Complexidade de Algoritmos

Introdução à
Complexidade de algoritmos
Crescimento de funções

Visão geral

Complexidade de Algoritmos

Complexidade de algoritmos

Algoritmos:

Conjunto e estrutura de operações destinada a geração de resultados esperados.

Dependente do conjunto de dados de entrada (tanto em sua distribuição como em seu tamanho).

Associados à classes de problemas, com diversidade de opções de utilização (em geral existem várias soluções, vários algoritmos para o mesmo problema).

Complexidade de algoritmos

Objetivo de analisar algoritmos:

Análise proporciona oportunidade de projetar algoritmos eficientes.

Mecanismos de análise permitem desenvolver um algoritmo e depois disso :

- Avaliar sua eficiência
- Comparar com outros algoritmos

Mecanismos de análise apoiam a seleção de metodologias adequadas para concepção de algoritmos eficientes

Complexidade de algoritmos

Aspectos analisados, em geral:

Espaço de armazenamento utilizado

- Quantidade de memória, seja principal ou secundária.

Capacidade de processamento necessária

- Tempo de processamento envolvido nos casos médios e no pior caso

Algoritmos:

Conjunto e estrutura de operações destinada a geração de resultados esperados.

Dependente do conjunto de dados de entrada (tanto em sua distribuição como em seu tamanho).

Associados à classes de problemas, com diversidade de opções de utilização (em geral existem várias soluções, vários algoritmos para o mesmo problema).

A análise de complexidade de algoritmos pode ser feita levando em conta a análise do tempo de execução do mesmo.

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`int aux1=1;`	1	op1
`int aux2=1;`	1	op1
`int i=0;`	1	op1
`while (i < n) {`	n+1	op4
`aux1 = aux1 * aux2;`	n	op1 e op2
`aux2 = aux2 + 1;`	n	op1 e op3
`i = i + 1;`	n	op1 e op3
`}`		
`i = 0;`	1	op1
`}`		

$$T(n) = 4\ op1 + (n+1)\ op4 + 3n\ op1 + n\ op2 + 2n\ op3$$

Nesta análise são sempre realizadas diversas simplificações

Utilização do tempo absoluto (minutos, milissegundos) não é adequada, pois varia de acordo com:

- Processador e sua velocidade
- Quantidade de processadores
- Quantidade e tipo de memória
- Sistema operacional
- Quantidade de processos simultâneos no sistema operacional
- Compilador, linguagem
- .....

Nesta análise são sempre realizadas diversas simplificações

Principal motivo de simplificações:

Os detalhes envolvidos nos cálculos da estimativa do tempo se tornam irrelevantes com o aumento da quantidade de dados tratados pelo algoritmo

Exemplo - instruções fora do laço principal:

$n=1 \rightarrow$ 36%;

$n=100 \rightarrow$ 0,6%

Complexidade de algoritmos
Crescimento de funções

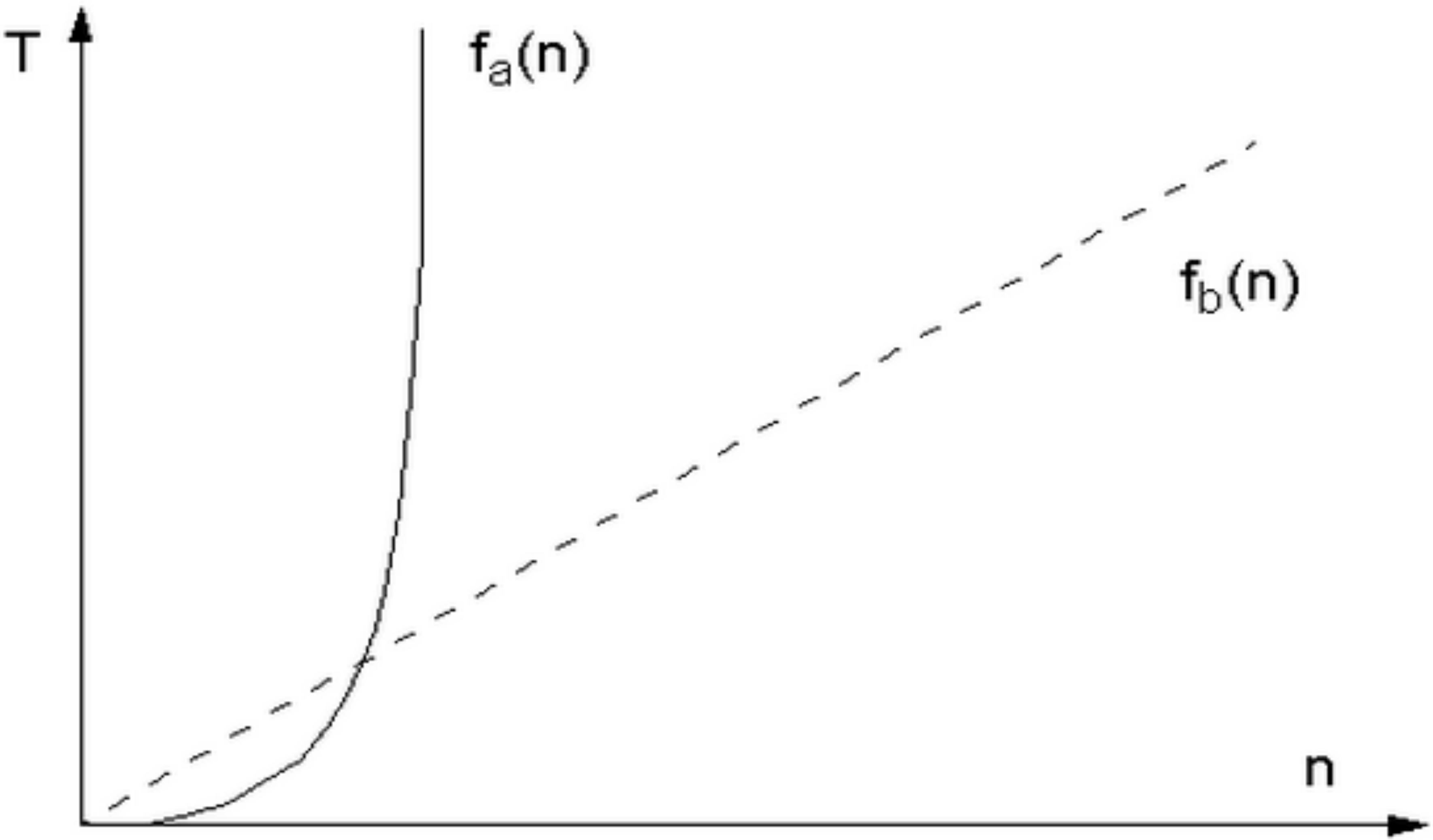
Quando observa-se apenas tamanhos de entradas de dados muito grandes, a tendência é que apenas a ordem de crescimento do tempo de execução se torne relevante.

Complexidade de algoritmos
Crescimento de funções

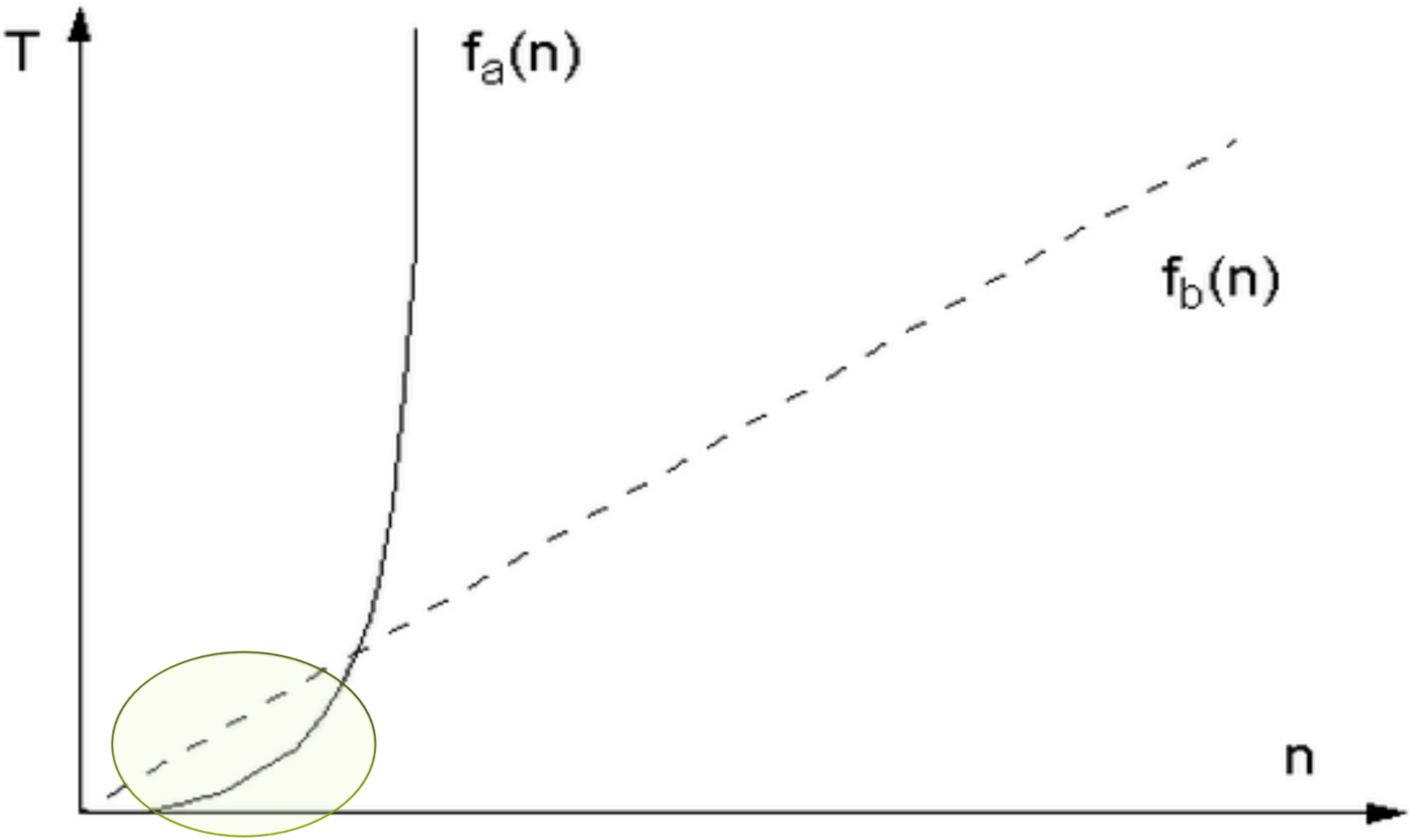
Quando observa-se apenas tamanhos de **entradas de dados muito grandes**, a tendência é que apenas a **ordem de crescimento** do tempo de execução se torne relevante.

Apesar de diferenças em valores pequenos, quando os valores aumentam as tendências de crescimento evidenciam os resultados gerais

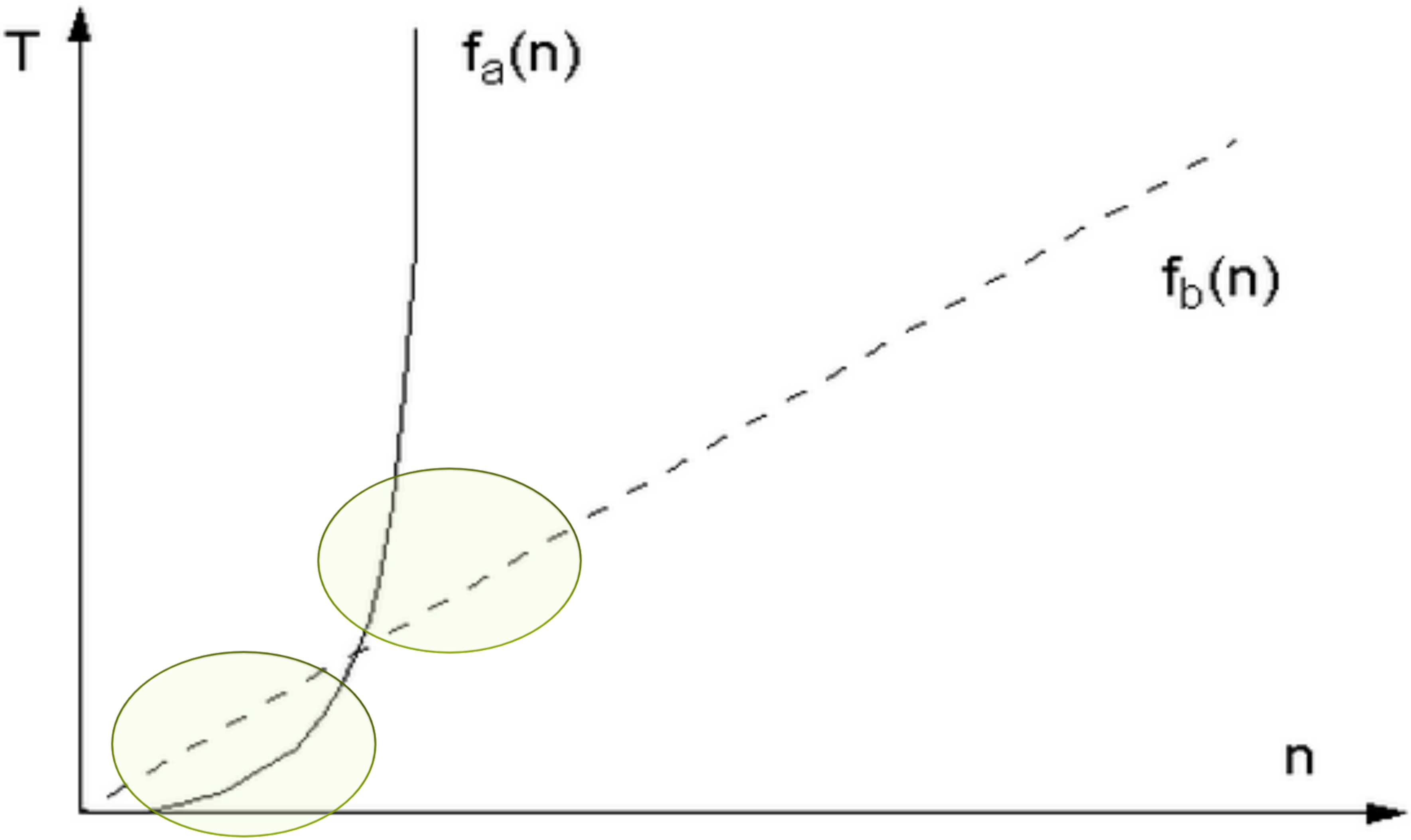
Complexidade de algoritmos
Crescimento de funções



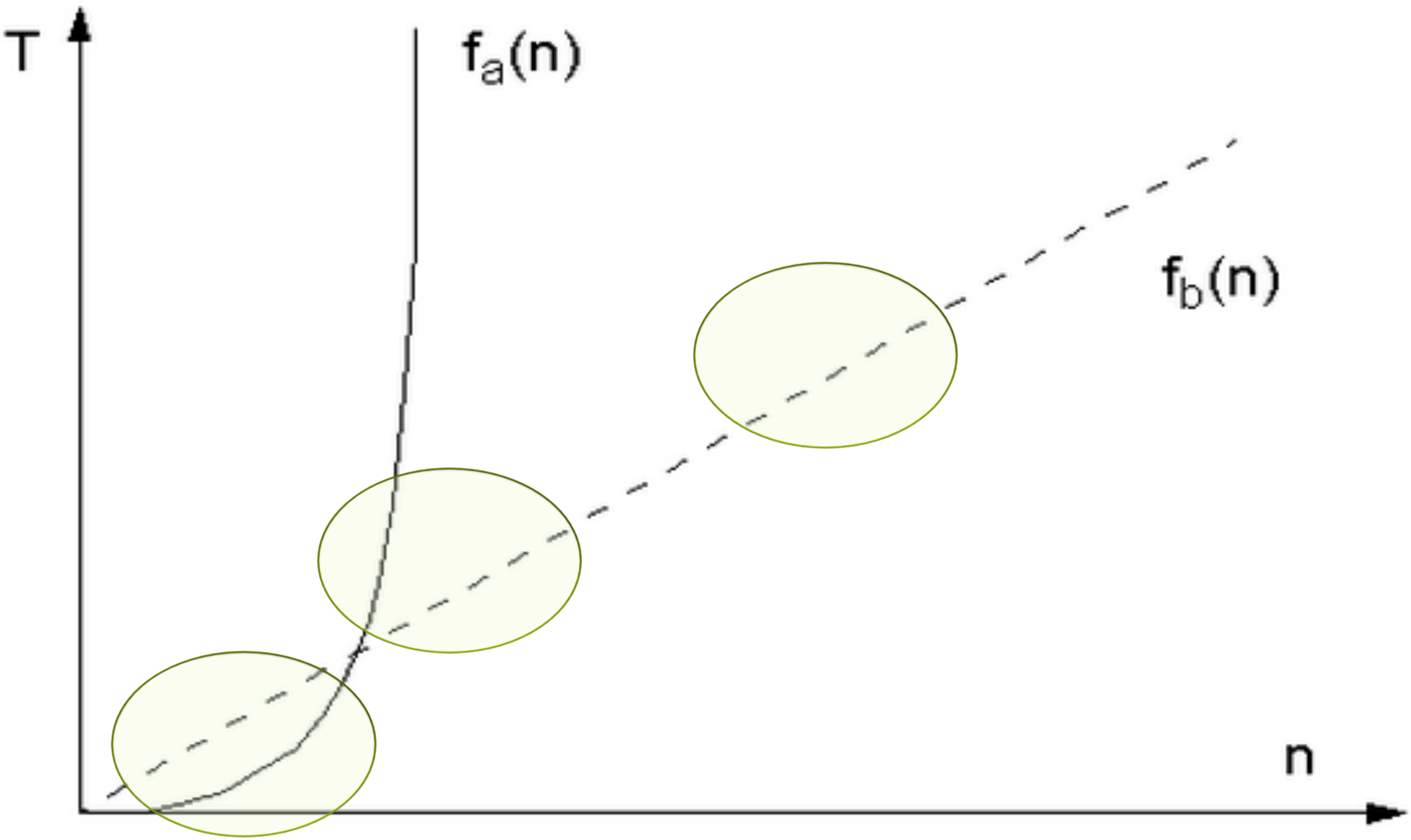
Complexidade de algoritmos
Crescimento de funções



Complexidade de algoritmos
Crescimento de funções



Complexidade de algoritmos
Crescimento de funções



Complexidade de algoritmos
Crescimento de funções

Exemplos de ordem de crescimento

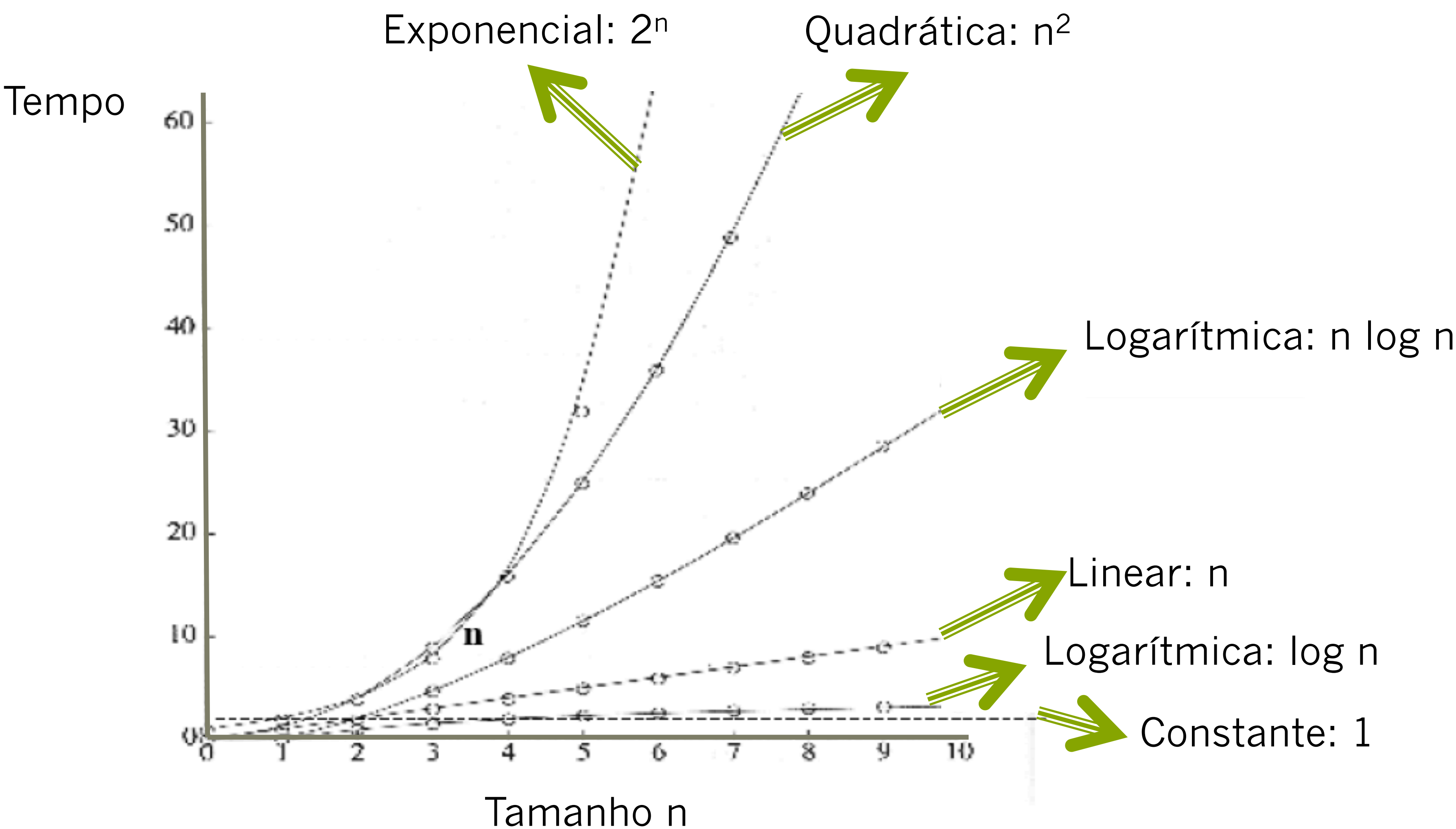
- Constante ($t(n)=1$)
- Logarítmico ($t(n)=\log n$)
- Linear ($t(n)=n$)
- Quadrático ($t(n)=n^2$)
- Cúbico ($t(n)=n^3$)
- Polinomial ($t(n)= n^k$, para k com valores pequenos)
- Exponencial ($t(n)= n^n$)

Complexidade de algoritmos
Crescimento de funções

Função de crescimento ou custo	**Tamanho da entrada de dados n**					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Fonte: Cormen

Complexidade de algoritmos
Crescimento de funções



Fonte: Cormen

Complexidade de algoritmos
Crescimento de funções

Exemplos de ordem de crescimento

- Exponencial ($t(n)= 2^n$; $t(n)= 3^n$; $t(n)= n^n$)

a) Dificuldades para utilização prática

b) Típico de algoritmos com abordagem de força bruta

c) Exemplos de grandezas obtidas:

$2^{10} = 1024$; $2^{20} = 1.048.576$;

$2^{40} = 1,09951E+12$.

Complexidade de algoritmos
Crescimento de funções

Exemplos de ordem de crescimento

- Quadrático ($t(n)=n^2$) ou Cúbico ($t(n)=n^3$)

a) Viáveis em problemas pequenos

b) Necessidade de processar todos os pares/conjuntos de dados

c) Produto de vetores, produto de matrizes

d) Para $n=10$, $n^2 = 100$; $n^3 = 1.000$;

Para $n=100$, $n^2 = 10.000$; $n^3 = 1.000.000$;

Para $n=1000$, $n^2 = 1.000.000$; $n^3 = 1.000.000.000$;

Complexidade de algoritmos
Crescimento de funções

Exemplos de ordem de crescimento

- Logarítmico ($t(n)=\log n$)

a) Viável em problemas grandes

b) Típico em abordagens de reduções de problemas e combinações de soluções

c) Taxas de crescimento muito pequenas (duplica apenas quando n aumenta até n^2)

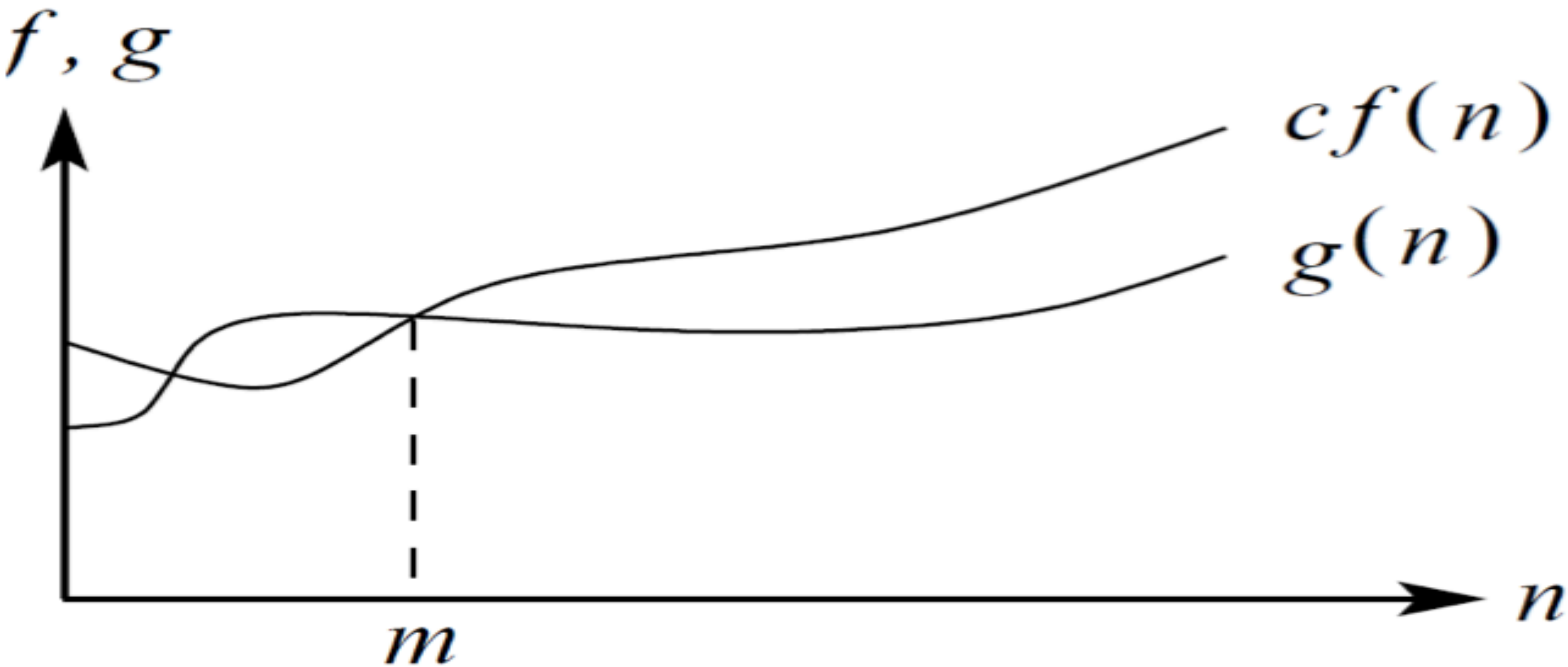
Análise assintótica

Resumindo:

- Identifica o comportamento do algoritmo com conjuntos de dados muito grandes
- Apenas a tendência de crescimento é relevante
- Possibilita uma avaliação válida para a maior parte das entradas

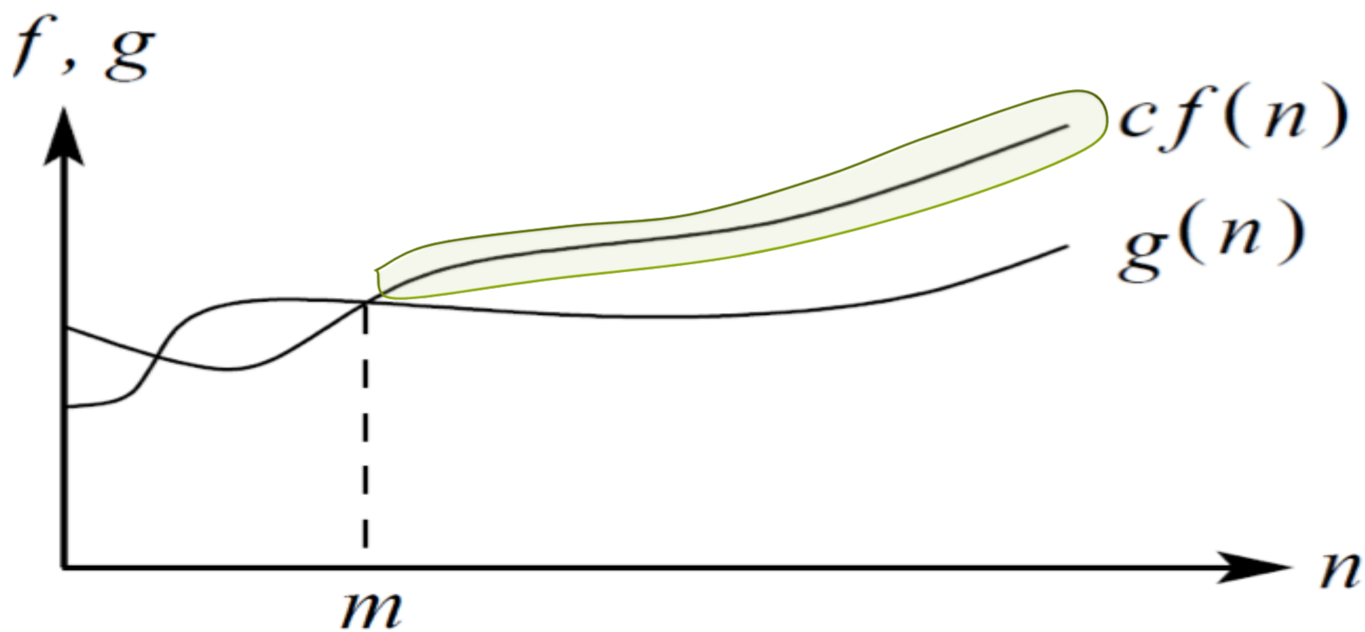
Análise assintótica

Intuitivamente: existe um valor inicial m e uma constante c que permitem a relação, exibida na figura, entre duas funções f(n) e g(n)



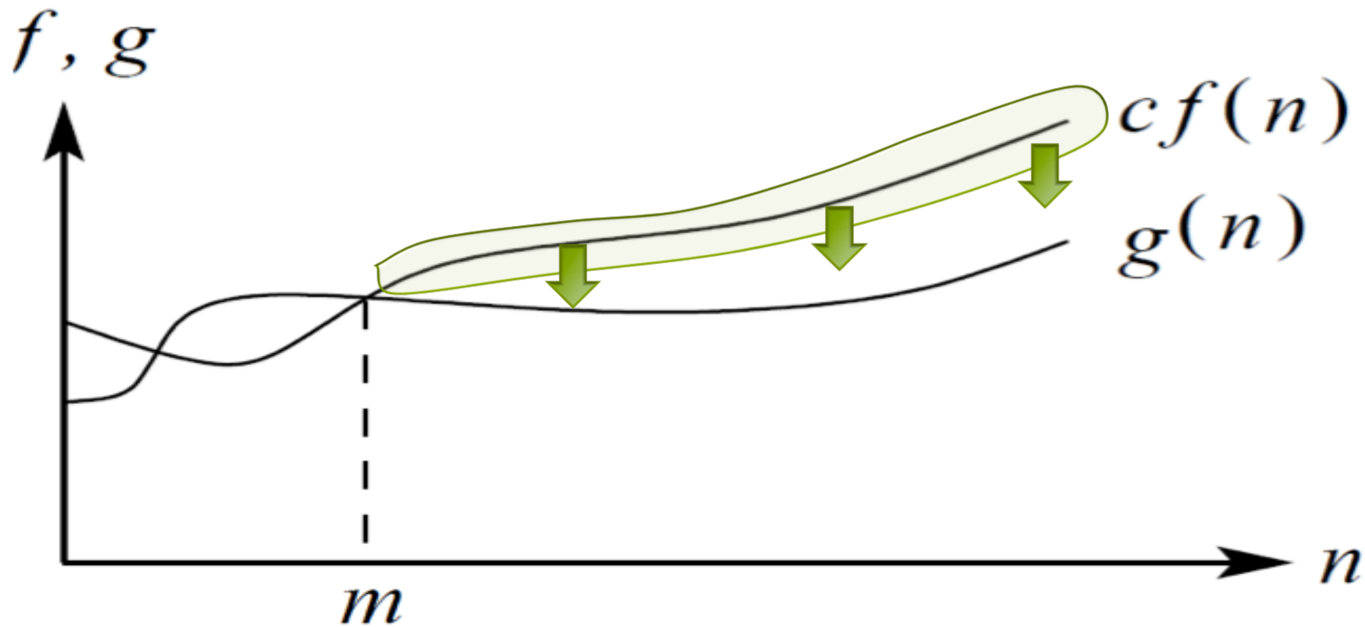
Análise assintótica

Intuitivamente: existe um valor inicial m e uma constante c que permitem a relação, exibida na figura, entre duas funções f(n) e g(n)



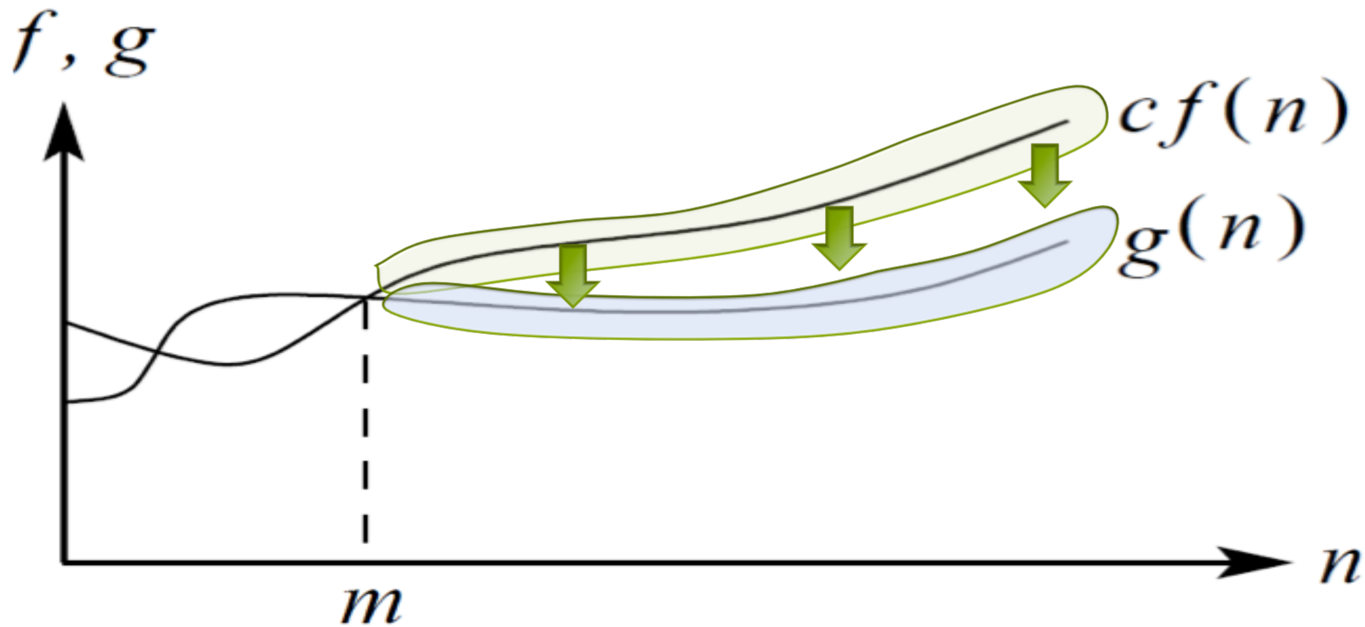
Análise assintótica

Intuitivamente: existe um valor inicial m e uma constante c que permitem a relação, exibida na figura, entre duas funções f(n) e g(n)



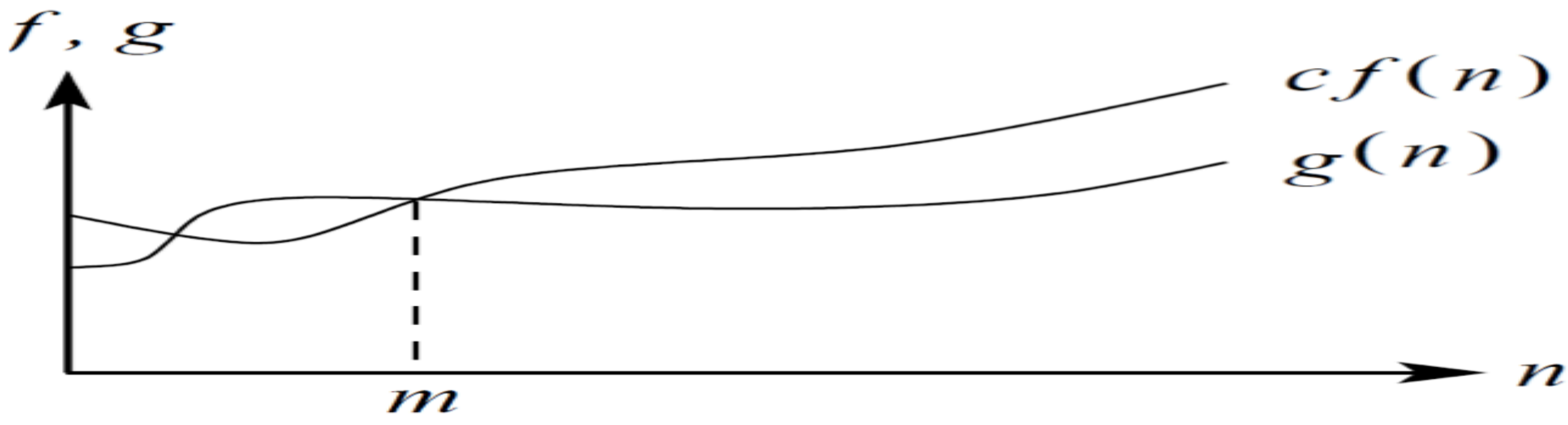
Análise assintótica

Intuitivamente: existe um valor inicial m e uma constante c que permitem a relação, exibida na figura, entre duas funções f(n) e g(n)



Análise assintótica

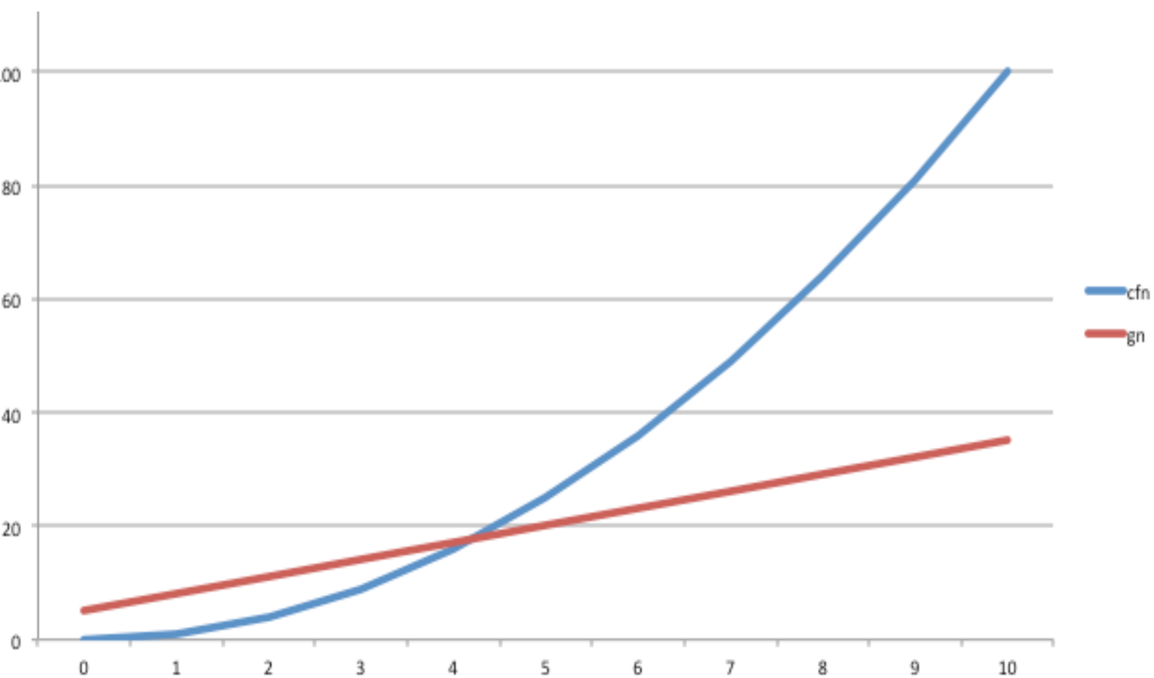
OU: a função f(n) é dominante assintoticamente sobre a função g(n) no caso de existirem duas constantes positivas c e m de modo que, para valores de n maiores ou iguais a m, o valor de g(n) será sempre menor que o valor de c f(n)

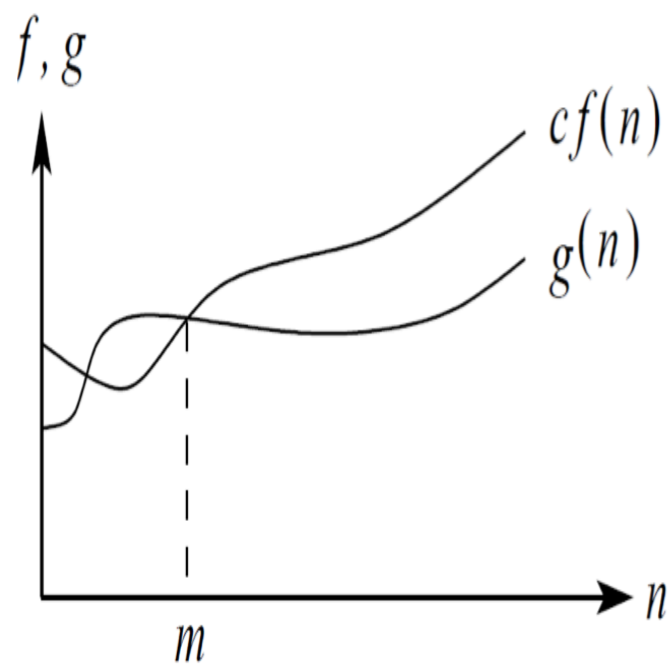


A análise assintótica

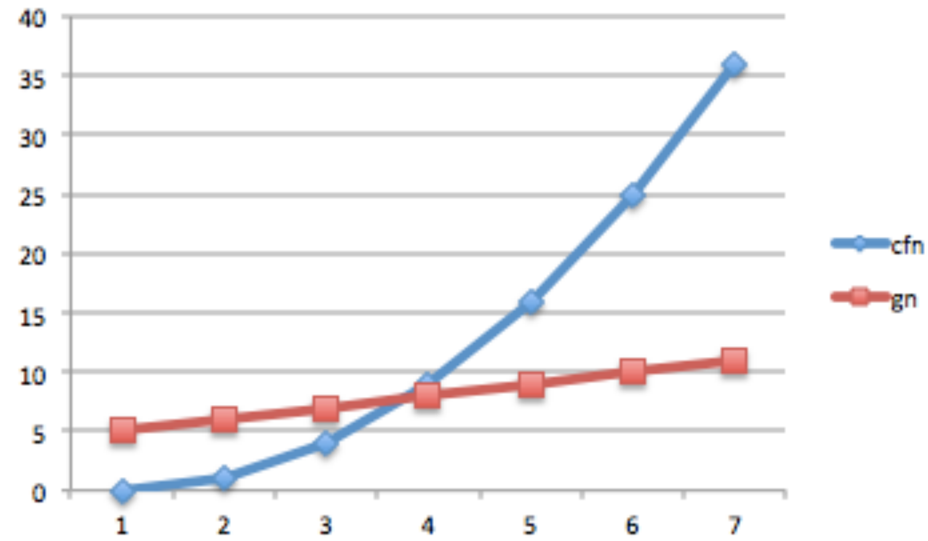
Exemplos:

a) $f(n) = n^2$; $g(n) = 3n+5$; $c=1$; $m=5$





A análise assintótica



Exemplos:

a) $f(n) = n^2$; $g(n) = n+5$

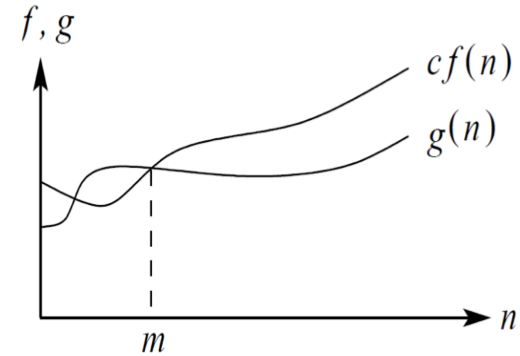
a função f(n) domina assintoticamente a função g(n), para c=1 e m=3

Ou seja:

$|g(n)| \leq c|f(n)|$ para todo $n \geq m$

$|g(n)| \leq 1|f(n)|$ para todo $n \geq 3$

Notação O (big O)



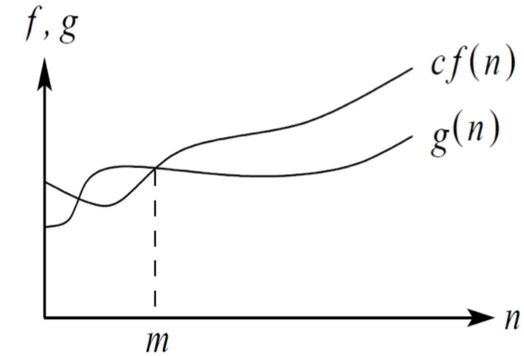
Definição:

Diz-se que: g(n) = O(f(n))

quando f(n) domina assintoticamente g(n)

Ou seja: g(n) é da ordem no máximo f(n)

Notação O (big O)



Definição:

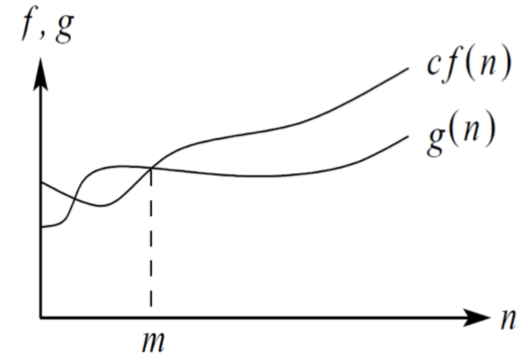
Diz-se que: g(n) = O(f(n))

quando f(n) domina assintoticamente g(n)

Ou seja: g(n) é da ordem no máximo f(n)

Ou ainda: o tempo para a execução de um programa é dado por $T(n) = O(n^2)$, se existem constantes c e m de forma que $T(n) \leq cn^2$ para qualquer $n \geq m$

Notação O (big O)



Definição:

Diz-se que: g(n) = O(f(n))

quando f(n) domina assintoticamente g(n)

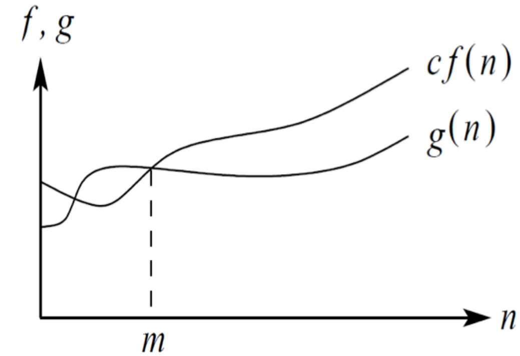
Ou seja: g(n) é da ordem no máximo f(n)

Ou ainda: o tempo para a execução de um programa é dado por $T(n) = O(n^2)$, se existem constantes c e m de forma que $T(n) \leq cn^2$ para qualquer $n \geq m$

Ou: a cota assintótica superior de g(n) é cf(n), para todo $n \geq m$

Fonte: Cormen

Notação O (big O)



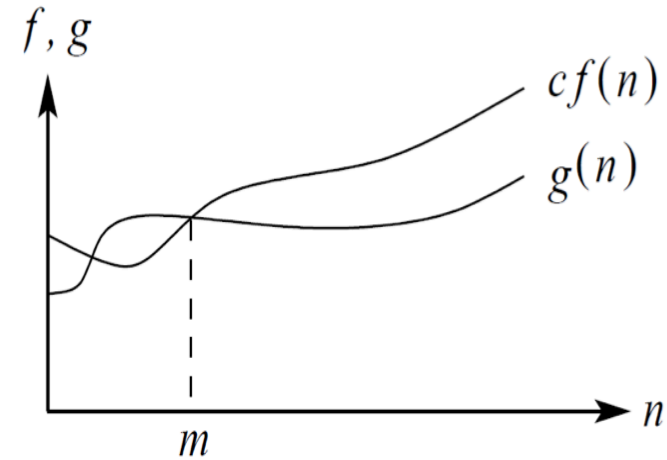
Definição:

Diz-se que: $g(n) = O(f(n))$

$O(f(n)) = \{g(n)$: existem constantes positivas c e m de modo que $0 \leq g(n) \leq cf(n)$ para todo $n \geq m\}$

$O(f(n))$, ou "O grande" de $f(n)$ é o conjunto de funções $g(n)$ de modo que existem constantes positivas c e m para as quais $g(n) \leq cf(n)$ para todo $n \geq m$

Fonte: Cormen



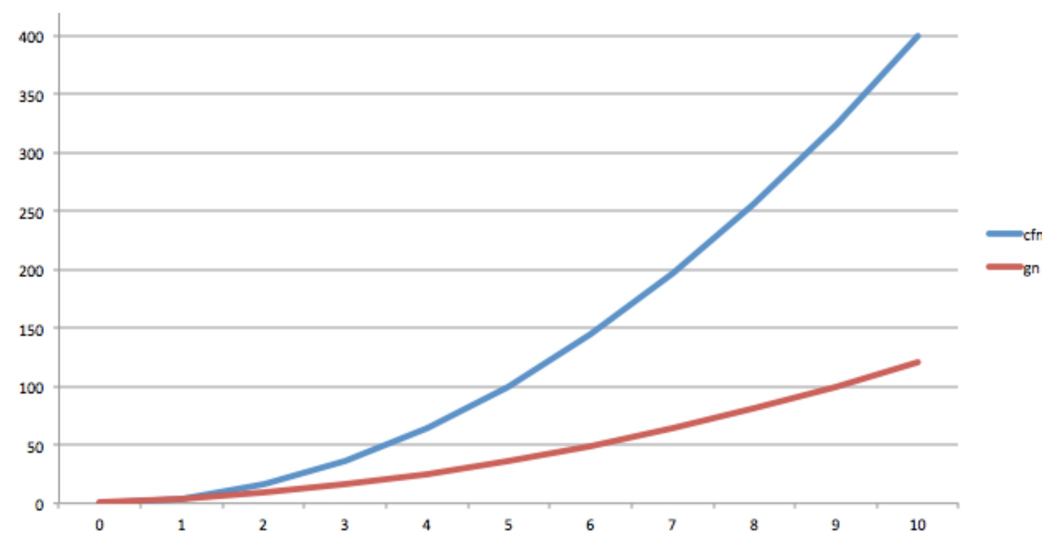
Notação O (big O)

Exemplos:

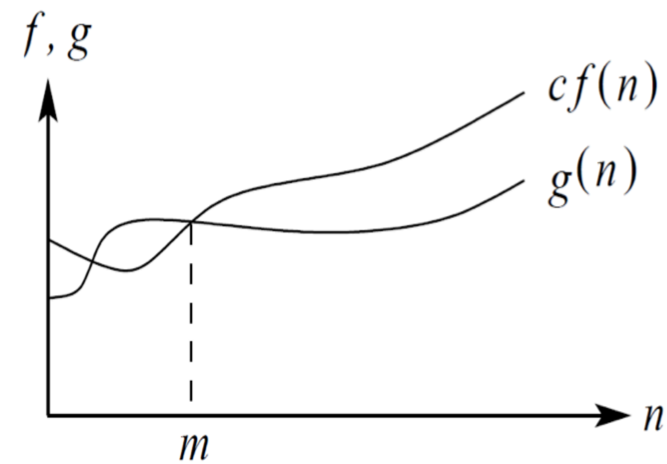
$f(n) = (n+1)^2 = O(n^2)$

pois existe c=4, m=1 de modo que:

$(n+1)^2 \leq 4n^2,\ n \geq 1$



Notação O (big O)

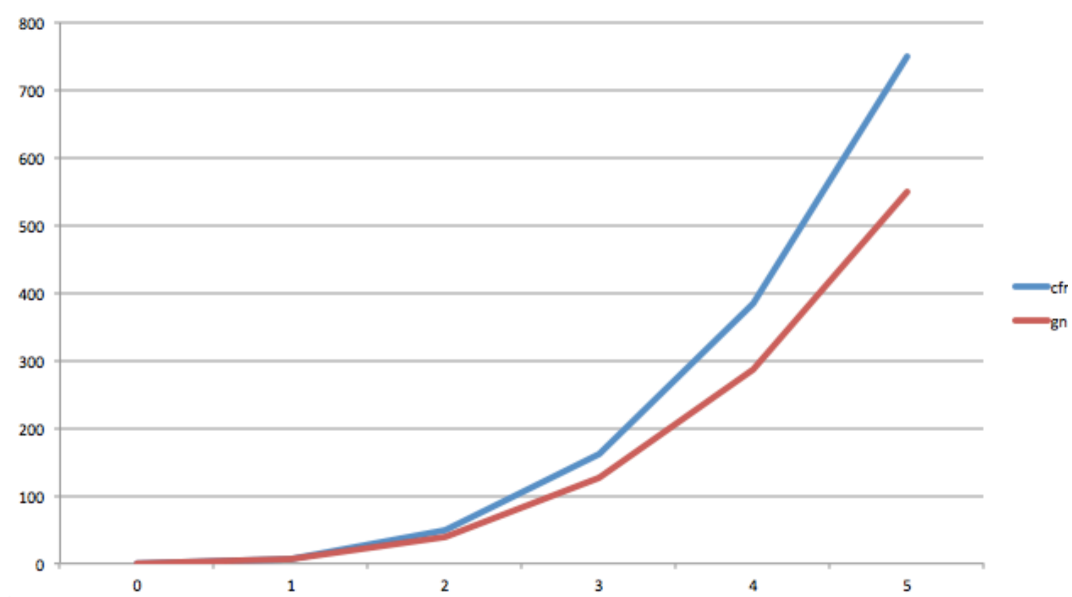


Exemplos:

$f(n) = 4n^3 + 2n^2 = O(n^3)$

pois existe c=6, m=1 de modo que:

$4n^3 + 2n^2 \leq 6n^3, n \geq 1$

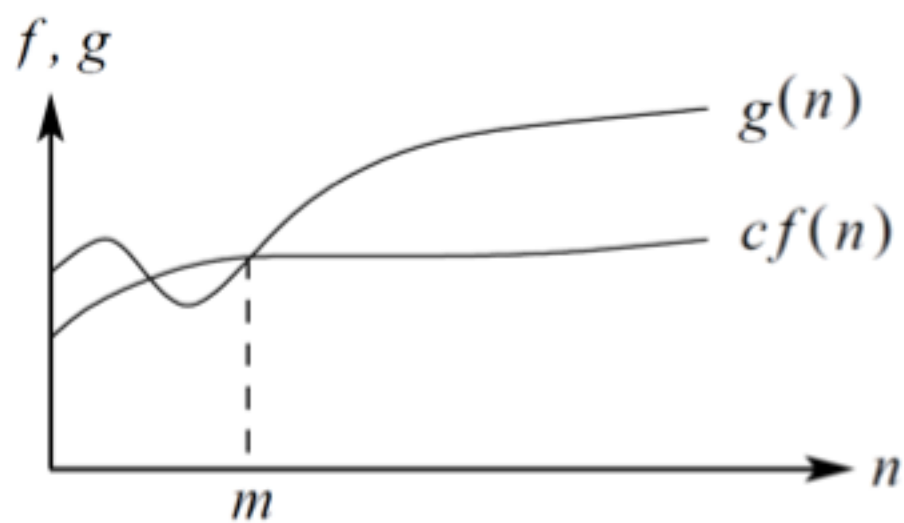


Notação Ω (Ômega)

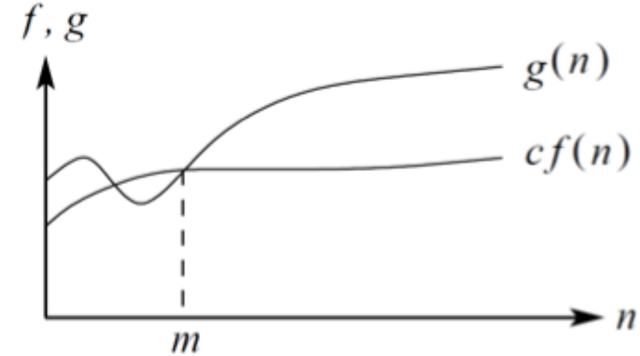
Definição:

Uma função g(n) é **Ω** (f(n)) se g(n) domina assintoticamente f(n)

Portanto a notação **Ω** denota um limite inferior , enquanto a notação **O** denota um limite superior



Fonte: Cormen



Notação Ω (Ômega)

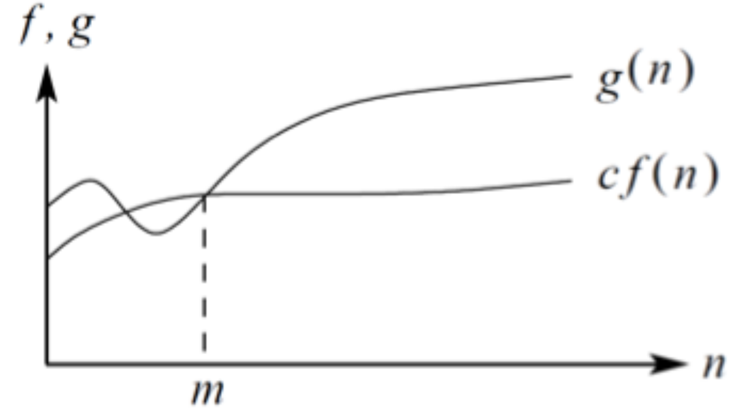
Definição:

Diz-se que: $g(n) = \mathbf{\Omega}(f(n))$

$\mathbf{\Omega}(f(n)) = \{g(n)$: existem constantes positivas c e m de modo que $0 \leq cf(n) \leq g(n)$ para todo $n \geq m\}$

$\mathbf{\Omega}(f(n))$, ou "ômega" de f(n) é o conjunto de funções g(n) de modo que existem constantes positivas c e m para as quais $g(n) \geq cf(n)$ para todo $n \geq m$

Fonte: Cormen



Notação Ω (Ômega)

Exemplos:

Considere que $f(n) = 3n^3 + 2n^2 + n = \mathbf{\Omega}(n^3)$

Pode ser demonstrado verificando que:

$n^3 \leq 3n^3 + 2n^2 + n$, para $n \geq m = 0$

Portanto, $f(n) = 3n^3 + 2n^2 + n = \mathbf{\Omega}(n^3)$

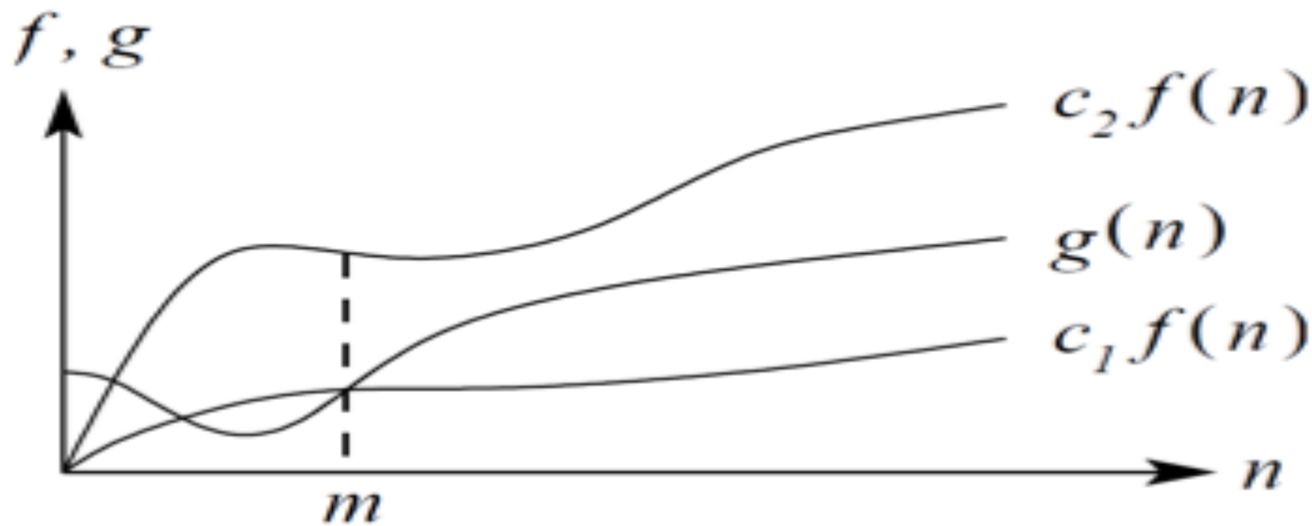
Fonte: Cormen

Notação Θ (Theta)

Definição:

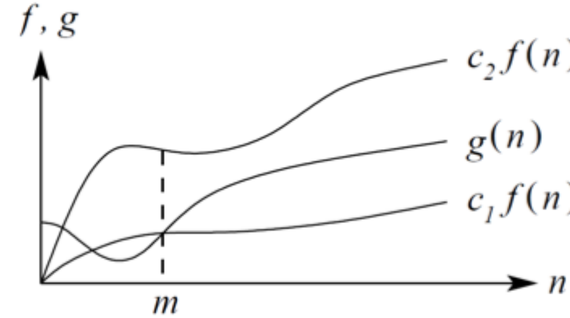
Uma função g(n) é **Θ** (f(n)) se g(n) e f(n) dominam assintoticamente uma à outra

Definição equivalente: g(n) = **Θ** (f(n)) se g(n) = O(c_1f(n)) e g(n) = **Ω** (c_2f(n))



Fonte: Cormen

Notação Θ (Theta)



Definição:

Diz-se que: $g(n) = \Theta(f(n))$

$\Theta(f(n)) = \{g(n)$: existem constantes positivas c_1, c_2 e m de modo que $0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$ para todo $n \geq m\}$

$\Theta(f(n))$, ou "Theta" de f(n) é o conjunto de funções g(n) de modo que existem constantes positivas c_1, c_2 e m para as quais $c_1 f(n) \leq g(n) \leq c_2 f(n)$ para todo $n \geq m$

Fonte: Cormen

Notação Θ (Theta)

Definição equivalente:

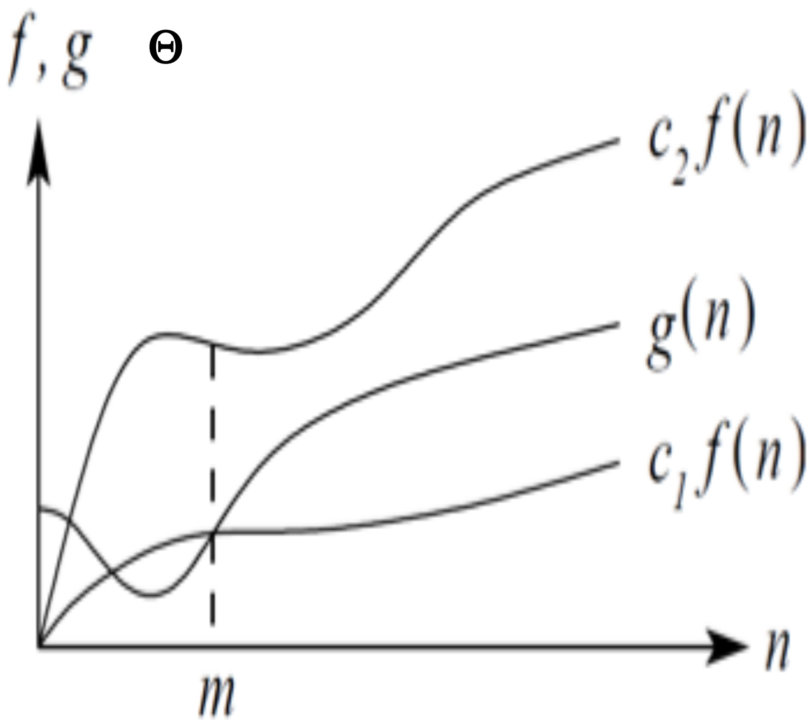
$g(n) = \Theta(f(n))$ se $g(n) = O(c_1 f(n))$ e $g(n) = \Omega(c_2 f(n))$

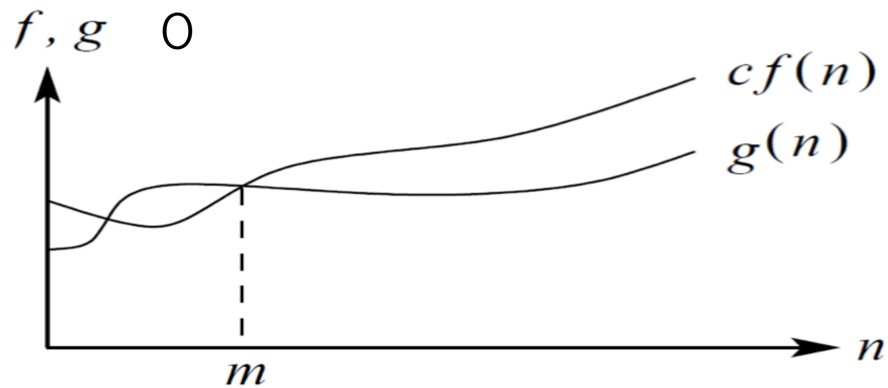
Fonte: Cormen

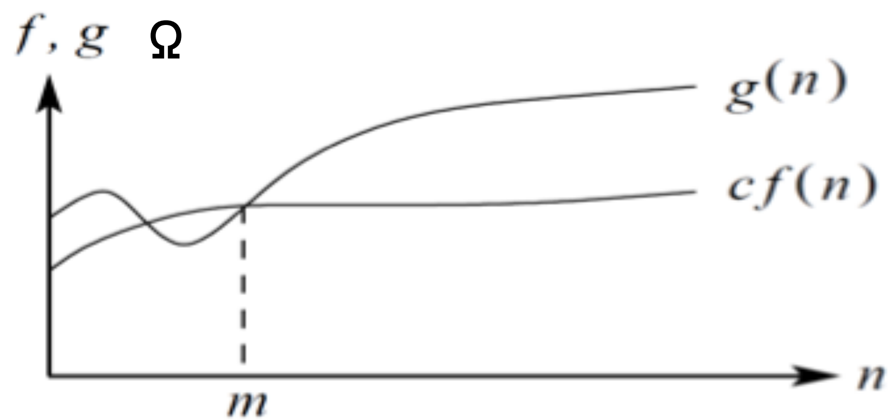
Notação Θ (Theta)

Definição equivalente:

$g(n) = \Theta(f(n))$ se $g(n) = O(c_1 f(n))$ e $g(n) = \Omega(c_2 f(n))$







Complexidade assintótica

Considerando

f é uma função de complexidade para um algoritmo, então

$O(f)$ é a complexidade assintótica do algoritmo

Uso: comparar algoritmos usando suas complexidades assintóticas

- O algoritmo $O(n)$ é melhor do que o $O(n^3)$
- Dois algoritmos com a mesma complexidade assintótica são equivalentes
- O algoritmo $O(\log n)$ é melhor do que o $O(n^3)$

Complexidade assintótica

Exercícios