Principais expressões

- 1 (constante): quando a execução independe dos dados de entrada
- n: quando é necessário processar todos elementos, cada um em tempo constante
- n^2: quando, para cada elemento, é necessário verificar todos os demais
- n^3: quando é necessário verificar combinações triplas dos dados
- $\log_2 n$: quando o problema é reduzido em 2 subproblemas com a metade do tamanho original e apenas 1 deles é processado.
- $n \log_2 n$: quando, para cada elemento, o problema é subdividido
- 2^n: normalmente, quando verifica-se todas as possíveis alternativas

Alguns padrões (para identificar)

- Uma sequência sem laço ou recursão conta passo constante (1)

```
/* bloco com número de passos constante */
```

- Um único laço com n passos internos constante: linear (n)

```
for(i=0; i < n; i++)
    /* bloco com número de passos constante */
```

- Dois laços de tamanho n aninhados: quadrático (n^2)

```
for(i=0; i < n; i++)
    for(j=0; j < n; j++)
        /* bloco com número de passos constante */
```

Alguns padrões (para identificar)

- Um laço interno dependente de um externo: quadrático (n^2)

  (requer uso de somatório duplo)

```
for(i=0; i < n; i++)
    for(j=0; j < i ; j++)
        /* bloco com número de passos constante */
```

- Quando divide o problema pela metade: logarítmico ($log_2 n$)

```
if (test)
    subprob(0, n/2);    /* do início à metade */
else
    subprob(n/2+1, n-1) /* da metade ao fim */
```

Pseudocódigo

```
Algoritmo recebe (n)
Entrada: n é um número inteiro.
Declare:
   x, y, raiz: Inteiros;
x ← y ← 0; //Inicializando os pontos no centro
raiz = arredondar( sqrt ( n ) );
Se raiz é par então
   Se n > raiz² + raiz então
      y ← - raiz/2;
      x ← (x - raiz/2) +(n – raiz²);
   Senão
      y ← y - raiz/2 + ( n - (raiz² + raiz) );
      x ← x + raiz/2;
   Fim se
Senão //raiz é impar
   Se n > raiz² + raiz então
      y ← y + raiz/2 + 1;
      x ← (x + raiz/2) - (n – raiz²);
   Senão
      y ← y + raiz/2 + 1 - ( n - (raiz² + raiz) );
      x ← x - raiz/2 - 1;
   Fim se
Fim se
Retorne x, y;
```

O pseudocódigo sempre deve considerar:

- Expressões;
- Declarações de variáveis;
- Declarações de métodos;
- Estruturas de decisão;
- Estruturas de Repetição;
- Indexação de arranjos;
- Chamadas de métodos;
- Retorno de variáveis;

Notação assintótica

- Quando observamos tamanhos de entrada grandes o suficiente para tornar relevante apenas a ordem de crescimento do tempo de execução, estamos estudando a eficiência assintótica dos algoritmos;

Notação assintótica

- **Quando observamos tamanhos de entrada grandes o suficiente** para tornar relevante apenas a ordem de crescimento do tempo de execução, **estamos estudando a eficiência assintótica dos algoritmos**;

- Ou seja, estamos preocupados com a maneira como o tempo de execução de um algoritmo aumenta, à medida que o tamanho da entrada da entrada aumenta INDEFINIDAMENTE;

Notação assintótica

- **Quando observamos tamanhos de entrada grandes o suficiente** para tornar relevante apenas a ordem de crescimento do tempo de execução, **estamos estudando a eficiência assintótica dos algoritmos**;

- Ou seja, **estamos preocupados com a maneira como o tempo de execução de um algoritmo aumenta**, à medida que o tamanho da entrada da **entrada aumenta <u>INDEFINIDAMENTE</u>**;

- Em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas, exceto as muito pequenas.

Análise assintótica - outro exemplo

Para valores enormes de n, as funções

$$n^2, \quad \frac{3}{2}n^2, \quad 9999n^2, \quad \frac{n^2}{1000}, \quad n^2 + 100n$$

crescem todas com a mesma velocidade e portanto são todas “equivalentes”. Nesse estudo, as funções são classificadas em “ordens”; todas as funções de uma mesma ordem são equivalentes. As cinco funções acima, por exemplo, pertencem à mesma ordem.

Notação assintótica

- A análise de um algoritmo geralmente considera com apenas algumas operações elementares.
  - Comparações;
  - Atribuições;

Notação assintótica

- A análise de um algoritmo geralmente **considera com apenas algumas operações elementares**.
  - Comparações;
  - Atribuições;

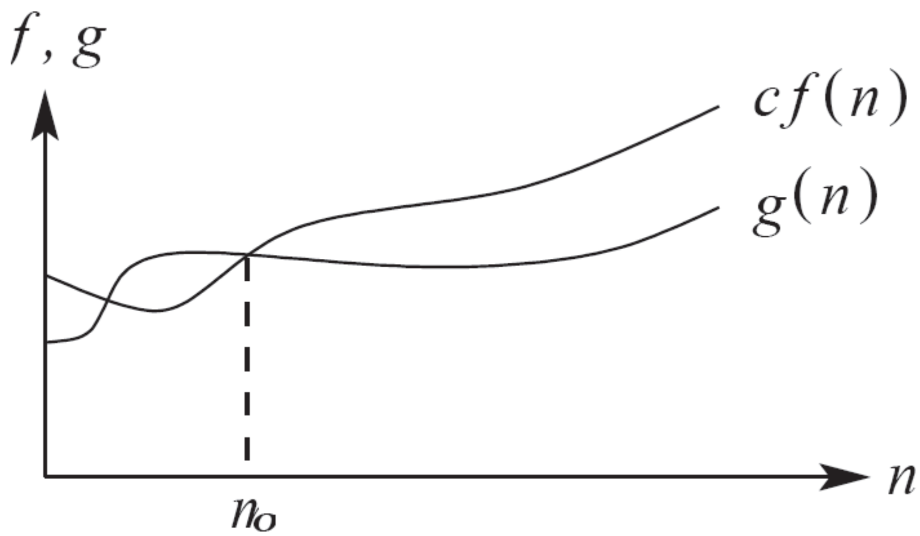
- Este fator **varia de autor para autor** na literatura;

  - Na prática, **as comparações são mais demoradas que atribuições**. Sendo assim, alguns analistas consideraram apenas comparações na análise de algoritmos.

Notação assintótica

- A medida de custo ou medida de complexidade relata o crescimento assintótico da operação considerada.

- Definição: Uma função $f(n)$ domina assintoticamente outra função $g(n)$ se existem duas constantes positivas
  - c e n_o
- tais que, para qualquer
  - $n >= n_o$,
- temos
  - $g(n) <= c \cdot f(n)$



Notação assintótica

- Escrevemos $g(n) = O(f(n))$ ou $g(n) \in O(f(n))$ para expressar que $f(n)$ domina assintoticamente $g(n)$. Encontramos leituras nos modos:
  - $g(n)$ é da ordem no máximo $f(n)$; *// formal*
  - $g(n)$ é O de $f(n)$; *// informal*
  - $g(n)$ é igual a O de $f(n)$; *// informal*
  - $g(n)$ pertence a O de $f(n)$; *// formal*

Notação assintótica

- Escrevemos ***g(n) = O(f(n))*** ou ***g(n) ∈ O(f(n))*** para expressar que *f(n)* domina assintoticamente *g(n)*. Encontramos leituras nos modos:
  - *g(n)* é da ordem no máximo *f(n)*; ***// formal***
  - *g(n)* é *O* de *f(n)*; ***// informal***
  - *g(n)* é igual a *O* de *f(n)*; ***// informal***
  - *g(n)* pertence a *O* de *f(n)*; ***// formal***

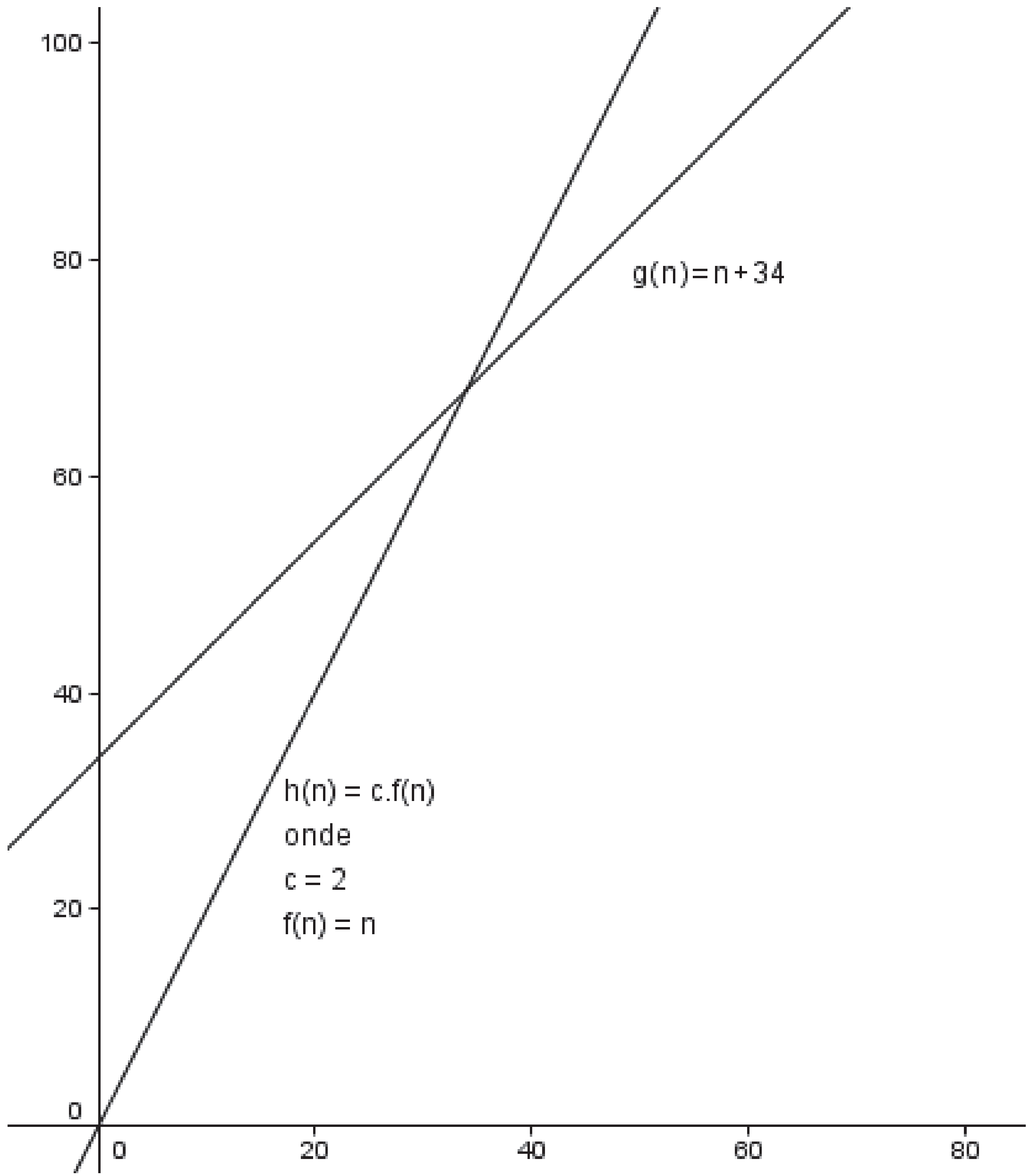
- Exemplo: quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes
  - c e n_o
- tais que, para valores de
  - $n >= n_o$,
- temos:
  - $T(n) <= c.n^2$

Notação assintótica

- Exemplo:
  - $f(n) = n$
  - $g(n) = n+34$
  - $g(n) \in O(f(n))$???
  - ou
  - $n+34 \in O(n)$???

Notação assintótica

- Exemplo:
  - $f(n) = n$
  - $g(n) = n+34$
  - $g(n) \in O(f(n))$???
  - *ou*
  - $n+34 \in O(n)$???



Notação assintótica

- Uma visão um pouco diferente do que a já vista por vocês em Estrutura de Dados I...
- Questão 1:
  - n é $O(n^2)$?

Notação assintótica

- Questão 1:
  - n é $O(n^2)$?
  - Sim.
  - Para $n >= 1$,
    - $n <= n^2$.

Notação assintótica

- Questão 2:
  - n^2 é $O(n)$?

Notação assintótica

- Questão 2:
  - $n^2 \in O(n)$?
  - Não.
  - Suponha:

$$\exists c, n_0$$

$$\forall n \geq n_0$$

$$n^2 \leq c \cdot n$$

Notação assintótica

- Questão 2:
  - $n^2 \in O(n)$?
  - Não.
  - Suponha:

$$\exists c, n_0$$

$$\forall n \geq n_0$$

$$n^2 \leq c \cdot n$$

Absurdo, pois
c é uma constante e
n assume valores até o infinito

$$\boxed{\begin{array}{l} n^{\cancel{2}} \leq c \cdot \cancel{n} \\ n \leq c \end{array}}$$

Portanto, $\nexists c, n_0$

Notação assintótica

- Operações básicas com a notação O

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times O(f(n)) &= O(f(n)) \qquad c = constante \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n)) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Exercícios

(a) $10^{56} \cdot n^2 \in O(n^2)$?

(b) $10^{56} \cdot n^2 \in O(n^3)$?

(c) $10^{56} \cdot n^2 \in O(n)$?

(d) $2^{n+1} \in O(2^n)$?

(e) $2^{2n} \in O(2^n)$?

(f) $n \in O(n^3)$?

Prove ou disprove:

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

2- $3^n = O(2^n)$

3- $\log_2 n = O(\log_3 n)$

4- $\log_3 n = O(\log_2 n)$

5- $\log_2 n = O(n)$

6- $100 \log n - 10n + 2n \log n = O(n \log n)$

Prove os disprove as afirmações:

1. $10n = O(n)$
2. $10n^2 = O(n)$
3. $n^2 + 200n + 300 = O(n^2)$
4. $n^2 - 200n - 300 = O(n^2)$
5. $n^2 - 200n - 300 = O(n)$
6. $\frac{3}{2}n^2 + \frac{7}{2}n - 4 = O(n)$
7. $\frac{3}{2}n^2 + \frac{7}{2}n - 4 = O(n^2)$
8. $n^3 - 999999n^2 - 1000000 = O(n^2)$

Outras notações

- Assim como a notação *O* fornece uma maneira assintótica de dizer que uma função é “menor ou igual a”outra, existem outras notação que fornecem outras conclusões sobre a complexidade de algoritmos;

Outras notações

- Assim como a notação O fornece uma maneira assintótica de dizer que uma função é "menor ou igual a"outra, existem outras notação que fornecem outras conclusões sobre a complexidade de algoritmos;
  - Θ

Outras notações

- Assim como a notação O fornece uma maneira assintótica de dizer que uma função é "menor ou igual a" outra, existem outras notação que fornecem outras conclusões sobre a complexidade de algoritmos;
  - Θ
  - Ω

Outras notações

- Assim como a notação O fornece uma maneira assintótica de dizer que uma função é “menor ou igual a”outra, existem outras notação que fornecem outras conclusões sobre a complexidade de algoritmos;
  - Θ
  - Ω
  - ω

Outras notações

- Assim como a notação O fornece uma maneira assintótica de dizer que uma função é "menor ou igual a"outra, existem outras notação que fornecem outras conclusões sobre a complexidade de algoritmos;
  - Θ
  - Ω
  - ω
  - o

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - ‘O’ define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - ‘O’ define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

- Exemplos:

$$n^4 \in \Omega(n^3)$$

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - ‘O’ define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

- Exemplos:

$$n^4 \in \Omega(n^3)$$

$$n \in \Omega(1)$$

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - ‘O’ define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

- Exemplos:

$$n^4 \in \Omega(n^3)$$

$$n \in \Omega(1)$$

$$3 \cdot \log(n) \in \Omega(\log(n))$$

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - 'O' define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

- Exemplos:

$$n^4 \in \Omega(n^3)$$

$$n \in \Omega(1)$$

$$3 \cdot \log(n) \in \Omega(\log(n))$$

$$1 \in \Omega(1)$$

Notação Ω

- A notação Ω é bem parecida com a notação O;
  - ‘O’ define um limite assintótico superior, e;
  - Ω define um limite assintótico inferior.

- Exemplos:

$$n^4 \in \Omega(n^3)$$

$$n \in \Omega(1)$$

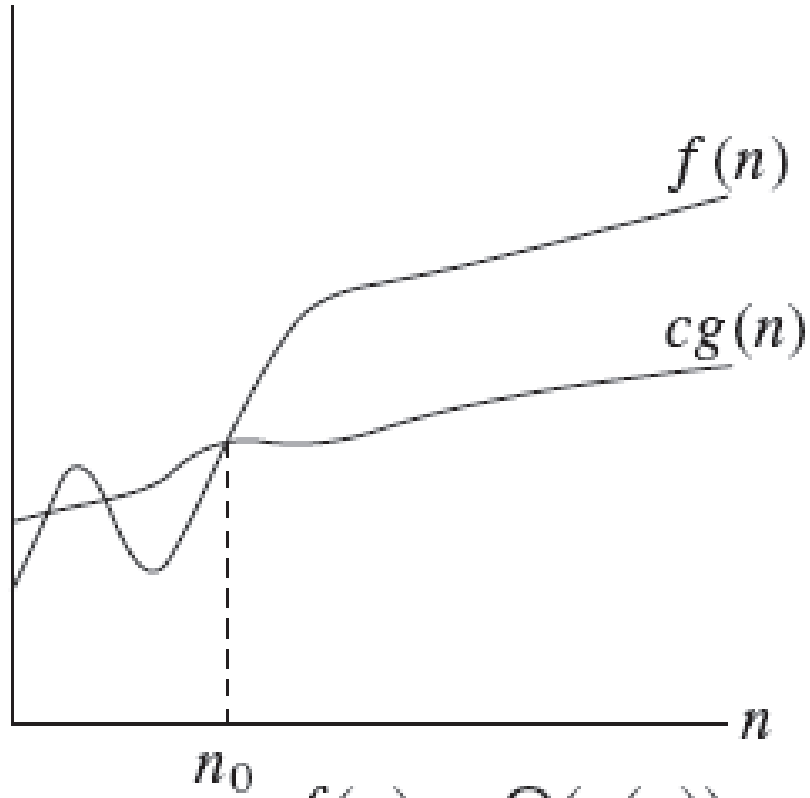
$$3 \cdot \log(n) \in \Omega(\log(n))$$

$$1 \in \Omega(1)$$

$$n! \in \Omega(2^n)$$

Notação Ω

- Limite assintótico inferior

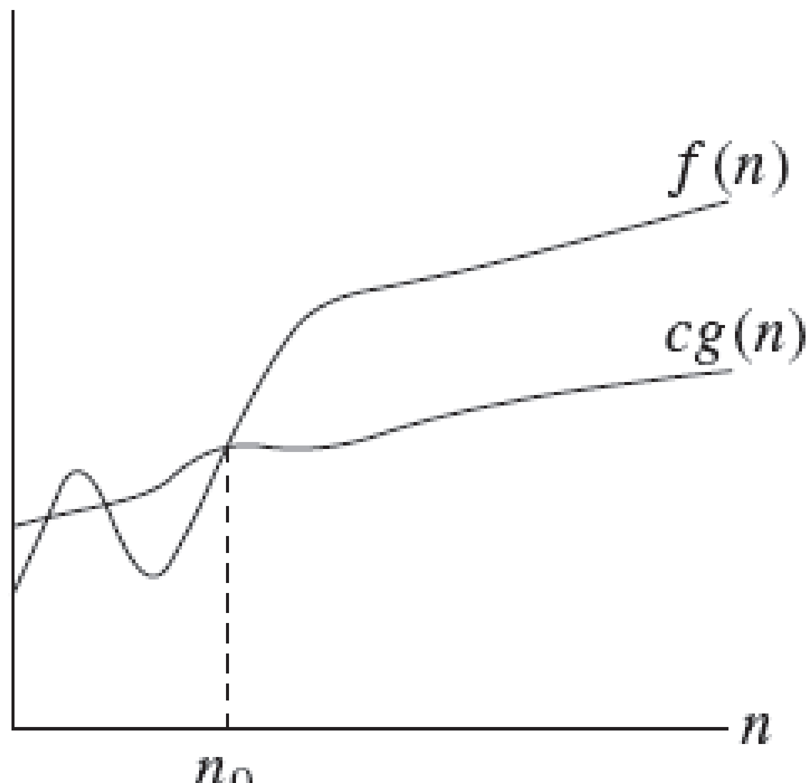


$f(n) = \Omega(g(n))$

$$\Omega(g(n)) = \{f(n):\ \exists\ c\ e\ n_0\ > 0$$
$$|\ 0 \leq c \cdot g(n) \leq f(n) \quad \forall \quad n \geq n_0\}$$

Notação Ω

- Limite assintótico inferior



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

Notação Ω

- Na prática a notação Ω não é vista sozinha em análises de algoritmos;

Notação Ω

- Na prática a notação Ω não é vista sozinha em análises de algoritmos;
    - Pelo motivo de não interessar para a análise de algoritmos;

Notação Ω

- Na prática a notação Ω não é vista sozinha em análises de algoritmos;
  - Pelo motivo de não interessar para a análise de algoritmos;
  - A notação O possui sua importância, pois o programador conclui que seu algoritmo é no máximo tão complexo a uma função.

Notação Ω

- Na prática a notação Ω não é vista sozinha em análises de algoritmos;
  - Pelo motivo de não interessar para a análise de algoritmos;
  - A notação O possui sua importância, pois o programador conclui que seu algoritmo é no máximo tão complexo a uma função.
  - Mas no mínimo tão complexo, como a notação Ω descreve, não é importante para conclusões práticas sobre algoritmos.

Notação Ω

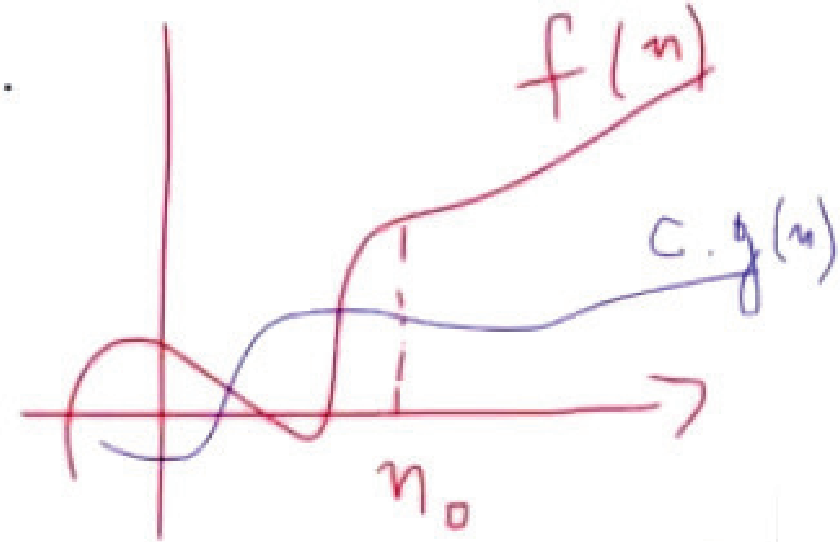
- Na prática a notação Ω não é vista sozinha em análises de algoritmos;
  - Pelo motivo de não interessar para a análise de algoritmos;
  - A notação O possui sua importância, pois o programador conclui que seu algoritmo é no máximo tão complexo a uma função.
  - Mas no mínimo tão complexo, como a notação Ω descreve, não é importante para conclusões práticas sobre algoritmos.
- Ω vem na maioria das vezes acompanhada a notação Θ;
  - Como um complemento na análise, nunca sozinha...

Ordem Ω

Em outras palavras, $f = \Omega(g)$ se existe um número positivo c e um número n_0 tais que

$$f(n) \geq c.g(n)$$

para todo n maior que n_0.



Prove ou disprove:

1- Seja $\binom{n}{k}$ o número de combinações de n objetos tomados k a k. Mostre $\binom{n}{2} = \Omega(n^2)$.

2- Prove que $100 \log n - 10n + 2n \log n$ está em $\Omega(n \log n)$.

3- É verdade que $2n + 1$ está em $\Omega(n)$?

Algoritmo ótimo

Se um algoritmo tem uma complexidade igual á cota inferior do problema, ele é **assintoticamente ótimo** ou simplesmente **ótimo**.

Notação θ

- Conhecida também como “limite firme” ou “limite assintoticamente restrito”.

Notação θ

- Conhecida também como "limite firme" ou "limite assintoticamente restrito".

- A notação O, apesar de fornecer informações sobre a complexidade do algoritmo, nem sempre nos revela algo importante;

Notação θ

- Conhecida também como **"limite firme" ou "limite assintoticamente restrito".**

- **A notação O**, apesar de fornecer informações sobre a complexidade do algoritmo, **nem sempre nos revela algo importante**;

- Não faz sentido, para algum algoritmo, dizer que suas complexidade é por exemplo *O(n!)*.
  - Ou faz?

Notação θ

- Conhecida também como **"limite firme" ou "limite assintoticamente restrito".**

- **A notação O**, apesar de fornecer informações sobre a complexidade do algoritmo, **nem sempre nos revela algo importante**;

- Não faz sentido, para algum algoritmo, dizer que suas complexidade é por exemplo *O(n!)*. Ou faz?

- Exemplos da falta de precisão de O:

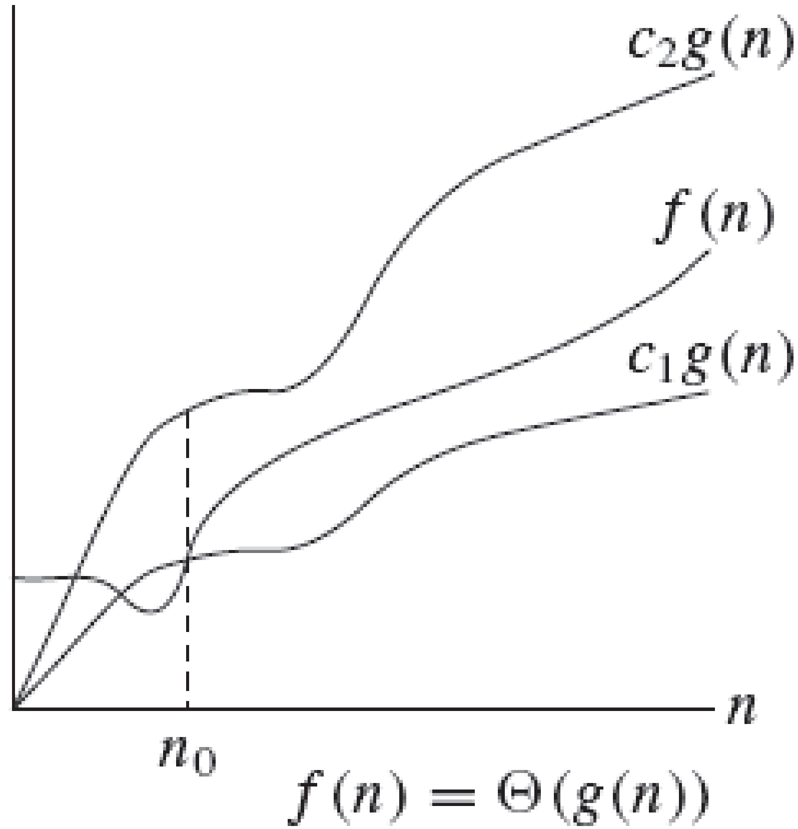
$$n \in O(n^3)$$
$$n \in O(n^4)$$
$$n \in O(n^5)$$
$$n \in O(n^{1000})$$
$$n \in O(2^n)$$
$$n \in O(n!)$$

Notação θ

- Uma função *f(n)* pertence ao conjunto $\theta(g(n))$ se existem constantes positivas n_0, c_1 e c_2

Notação θ

- Uma função *f(n)* pertence ao conjunto *θ(g(n))* se existem constantes positivas n_0, c_1 e c_2 tais que ela possa ser "imprensada" entre
  c1.g(n) e *c2.g(n)*,
  para um valor de *n* suficientemente grande.



$$\Theta(g(n)) = \{f(n) : \ \exists\ c_1, c_2 \ e \ n_0 \ > 0$$
$$\mid 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \quad \forall \quad n \ge n_0\}$$

Ordem Θ

Dizemos que as funções f e g estão na mesma ordem e escrevemos $f = \Theta(g)$ se $f = O(g)$ e $f = \Omega(g)$.

Em outras palavras, $f = \Theta(g)$ significa que existe números positivos c e d tais que

$$c.g(n) \leq f(n) \leq d.g(n)$$

para todo n suficientemente grande.

Notação θ

$$\Theta(g(n)) = \{f(n):\ \exists\ c_1, c_2\ e\ n_0\ > 0\ \mid 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)\quad \forall\ \ n \ge n_0\}$$

- Exemplo: $$\frac{n^2}{2} - 3n \in \Theta(n^2)$$

- Para isso, devemos definir constantes c_1, c_2 e n_0 tais que: $$c_1 n^2 \le \frac{1}{2} n^2 - 3n \le c_2 n^2$$

- Encontre constantes que satisfaça as duas desigualdades...

Notação θ

- Exemplo de constantes:

$$\left(c_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 n^2 \right)$$ Dividindo por n^2 ...

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

Notação θ

- Exemplo de constantes:

$$\left(c_1 n^2 \le \frac{1}{2} n^2 - 3n \le c_2 n^2\right)$$ Dividindo por n^2 ...

$$= c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2 \longrightarrow c_1 = \frac{1}{14}$$

$$c_2 = \frac{1}{2}$$

- Portanto, se existem tais constantes

$$\frac{n^2}{2} - 3n \in \Theta(n^2) \qquad n_0 = 7$$

Vamos relembrar o custo do algoritmo… $T(n) = n^2 + 3n$.

Vamos ver em que notação ele pode se encaixar, sabendo que $g(n)$ seria a **ordem de crescimento** (parte importante) do nosso custo; no caso, n^2.

Testamos primeiro se ele encaixa na função $\Theta(n^2)$. Vamos substituir $f(n)$ e $g(n)$ (naquela função onde diz **A notação** Θ) pelos valores que conhecemos.

$c_1 n^2 \leq n^2 + 3n \leq c_2 n^2$

Se dividirmos tudo por n^2, obteremos:

$c_1 \leq 1 + \frac{3}{n} \leq c_2$

Agora separaremos as inequações.

Inequação 1: $c_1 \leq 1 + \frac{3}{n}$

Inequação 2: $1 + \frac{3}{n} \leq c_2$

Para satisfazer a **Inequação 1**, podemos quase automaticamente perceber que para qualquer $n \geq 1$, é válido $c_1 = 1$ (ora, por mais que $\frac{3}{n}$ chegue perto de 0, sempre ainda vamos ter a constante 1 adicionada a ele). Para satisfazer a **Inequação 2**, podemos perceber facilmente que para qualquer $n \geq 1$, é válido $c_2 = 4$ (a função só tende a diminuir a partir que n vai aumentando e com $n = 1$, $c_2 = 4$). Com isso, agora chegamos as três constantes que precisávamos.

n_0 (o menor valor de n) = 1; $c_1 = 1$; $c_2 = 4$.

Logo, concluímos que $f(n) = n^2 + 3n = \Theta(n^2)$. Uma função que pertence a Θ, tem um **limite assintótico superior** e **inferior** e, portanto, pertenceria também a $O(n^2)$ e $\Omega(n^2)$, mas nem é necessário testar os outros valores porque já identificamos nossa função como *"theta de ene ao quadrado"*, que é a função mais "retinha" que podemos esperar.

Notação θ

- Observação:

$$f(x) \in \Theta(g(x))$$

$$sse$$

$$f(x) \in O(g(x))$$

$$e$$

$$f(x) \in \Omega(g(x))$$

Prove ou disprove:

1. $9999n^2 = \Theta(n^2)$
2. $(3/2)n^2 + (7/2)n - 4 = \Theta(n^2)$
3. $n^2/1000 - 999n = \Theta(n^2)$
4. $\log_2 n + 1 = \Theta(\log_{10} n)$