

Complexidade Assintótica

Prof. Luiz Chaimowicz

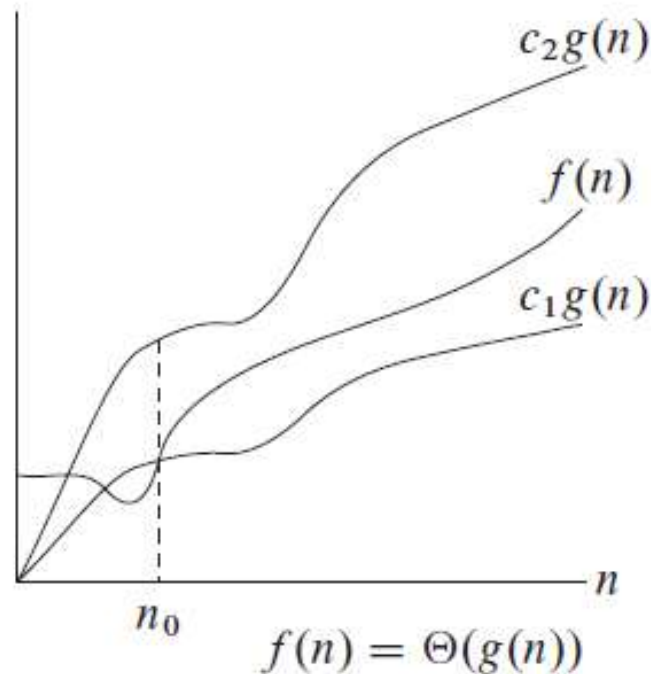
Complexidade Assintótica

- Na análise de algoritmos, na maioria das vezes usa-se o estudo da complexidade assintótica, ou seja, **analisa-se o algoritmo quando o valor de n tende a infinito**
- Nesse caso, não é necessário se preocupar com as constantes e termos de menor crescimento
- Usa-se notações especiais para representar a complexidade assintótica

Notação Θ

- Limite assintótico firme

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\} .^1$



Notação Θ

- Exemplo: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- É necessário encontrar constantes c_1 , c_2 e n_0 tais que:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

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

O lado direito é satisfeito com $c_2 = 1/2$ para n maior ou igual a 1

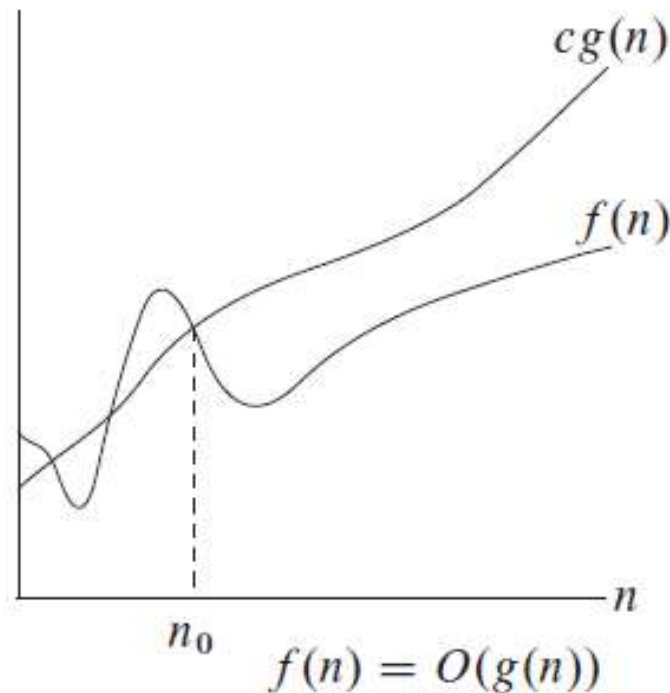
O lado esquerdo é satisfeito com $c_1 = 1/14$ para n maior ou igual a 7

Logo, a equação é satisfeita com $c_1 = 1/14$, $c_2 = 1/2$ e n_0 igual a 7

Notação O

- Limite assintótico superior

$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$



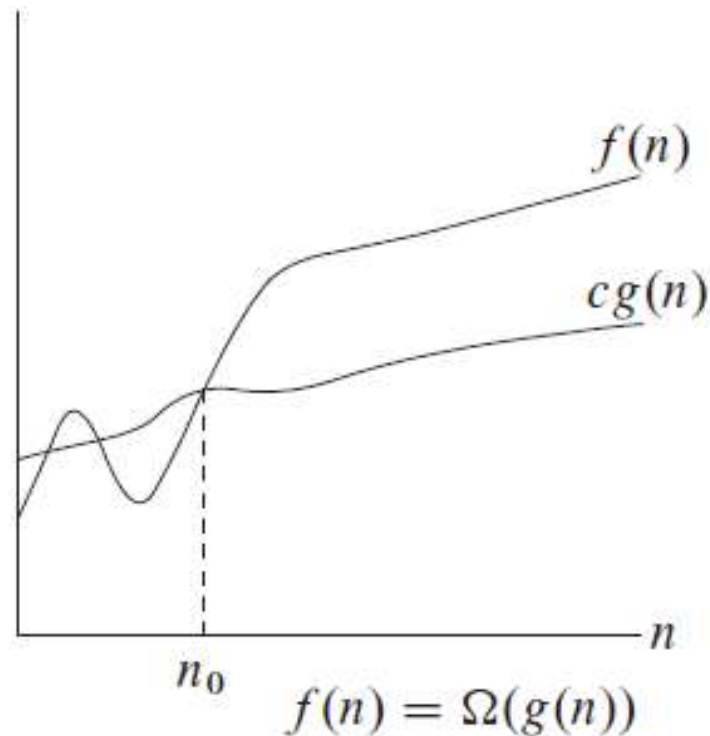
Notação O

- Apesar de muito usada na literatura como limite firme, a notação O é mais fraca que a notação Θ
 - $f(n) = \Theta(g(n))$ implica em $f(n) = O(g(n))$, mas não o contrário
 - $\Theta(g(n)) \subset O(g(n))$
- Como é um limite superior, é muito usado para o pior caso. Ex.
 - O inserção é $O(n^2)$, para qualquer entrada.

Notação Ω

- Limite assintótico inferior

$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$
 $0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\} .$



Notação Ω

- Também mais fraca que Θ
- Limite inferior, portanto pode ser usado para o melhor caso
- Inserção é $\Omega(n)$, para qualquer entrada

- **Teorema:**

Para quaisquer duas funções $f(n)$ e $g(n)$,
teremos $f(n) = \Theta(g(n))$ se e somente se
 $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Notações o e ω

- Representam, respectivamente limites superiores e inferiores estritos
- Notação o :

$o(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\} .$

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

- Notação ω :

$\omega(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\} .$

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

Propriedades

- Transitividade

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \text{ imply } f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \text{ imply } f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \text{ imply } f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \text{ imply } f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \text{ imply } f(n) = \omega(h(n))$$

- Reflexividade

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

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

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

Propriedades

- Simetria

$f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

- Simetria transposta

$f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

$f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$

Propriedades

- Com essas propriedades, é possível fazer uma analogia entre a comparação das funções assintóticas ***f*** e ***g*** e a comparação de números reais ***a*** e ***b***

$$f(n) = O(g(n)) \quad \text{is like} \quad a \leq b$$

$$f(n) = \Omega(g(n)) \quad \text{is like} \quad a \geq b$$

$$f(n) = \Theta(g(n)) \quad \text{is like} \quad a = b$$

$$f(n) = o(g(n)) \quad \text{is like} \quad a < b$$

$$f(n) = \omega(g(n)) \quad \text{is like} \quad a > b$$

Classes de Comportamento Assintótico

- Em geral, é interessante agrupar os algoritmos / problemas em **Classes de Comportamento Assintótico**, que vão determinar a complexidade inerente do algoritmo
- Como explicado, o comportamento assintótico é medido quando o tamanho da entrada (n) tende a infinito, com isso, as constantes são ignoradas e apenas o componente mais significativo da função de complexidade é considerado
- **Quando dois algoritmos fazem parte da mesma classe de comportamento assintótico, eles são ditos equivalentes.** Nesse caso, para escolher um deles deve-se analisar mais cuidadosamente a função de complexidade ou o seu desempenho em sistemas reais

Principais Classes de Problemas

$$f(n) = O(1)$$

- Algoritmos de complexidade $O(1)$ são ditos de **complexidade constante**.
- Uso do algoritmo independe de n .
- As instruções do algoritmo são executadas um número fixo de vezes.

Principais Classes de Problemas

$f(n) = O(\log n)$.

- Um algoritmo de complexidade $O(\log n)$ é dito ter **complexidade logarítmica**.
- Típico em algoritmos que transformam um problema em outros menores.
- Pode-se considerar o tempo de execução como menor que uma constante grande.
- Quando n é mil, $\log_2 n \approx 10$, quando n é 1 milhão, $\log_2 n \approx 20$.
- Para dobrar o valor de $\log n$ temos de considerar o quadrado de n .
- A base do logaritmo muda pouco estes valores: quando n é 1 milhão, o $\log_2 n$ é 20 e o $\log_{10} n$ é 6.

Principais Classes de Problemas

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

- Um algoritmo de complexidade $O(n)$ é dito ter **complexidade linear**.
- Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
- É a melhor situação possível para um algoritmo que tem de processar/produzir n elementos de entrada/saída.
- Cada vez que n dobra de tamanho, o tempo de execução dobra.

Principais Classes de Problemas

$$f(n) = O(n \log n)$$

- Típico em algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e juntando as soluções depois.
- Quando n é 1 milhão, $n \log_2 n$ é cerca de 20 milhões.
- Quando n é 2 milhões, $n \log_2 n$ é cerca de 42 milhões, pouco mais do que o dobro.

Principais Classes de Problemas

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

- Um algoritmo de complexidade $O(n^2)$ é dito ter **complexidade quadrática**.
- Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro de outro.
- Quando n é mil, o número de operações é da ordem de 1 milhão.
- Sempre que n dobra, o tempo de execução é multiplicado por 4.
- Úteis para resolver problemas de tamanhos relativamente pequenos.

Principais Classes de Problemas

$$f(n) = O(n^3)$$

- Um algoritmo de complexidade $O(n^3)$ é dito ter **complexidade cúbica**.
- Úteis apenas para resolver pequenos problemas.
- Quando n é 100, o número de operações é da ordem de 1 milhão.
- Sempre que n dobra, o tempo de execução fica multiplicado por 8.

Principais Classes de Problemas

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

- Um algoritmo de complexidade $O(2^n)$ é dito ter **complexidade exponencial**.
- Geralmente não são úteis sob o ponto de vista prático.
- Ocorrem na solução de problemas quando se usa **força bruta** para resolvê-los.
- Quando n é 20, o tempo de execução é cerca de 1 milhão. Quando n dobra, o tempo fica elevado ao quadrado.

Principais Classes de Problemas

$f(n) = O(n!)$

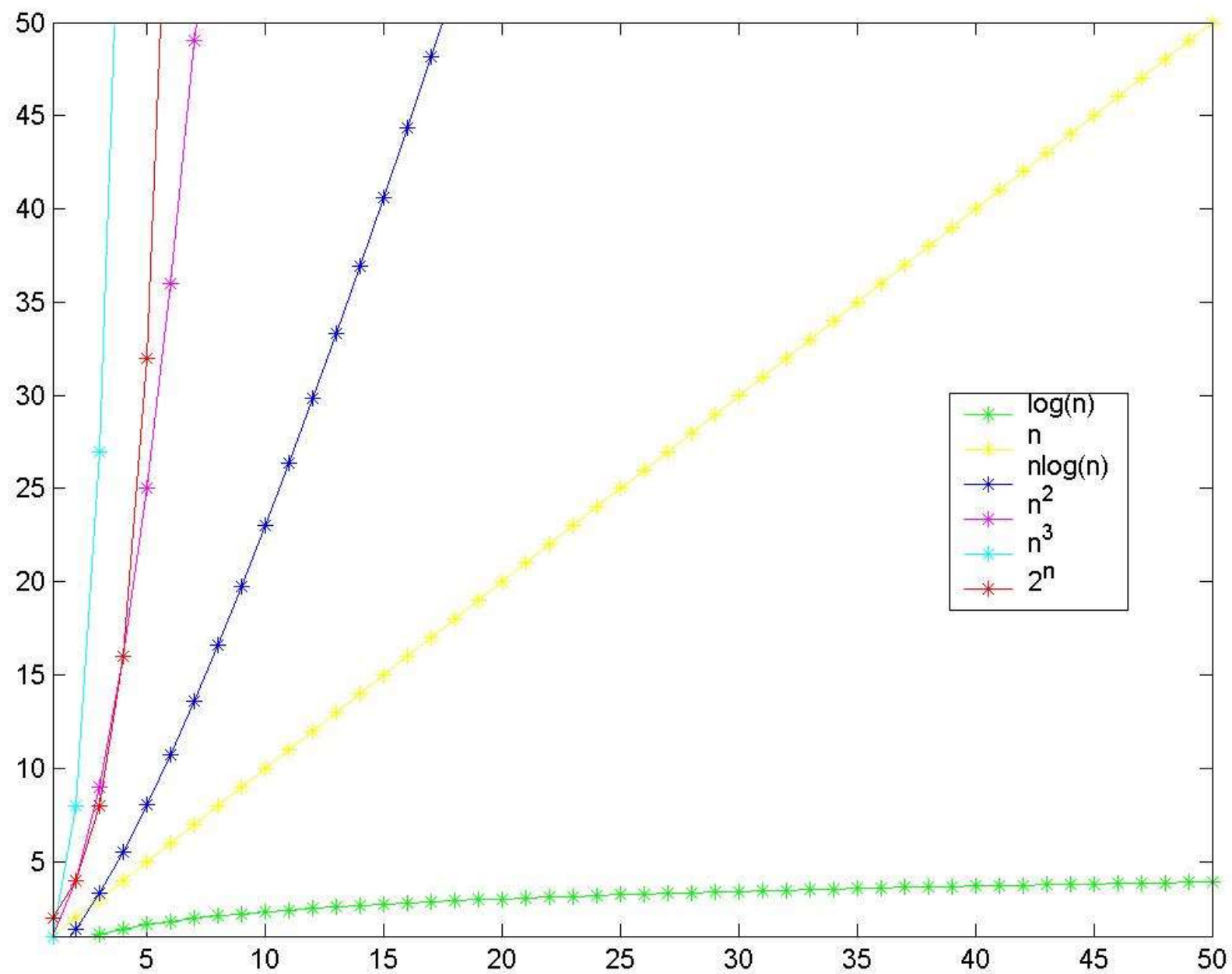
- Um algoritmo de complexidade $O(n!)$ é dito ter complexidade exponencial, apesar de $O(n!)$ ter comportamento muito pior do que $O(2^n)$.
- Geralmente ocorrem quando se usa **força bruta** para na solução do problema.
- $n = 20 \Rightarrow 20! = 2432902008176640000$, um número com 19 dígitos.
- $n = 40 \Rightarrow$ um número com 48 dígitos.

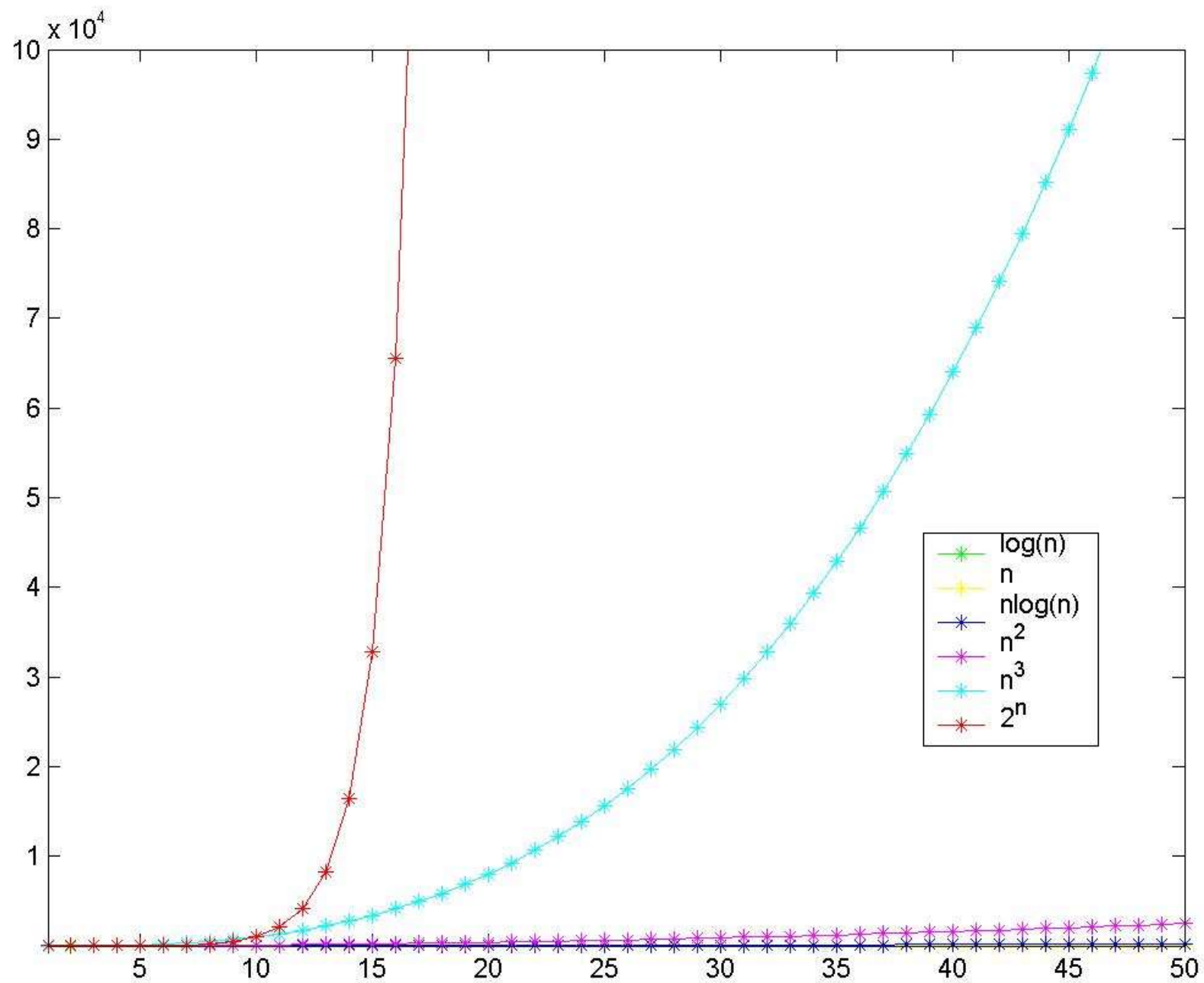
Comparação de Funções de Complexidade

Função de custo	Tamanho 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	0,0035 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0,316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
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.

Comparação de Funções de Complexidade

Função de custo de tempo	Computador atual	Computador 100 vezes mais rápido	Computador 1.000 vezes mais rápido
n	t_1	$100 t_1$	$1000 t_1$
n^2	t_2	$10 t_2$	$31,6 t_2$
n^3	t_3	$4,6 t_3$	$10 t_3$
2^n	t_4	$t_4 + 6,6$	$t_4 + 10$





Para Casa

- Ler capítulo 3 do Cormen
- Resolver os seguintes exercícios:
 1. Sejam as funções $f(n)$ e $g(n)$ assintoticamente não negativas. Usando as definições básicas da notação Θ , prove que:

$$\Theta(f(n) + g(n)) = \max(f(n), g(n))$$

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