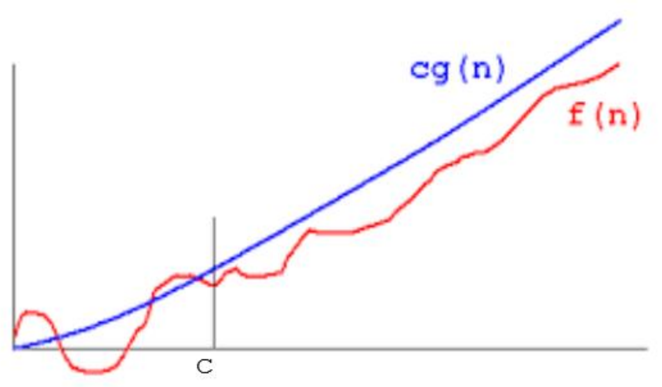
- A cota superior para um problema é análoga ao Record Mundial de uma modalidade de esporte, por exemplo, Atletismo;
- Ele é estabelecido pelo melhor atleta (algoritmo) do momento;
- Assim, como o record mundial, a Cota Superior, pode ser melhorada por um algoritmo (atleta) maiz veloz.

Cota Superior – 100m rasos

1998	Carl Lewis	9s92
1993	Linford Christie	9s87
1999	Maurice Greene	9s79
2007	Asata Powel	9s74
2008	Usain Bolt	9s72
2009	Usain Bolt	9s58

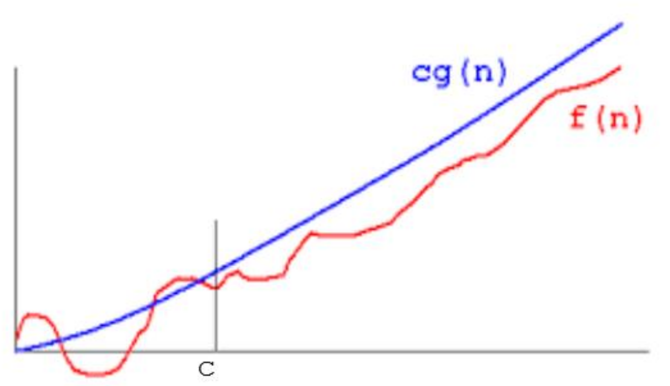
Limite Inferior

- Às vezes é possível demontrar que, para um dado problema, qualquer que seja o algoritmo a ser usado, o problema requer pelo menos um certo número de operações;
- Esse número mínimo de operações é **INTRÍNSECO** ao problema a ser resolvido**;**
- Essa complexidade é chamada Cota Inferior (Lower Bound) do problema;
- Para o problema da multiplicação de matrizes quadradas nxn, apenas ler os elementos das duas matrizes de entrada ou para produzir os elementos da matriz produto leva tempo $O(n^2)$.
- **Assim, uma possível cota inferior trivial para o problema é** $O(n^2)$.



Algoritmo Ótimo

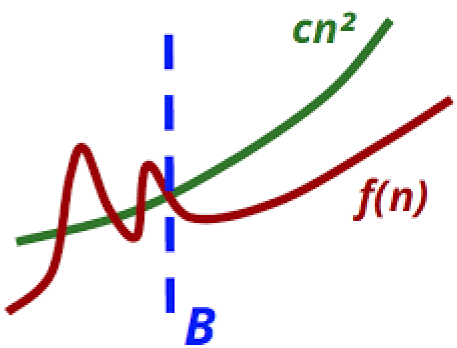
- Se um algoritmo tem uma complexidade igual à cota inferior do problema, então ele é assintoticamente ótimo ou simplesmente **ótimo**!

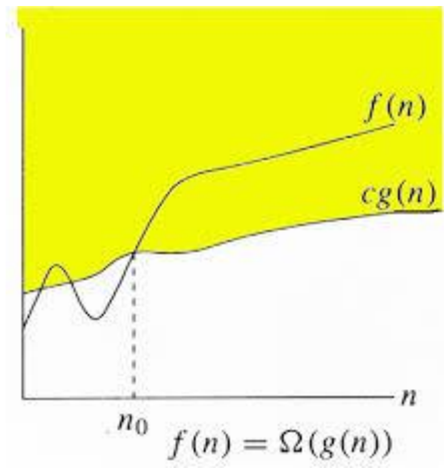


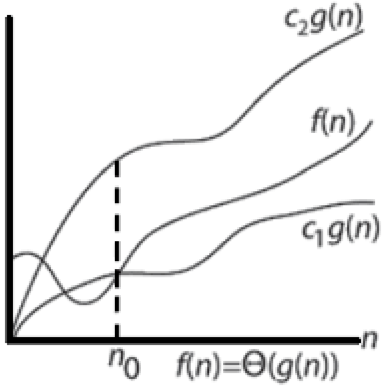
Outras Notações

- Da mesma forma que a notação Big-Oh provê uma forma assintótica de dizer que uma função é "**menor ou igual**" a outra função, há outras notações que provêm formas assintóticas para fazer outras formas de comparação;

- Em Análise de Algoritmos essas outras formas assintóticas são conhecidas por Big-Omega e Big-Theta.

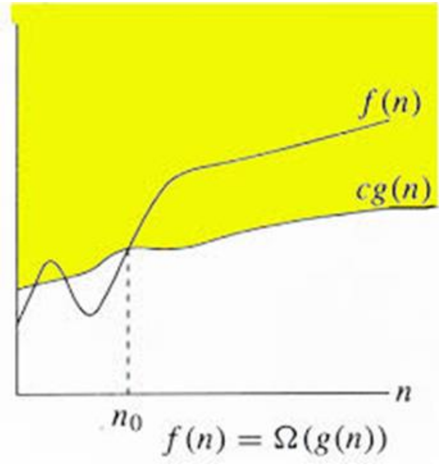






Notação Big-Omega (Ω)

- Seja f(n) e g(n) funções que mapeiam inteiros não negativos para números reais;
- Diz-se que f(n) é **Ω**(g(n)) se existir uma constante real c > 0 e uma constante inteira $n_o \geq 1$ tal que $f(n) \geq cg(n)$ para todo inteiro $n \geq n_0$;
- Essa definição permite nos dizer – de forma assintótica – que uma função é maior ou igual à outra função, por um fator constante.

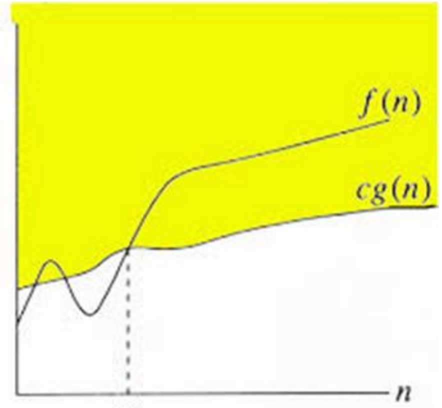


Notação Big-Omega (Ω)

- Essa notação é importante para se visualizar a quantidade mínima de recursos que um algoritmo requer, em tempo de execução;

- Encontrar-se a quantidade de recursos necessárias para se executar um algoritmo é tão importante quanto a determinação do tempo de execução.

- Se $\Omega(n^3)$ é o limite inferior de um algoritmo, então $\Omega(n^2)$, $\Omega(n)$, $\Omega(1)$, etc,. Também o são. Por exemplo, se o tempo mínimo de execução de um algoritmo é 2n + 5 e o máximo é 4n+34, pode-se dizer que o algoritmo é $\Omega(n)$.

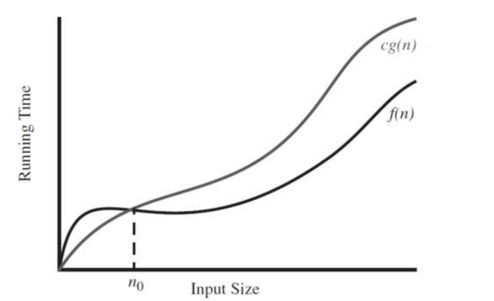


$f(n) = \Omega(g(n))$

A função de complexidade $F(n) = 1/2n^2 - 1/2n$ é $\Omega(n^2)$?

Ordem de Complexidade
$F(n) = 1/2n^2 - 1/2n$ é $\Omega(n^2)$?

- Necessita-se de uma constante real **$c > 0$** e uma constante inteira **$n_o \geq 1$** tal que **$1/2n^2 - 1/2n \geq c.n^2$** para todo inteiro **$n \geq n_0$** .
- **$1/2n^2 - 1/2n = n/2\,(n-1) \geq n/2 \,.\, n/2$** (pois, $n-1 > n/2$, para $n>=2$)
- **Portanto, $n/2\,(n-1) \geq ¼\, n^2$**
- Logo, existe uma constante $c=1/4$ ($c>0$), tal que **$1/2n^2 - 1/2n \geq 1/4.n^2$**
- **Assim, $1/2n^2 - 1/2n$** é **$\Omega(n^2)$**



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Ordem de Complexidade
$F(n) = 1/2n^2 - 1/2n$ é $\Omega(n^2)$? – (outra solução)

- Necessita-se de uma constante real **c > 0** e uma constante inteira $\mathbf{n_0} \geq$ **1** tal que $\mathbf{1/2n^2 - 1/2n} \geq \mathbf{c.n^2}$ para todo inteiro $\mathbf{n \geq n_0}$.

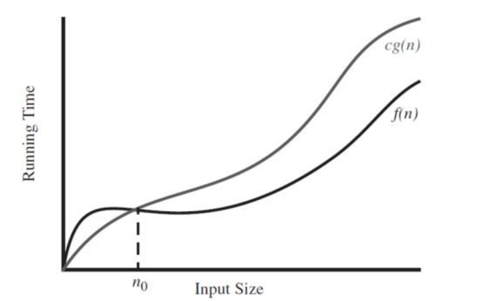
- $n^2/2 - n/2 = n^2/4 + n^2/4 - n/2 = n^2/4 + (n^2/4 - n/2)$

- **($n^2/4 - n/2$) é sempre positivo para n >= 2**

- **Portanto,** $\mathbf{1/2n^2 - 1/2n = n^2/4 + (n^2/4 - n/2) \geq 1/4\ n^2}$ **para n >= 2**

- Logo, existe uma constante c=1/4 (c>0), tal que $\mathbf{1/2n^2 - 1/2n \geq 1/4.n^2}$

- **Assim,** $\mathbf{1/2n^2 - 1/2n}$ é $\mathbf{\Omega(n^2)}$



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

A função de complexidade $F(n) = 100\log n - 10n + 2n\log n$ é $\Omega(n\log n)$?

Ordem de Complexidade
$F(n) = 100\log n - 10n + 2n\log n$ é $\Omega(n\log n)$

- Necessita-se de uma constante real **c > 0** e uma constante inteira **$n_o \geq$ 1** tal que **100log n – 10n + 2nlogn** $\geq$ **c.nlog n** para todo inteiro **$n \geq n_0$** .

- **100log n – 10n + 2nlogn** $\geq$ **2nlogn – 10n** (descartando-se 100logn)

- **100log n – 10n + 2nlogn** $\geq$ **2nlogn – 10n = nlogn + (nlogn – 10n)**

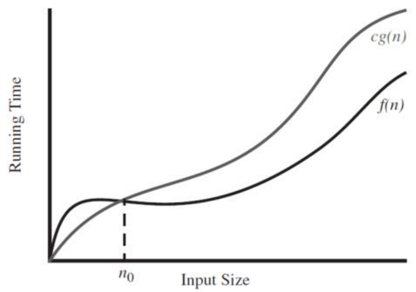
- nlogn – 10n será sempre positivo a partir de n = 1024 (log na base 2)

Sempre positivo, a partir de n = 1024

- Logo: **100log n – 10n + 2nlogn** $\geq$ **2nlogn – 10n = nlogn + (nlogn – 10n)** $\geq$ nlogn

- **Assim, existe uma constante c = 1 tal que 100log n – 10n + 2nlogn** ***para todo n >= 1024.***

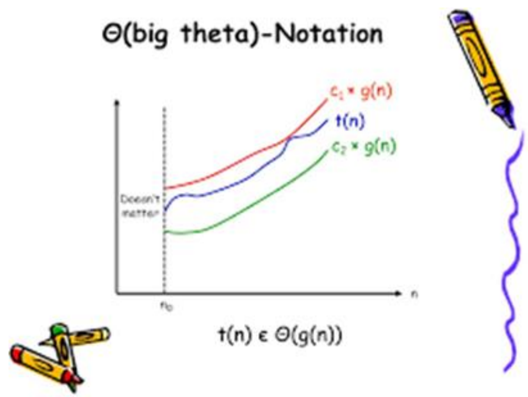
- **Portanto, 100log n – 10n + 2nlogn é Ω(nlogn)**



The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Notação Big-Theta (Θ)

- Seja f(n) e g(n) funções que mapeiam inteiros não negativos para números reais;
- Diz-se que f(n) é **Θ**(g(n)) se existir constantes reais c’ > 0 e c’’ > 0 e uma constante inteira $n_o \geq 1$ tal que $c'g(n) \leq f(n) \leq c''g(n)$, para todo inteiro $n \geq n_0$;
- Essa definição permite nos dizer – de forma assintótica – que duas funções são iguais, por um fator constante.



A função de complexidade $F(n) = 555n^2$ é $\Theta(n^2)$?

A função de complexidade $F(n) = 555n^2$ é $\Theta(n^2)$?

- Deve-se mostrar que $555n^2$ é $O(n^2)$ e também que $555n^2$ é $\Omega(n^2)$;
- Dos slides anteriores, é fácil mostrar que $555n^2$ é $O(n^2)$ e que $555n^2$ é $\Omega(n^2)$;
- Portanto, $F(n) = 555n^2$ é $\Theta(n^2)$.