Retomando a Notação O, θ, Ω.

Prof. José Carlos Althoff

Na análise de algoritmos na maioria das vezes usa-se o estudo de complexidade assintótica ou seja analisa-se o algoritmo quando o valor de n tende a infinito.

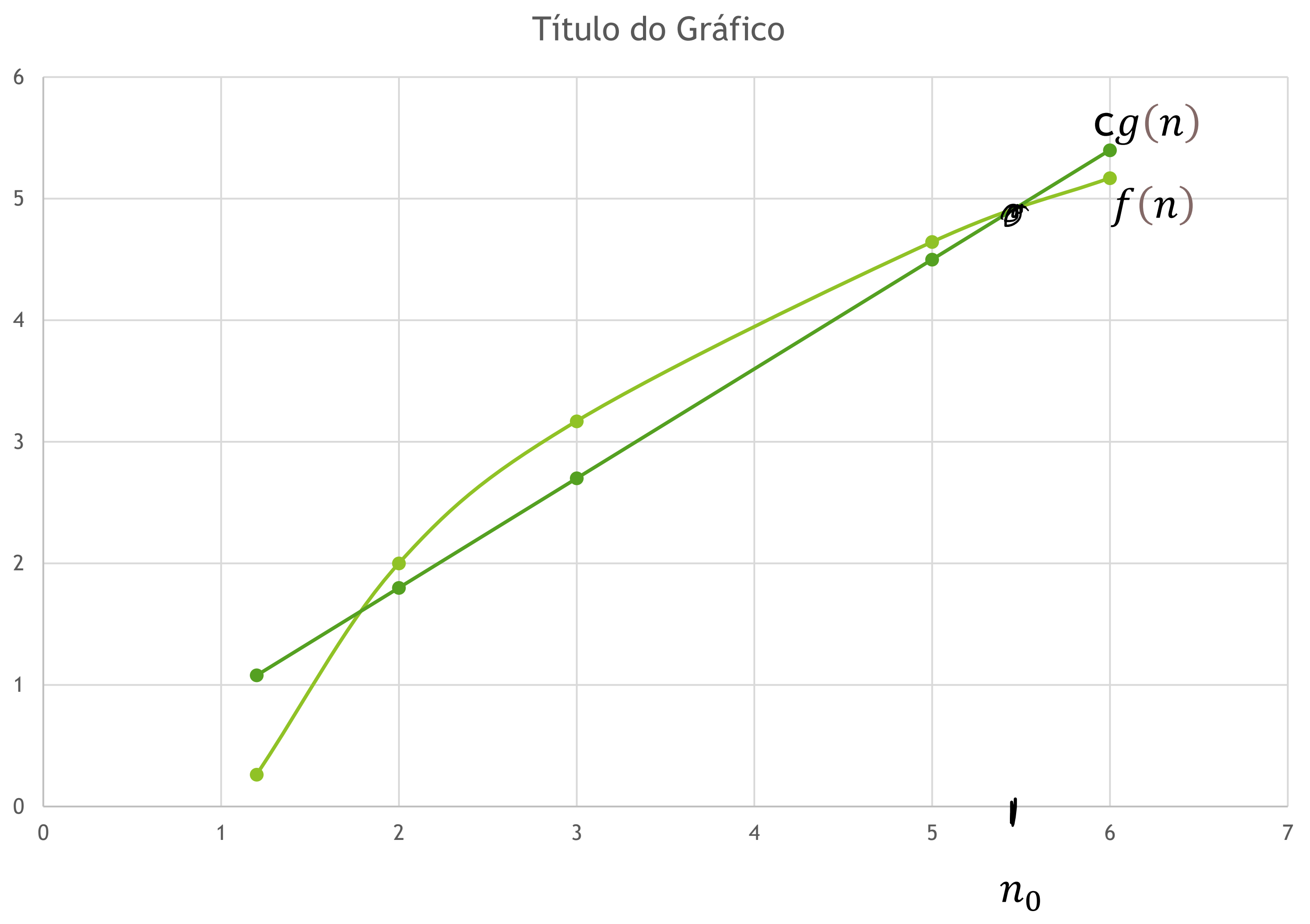
$$n \Rightarrow \infty$$

LIMITE ASSINTÓTICO SUPERIOR NOTAÇÃO "O"

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

Então existe uma constate positiva c e n_0 tal que

$$0 \leq f(n) \leq c.g(n) \ \forall n \geq n_0$$


Título do Gráfico
0
1
2
3
4
5
6
7
$cg(n)$
$f(n)$
n_0

Observações.

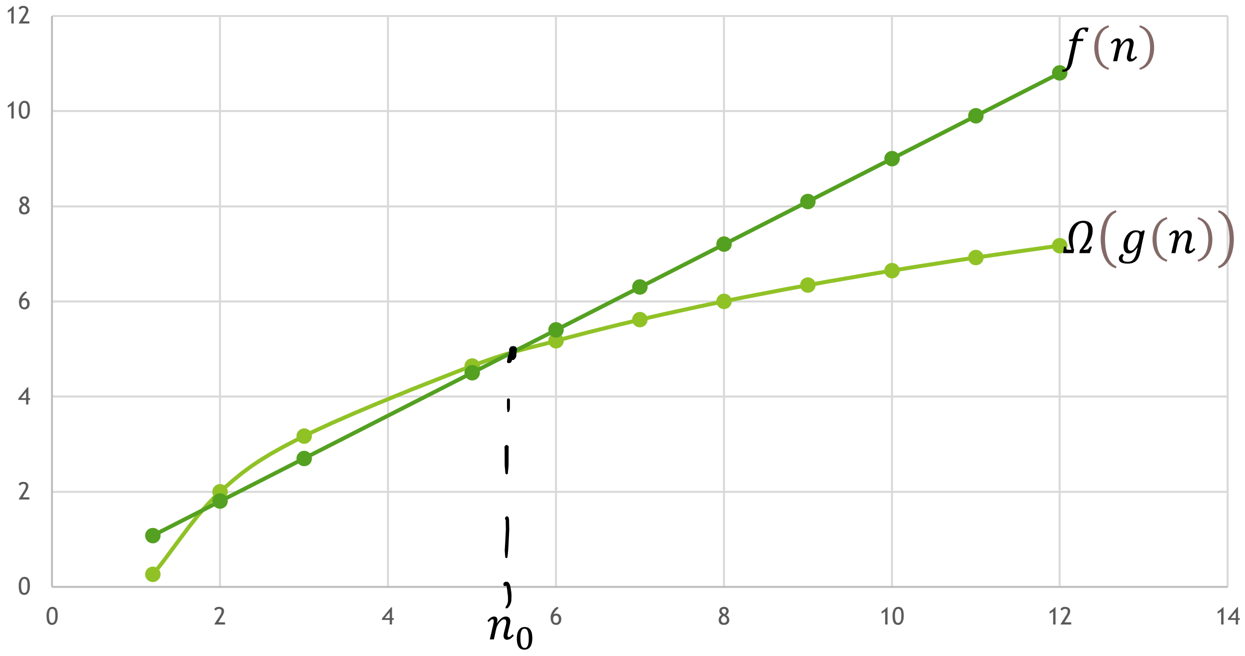
- Utilizada como limite superior, pior caso.
- $f(n) = \boldsymbol{\theta}\big(g(n)\big)$ $implica\ em$ $f(n) = \boldsymbol{O}(g(n))$ mas não o contrário.
- Podemos escrever $\theta(g(n)) \subset O\big(g(n)\big)$.

LIMITE ASSINTÓTICO INFERIOR NOTAÇÃO “Ω”

$$f(n) = \Omega(g_{(n)})$$

então existe uma constante c e n_0 tal que:

$$0 \leq C \cdot g(n) \leq f(n), \forall n \geq n_0$$



12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
$f(n)$
$\Omega(g(n))$
n_0

NOTAÇÃO θ

DEFINIÇÂO DA NOTAÇÃO θ

Agora vamos definir formalmente o que significa essa notação.

Para duas funções f(n) e g(n), dizemos que f(n) é Θ(g(n)) se

$$0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n) \quad \forall n \geq n_0$$

$c_1, c_2 \ e \ n_o > 0$

Vamos entender o que essa inequação complicada quer nos dizer.

Em um resumo bem simplista ela está dizendo que se a gente "imprensar" $f(n)$ com $g(n)$ multiplicada por duas constantes diferentes, dizemos que $f(n)$ é $\Theta(g(n))$

Exemplo.

$$\frac{1}{2}n^2 - 3n \text{ é } \theta(n^2) \text{ ?}$$

Se for $\theta(n^2)$ é necessário encontrar constantes $c_1, c_2 \; e \; n_0$ tais que:

$C_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq C_2 n^2, \forall n > n_0$ onde $c_1 > 0;\; c_2 > 0 \; e \; n_0 > 0$

Procurando as constantes:

$$C_1 \frac{n^2}{n^2} \leq \frac{1}{2}\frac{n^2}{n^2} - 3\frac{n}{n^2} \leq C_2 \frac{n^2}{n^2}$$ Então teremos:

$$C_1 \leq \frac{1}{2} - \frac{3}{n} \leq C_2$$

Procurando as constantes:

$$C_1 \leq \frac{1}{2} - \frac{3}{n}$$

Observe o lado esquerdo da inequação.
Se fizermos n variar n= 1,2,3,4,5,6,7.

Observe o lado esquerdo da inequação.

$$C_1 \leq \frac{1}{2} - \frac{3}{n}$$

Se fizermos n variar n= 1,2,3,4,5,6,7.

Quando chegarmos a n=7 teremos $c_1 \leq \frac{1}{14}$ o que atende o lado esquerdo da equação.

Agora vamos observar o lado direito da equação.

Se pensarmos que n tende ao infinito teremos:

$\frac{1}{2} - \frac{3}{\infty} \leq c_2$ a divisão de 3 por um número muito grande tende a zero. Logo o lado direito teremos que $c_2 \geq \frac{1}{2}$

- Portanto determinamos $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}$ $e\ n_0 = 7$
- $0 \leq \frac{1}{14} g(n) \leq f(n) \leq \frac{1}{2} g(n)\ \forall n \geq 7$
- O que atende a igualdade: $\frac{1}{2}n^2 - 3n = \theta(n^2)$

Portanto a igualdade é verdadeira.

O que atende a igualdade:

Note que existe outras escolhas para esta constantes c_1 e c_2,
mas o fato Importante é que a escolha existe.

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

$$\{f: N \rightarrow R^{+} \mid \exists c_1 > 0, c_2 > 0, n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$$

Exercício

- Usando a definição formal de Θ prove que $6n^3 \neq \theta(n^2)$?

Solução

Suponha que não, ou seja, suponha que $6n^3 = \theta(n^2)$.

Assim, pela definição formal da notação Θ, existem constantes positivas c1, c2 e n0 tais que

0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) para todo n ≥ n0.

Neste caso, temos que f(n) = $6n^3$, g(n) =n^2 e c1n^2 ≤ $6n^3$≤ c2n^2 .

Ao dividirmos cada termo dessa inequação por n^2 , temos: c1 ≤ 6n ≤ c2.

Não existem constantes positivas c2 > 0 e n0 tais que 6n ≤ c2 para todo n ≥ n0. Assim, a suposição $6n^3 = \theta(n^2)$ não é verdadeira, logo $6n^3 \neq \theta(n^2)$ é verdadeira.

Mais sobre a notação Assintótica de funções.

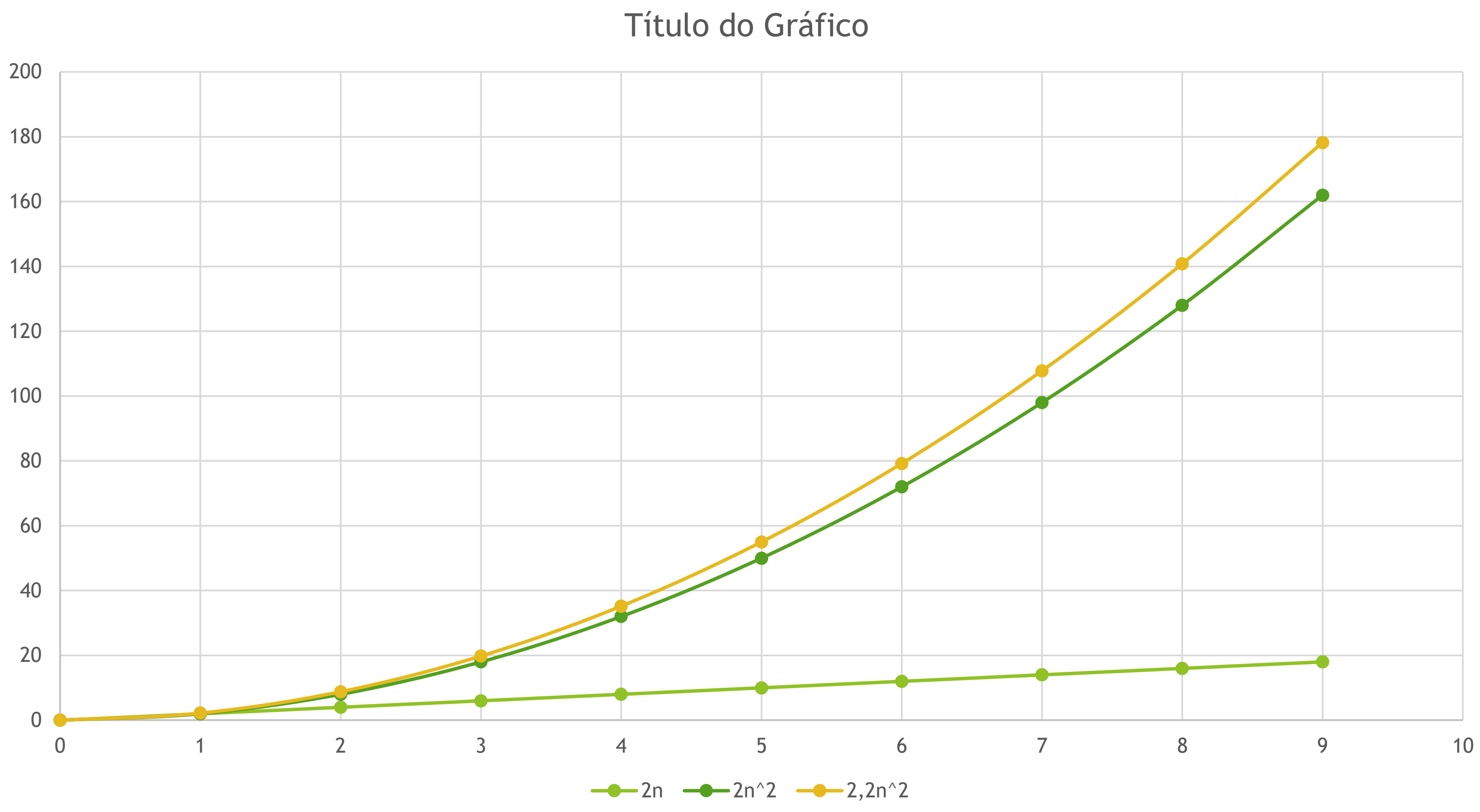
- Existem duas outras notações na análise assintótica de funções:
  - Notação o ("O" pequeno)
  - Notação ω

- 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 $2n = O(n^2)$ não é.


Título do Gráfico
200
180
160
140
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7
8
9
10
2n
2n^2
2,2n^2

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

- Formalmente a notação o é definida como:

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

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

- As definições das notações O (o grande) e o (o pequeno) são similares.
  - A diferença principal é que em $f(n) = O(g(n))$, a expressão $0 \leq f(n) \leq cg(n)$ 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:

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

- 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.

Comparação de programas

- Podemos avaliar programas comparando as funções de complexidade, negligenciando as constantes de proporcionalidade.

- Um programa com tempo de execução $O(n)$ é melhor que outro com tempo $O(n^2)$.
  - Porém, as constantes de proporcionalidade podem alterar esta consideração.

- Exemplo: um programa leva $100n$ unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
  - Depende do tamanho do problema.
  - Para $n < 50$, o programa com tempo $2n^2$ é melhor do que o que possui tempo $100n$.
  - Para problemas com entrada de dados pequena é preferível usar o programa cujo tempo de execução é $O(n^2)$.
  - Entretanto, quando n cresce, o programa com tempo de execução $O(n^2)$ leva muito mais tempo que o programa $O(n)$.

Complexidade Constante

- $f(n) = O(1)$
  - O uso do algoritmo independe do tamanho de n.
  - As instruções do algoritmo são executadas um número fixo de vezes.

O que significa um algoritmo ser $O(2)$ ou $O(5)$?

Complexidade Logarítmica

- $f(n) = O(\log n)$
  - Ocorre tipicamente em algoritmos que resolvem um problema transformando-o em problemas menores.
  - Nestes casos, o tempo de execução pode ser considerado como sendo menor do que uma constante grande.

- Supondo que a base do logaritmo seja 2:
  - Para $n = 1\,000$, $\log_2 \approx 10$.
  - Para $n = 1\,000\,000$, $\log_2 \approx 20$.

- Exemplo:
  - Algoritmo de pesquisa binária.

Complexidade Linear

- $f(n) = O(n)$
  - Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
  - Esta é a melhor situação possível para um algoritmo que tem que processar/produzir n elementos de entrada/saída.
  - Cada vez que n dobra de tamanho, o tempo de execução também dobra.

- Exemplos:
  - Algoritmo de pesquisa seqüencial.
  - Algoritmo para teste de planaridade de um grafo.

Complexidade Linear Logarítmica

- $f(n) = O(n \log n)$
  - Este tempo de execução ocorre tipicamente em algoritmos que resolvem um problema quebrando-o em problemas menores, resolvendo cada um deles independentemente e depois agrupando as soluções.
  - Caso típico dos algoritmos baseados no paradigma *divisão-e-conquista*.

- Supondo que a base do logaritmo seja 2:
  - Para $n = 1\,000\,000$, $\log_2 \approx 20\,000\,000$.
  - Para $n = 2\,000\,000$, $\log_2 \approx 42\,000\,000$.

- Exemplo:
  - Algoritmo de ordenação MergeSort.

Complexidade Quadrática

- $f(n) = O(n^2)$
  - Algoritmos desta ordem de complexidade ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro do outro
  - Para $n = 1\,000$, o número de operações é da ordem de $1\,000\,000$.
  - Sempre que n dobra o tempo de execução é multiplicado por 4.
  - Algoritmos deste tipo são úteis para resolver problemas de tamanhos *relativamente* pequenos.

- Exemplos:
  - Algoritmos de ordenação simples como seleção e inserção.

Complexidade Cúbica

- $f(n) = O(n^3)$
  - Algoritmos desta ordem de complexidade geralmente são úteis apenas para resolver problemas *relativamente* pequenos.
  - Para $n = 100$, o número de operações é da ordem de $1\,000\,000$
  - Sempre que n dobra o tempo de execução é multiplicado por 8.
  - Algoritmos deste tipo são úteis para resolver problemas de tamanhos *relativamente* pequenos.

- Exemplo:
  - Algoritmo para multiplicação de matrizes.

Complexidade Exponencial

- $f(n) = O(2^n)$
  - Algoritmos desta ordem de complexidade não são úteis sob o ponto de vista prático.
  - Eles ocorrem na solução de problemas quando se usa a *força bruta* para resolvê-los.
  - Para $n = 20$, o tempo de execução é cerca de $1\,000\,000$.
  - Sempre que n dobra o tempo de execução fica elevado ao quadrado.

- Exemplo:
  - Algoritmo do Caixeiro Viajante

Complexidade Fatorial

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

Um algoritmos de complexidade $O(n!)$ é pior do complexidade Exponencial. No entanto, alguns autores acabam falando que ele tem complexidade exponencial.

Apesar de $O(n!) ter\ um\ coportamento\ muito\ pior\ que\ O(2^n)$

Geralmente ocorre quando se usa força bruta na solução do problema.

Considerando:

- $n = 20$, temos que $20! = 2432902008176640000$, um número com 19 dígitos.
- $n = 40$ temos 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	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
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 anos
3^n	0,059 s	58 min	6,5 anos	3855 sec	10^8 sec	10^{13} sec

Hierarquias de funções

A seguinte hierarquia de funções pode ser definida do ponto de vista assintótico:

$$1 \prec \log\log n \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}}$$

onde ϵ e c são constantes arbitrárias com $0 < \epsilon < 1 < c$.