Análise de Complexidade de Algoritmos

Estrutura de Dados

Prof. Carla Koike - CIC

Porque Analisar um algoritmo?

- Para avaliar sua performance e comparar diferentes algoritmos
- Mas analisar o que?
  - tempo de execução e uso de memória
  - pior caso e caso típico, dependendo das entradas
- Análise de algoritmos compara *ALGORITMOS* e não programas

Tipos de problemas na análise de algoritmos

- Análise de um algoritmo particular

  Qual é o custo de usar um dado algoritmo na solução de um problema específico?

- Características que devem ser investigadas:
  - número de vezes que cada parte do algoritmo deve ser executada (ou o tempo de execução)

    complexidade do tempo

  - quantidade de memória necessária

    complexidade de espaço

Tipos de problemas na análise de algoritmos, cont.

- Análise de uma classe de algoritmos.

  Qual é o algoritmo de menor custo possível para resolver um problema particular?

- Toda uma família de algoritmos é investigada, e procura-se identificar um que seja o melhor possível.
- Colocam-se limites para a complexidade computacional dos algoritmos pertencentes à classe

Executando um algoritmo

- A complexidade de um algoritmo pode ser analisada a partir de uma implementação executando em um computador.
  - A eficiência do programa depende da linguagem (compilada ou interpretada)
  - depende do sistema operacional
  - depende do hardware (quantidade de memória, velocidade do processador e arquitetura)
- útil nas comparações entre programas em máquinas específicas, logo não somente os custos do algoritmo são comparados mas também os custos não aparentes (alocação de memória, indexação, carga de arquivos, etc)

Análise pelo modelo matemático

- Não depende do computador nem da implementação
- O custo das operações mais significativas deve ser especificado, e algumas operações são desprezadas.
- É possível analisar a complexidade do algoritmo dependendo dos dados de entrada

Exemplo 1

- Qual o custo do algoritmo?

```
soma <- 0
para i de 0 até n faça soma <- soma + i
```

- Somente *atribuições* são operações relevantes:
  - duas atribuições (soma e i recebem zero)
  - duas atribuições a cada laço (soma recebe soma + i, e i recebe i+1)
  - Total de atribuições: 2 + 2n
- Função que define a complexidade de tempo do algoritmo é g*(n)* = *2* + *2n*

Exemplo 2

- Algoritmo: dividir elementos de uma matriz quadrada por uma constante k

```
soma <- 0
para i de 0 ate n faça
   para j de 0 ate n faça
      a[i,j] <- a[i,j]/k
```

- Número de atribuições:
  - duas atribuições (soma e i recebem 0)
  - para o laço externo: uma atribuição (i recebe i+1), n vezes
  - para o laço interno: duas atribuições (a e j), n^2 vezes
- Complexidade de tempo: $g(n) = 2 + n + 2n^2$

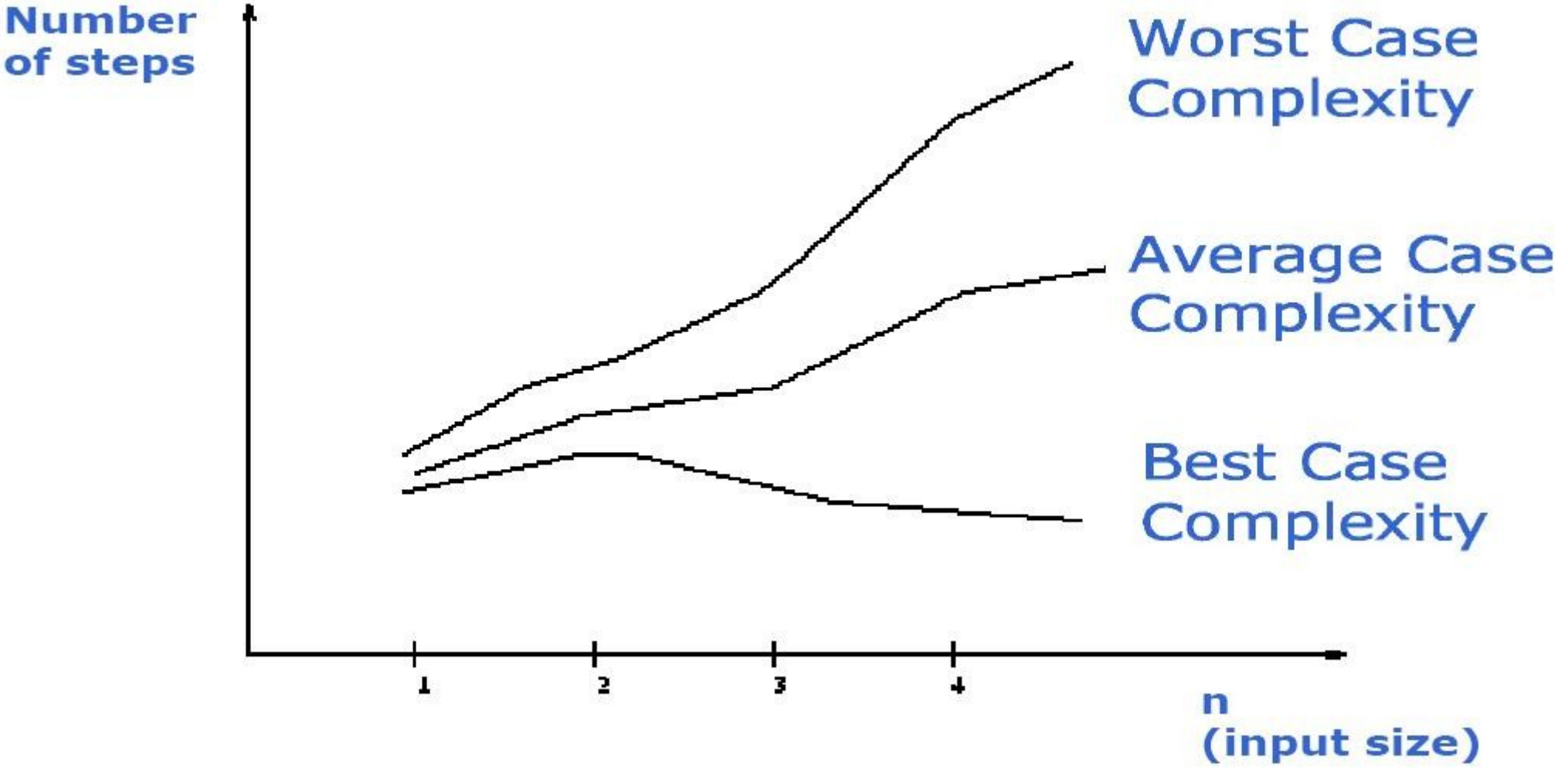
Análise da função complexidade

- Considere a função:
  - $g(n) = 2 + n + 2n^2$
- Crescimento dos termos em função de n:
  - para n= 1, $g(n) = 2 + 1 + 2$, todos os termos tem o mesmo peso
  - para n = 1000, $g(n) = 2 + 1000 + 2000000$, o último termo é dominante, portanto os dois primeiros podem ser desprezados

O melhor, o médio e o pior caso

- **Melhor caso**: função definida pelo menor número de passos executados para qualquer instância de tamanho n.
- **Pior caso**: função definida pelo maior número de passos executados para qualquer instância de tamanho n.
- **Caso médio**: função definida pela média do número de passos executados para qualquer instância de tamanho n.

O melhor caso, o caso médio e o pior caso

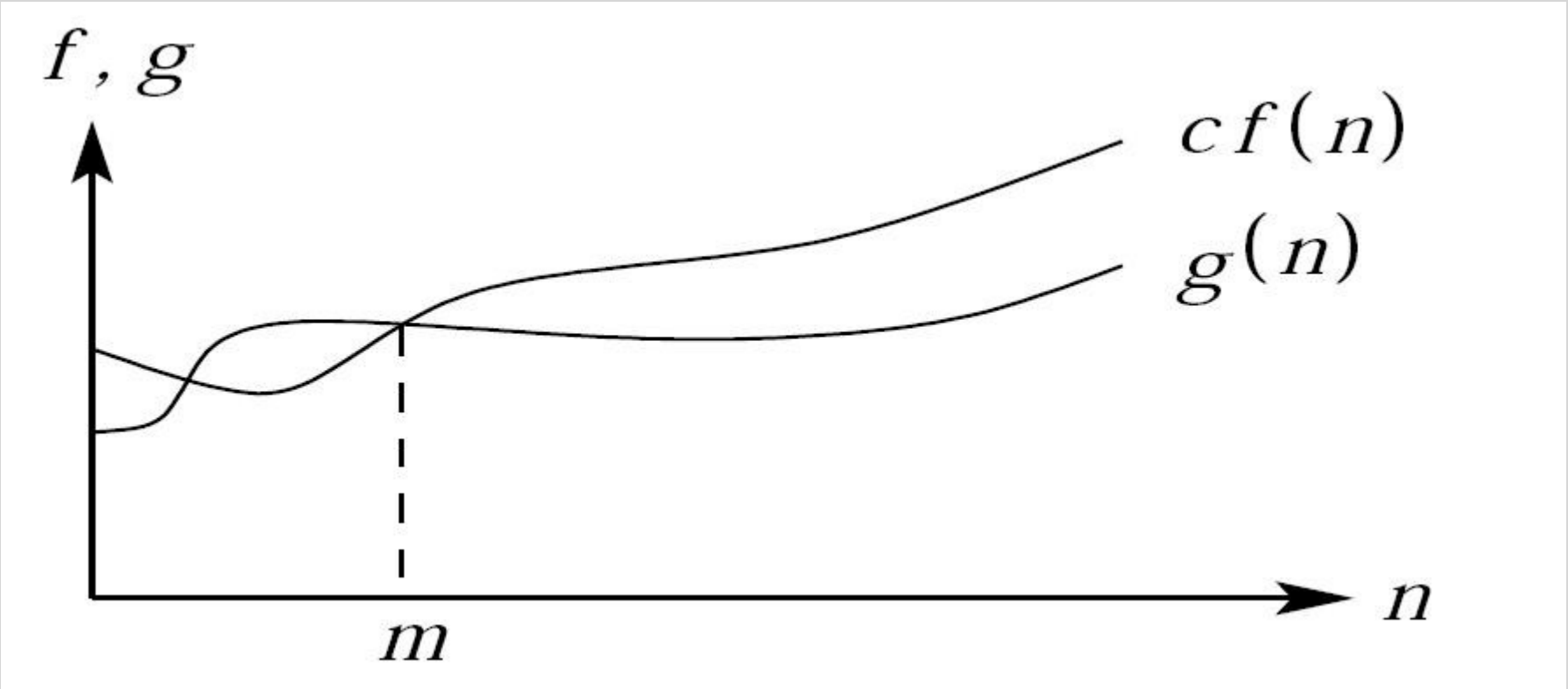


Complexidade Assintótica

- Para pequenos valores de n, a maioria dos algoritmos não representa problemas
- Estuda-se a complexidade somente para grandes valores de n, e essa complexidade é chamada assintótica
- O comportamento assintótico de g*(n)* representa o limite do comportamento do custo quando n cresce

Dominância assintótica

- Definição: Uma função f(n) domina assintoticamente outra função g(n) se existem duas constantes positivas c e m tais que, para n ≥ m, temos |g(n)| ≤ c x |f(n)|.



Notação O-Grande

- Escrevemos ***g(n) é O(f(n))*** (ou ainda ***g(n) = O(f(n))***) para expressar que f(n) domina assintoticamente g(n). Lê-se "*g(n) é da ordem no máximo f(n)*"
- Exemplo:
  - Sejam $g(n) = (n + 1)^2$ e $f(n) = n^2$.
  - As funções g(n) e f(n) dominam assintoticamente uma a outra, desde que $|(n + 1)^2| \leq 4|n^2|$ para $n \geq 1$ e $|(n+1)^2| \geq |n^2|$ para $n \geq 0$

Notação O-Grande, cont.

- Definição:
  - Uma função g(n) é O(f(n)) se existem duas constantes positivas c e m tais que $|g(n)| \leq |cf(n)|$, para todo $n \geq m$.
- Exemplos:
  - $g(n) = (n + 1)^2$.
  - g(n) é $O(n^2)$, quando m = 1 e c = 4.
  - Isto porque $(n + 1)^2 \leq 4n^2$ para $n \geq 1$.

Notação O-Grande, cont.

- $g(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$
  - Basta mostrar que $3n^3 + 2n^2 + n \leq 6n^3$, para $n \geq 0$.
  - A função $g(n) = 3n^3 + 2n^2 + n$ é também $O(n^4)$, entretanto esta afirmação é mais fraca do que dizer que $g(n)$ é $O(n^3)$.
- $g(n) = \log_5 n$ é $O(\log n)$

$$n = 10^{\log_{10} n}$$

$$\log_5 n = \log_5 (10^{\log_{10} n})$$

$$\log_5 n = \log_{10} n \times \log_5 10$$

Constante!

Operações com O-grande

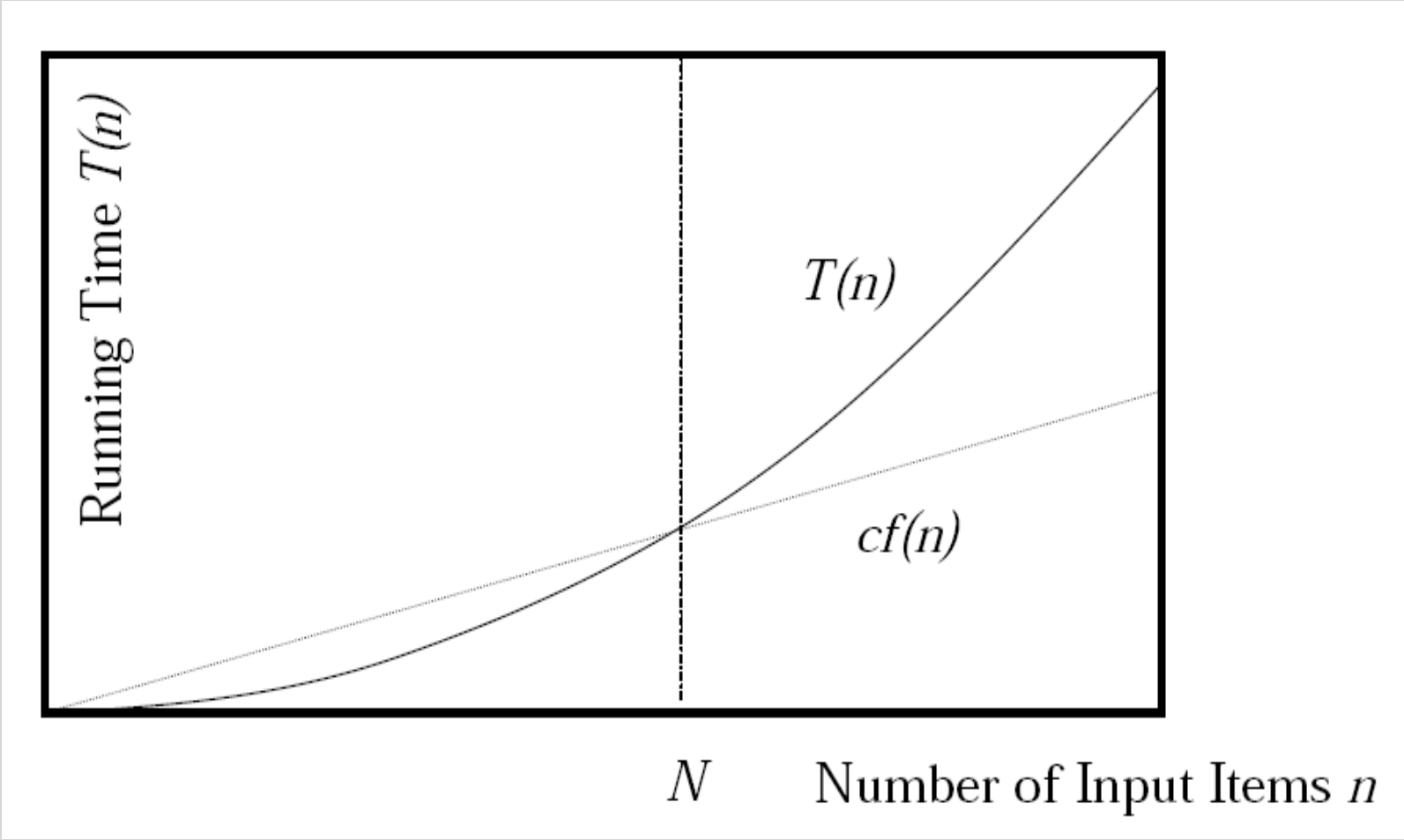
$$
\begin{array}{rcll}
f(n) & = & O(f(n)) & \\
c \times O(f(n)) & = & O(f(n)) & c = constante \\
O(f(n)) + O(f(n)) & = & O(f(n)) & \\
O(O(f(n)) & = & O(f(n)) & \\
O(f(n)) + O(g(n)) & = & O(max(f(n), g(n))) & \\
O(f(n))O(g(n)) & = & O(f(n)g(n)) & \\
f(n)O(g(n)) & = & O(f(n)g(n)) &
\end{array}
$$

- O(f(n)+g(n)) = O(max(f(n),g(n))
  - Suponha três trechos de programa com complexidades O(n), $O(n^2)$ e O(nlogn),
  - A complexidade dos três trechos é $O(n^2)$

Notação Ω

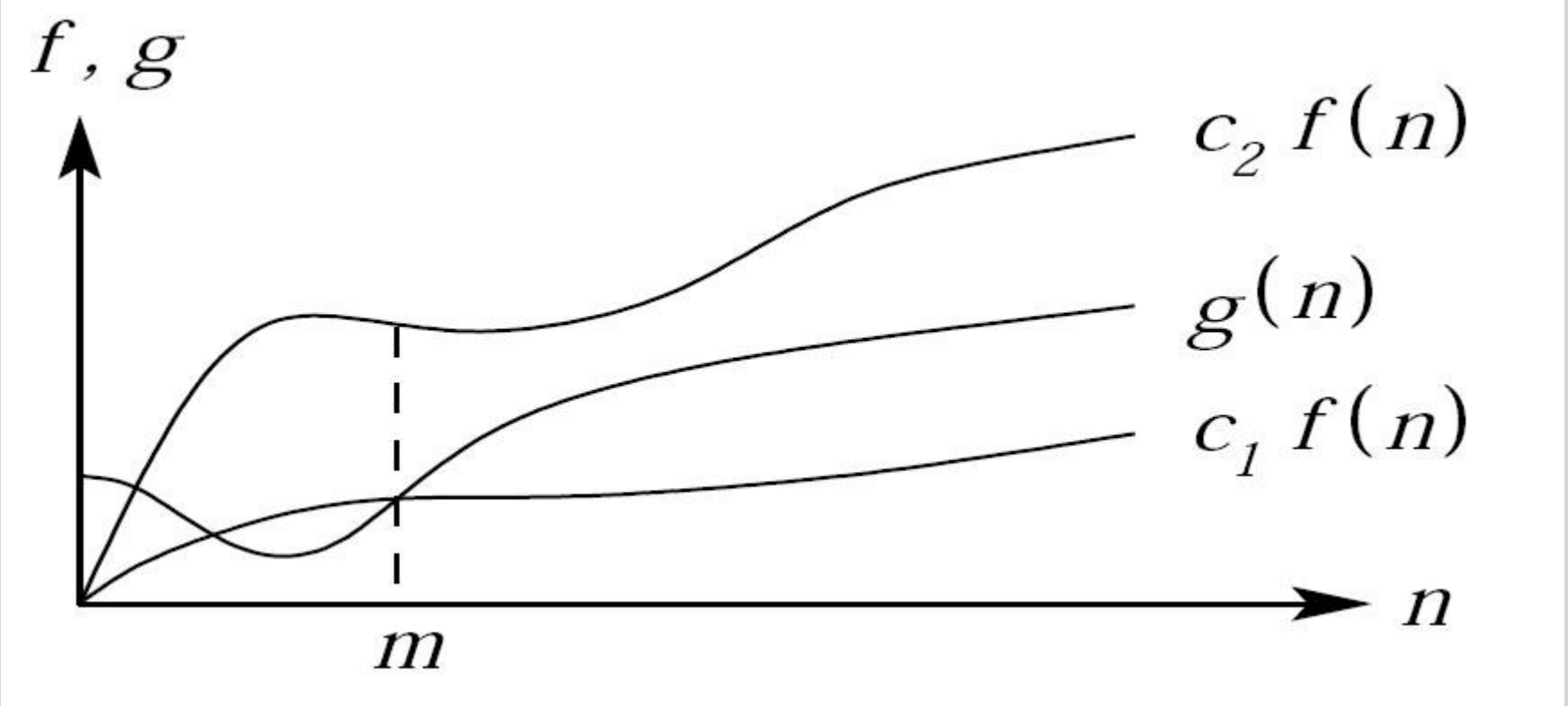
- Definição:
  - Uma função g(n) é Ω(f(n)) se existem duas constantes positivas c e m tais que $|g(n)| \geq |cf(n)|$, para todo $n \geq m$.
- Enquanto *O(f(n))* oferece um limite superior para a taxa de crescimento de g(n), Ω(f(n)) seria um limite inferior
- Exemplo:
  - $g(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$ basta fazer c = 1, e então $3n^3 + 2n^2 \geq n^3$ para $n \geq 0$

Notação Ω, Limite Inferior



Notação Θ

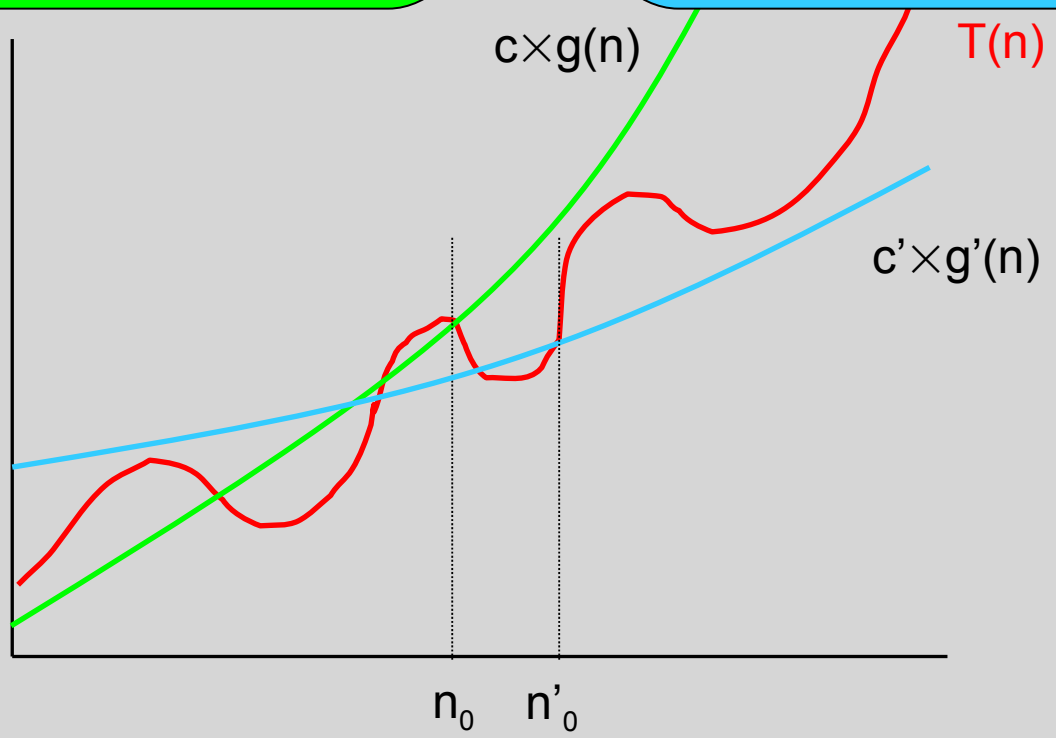
- Definição:
  - Uma função g(n) é Θ(f(n)) se existirem constantes positivas c1, c2 e m tais que 0 ≤ c1f(n) ≤ g(n) ≤ c2f(n), para todo n≥ m.
- A notação Θ é um *limite assintótico firme (ou exato)* de g(n)



Diferenciando O e Ω

$T(n) \in O(g(n))$
$\leftrightarrow$
$\exists$ constantes $c, n_0 > 0$
tais que
$\forall\ n \geq n_0,\ T(n) \leq c \times g(n)$

$T(n) \in \Omega(g'(n))$
$\leftrightarrow$
$\exists$ constantes $c', n'_0 > 0$
tais que
$\forall\ n \geq n'_0,\ T(n) \geq c' \times g'(n)$



Exercícios

- Encontre o número de passos e a complexidade dos seguintes algoritmos:

```
para i de 0 a n-1 faça
   para j de 0 a n-1 faça
     a[i][j] = b[i][j] + c[i][j]
```

```
para i de 0 a n-2 faça
   para j de i+1 a n-1 faça
      temp = a[i][j]
      a[i][j] = a[j][i]
      a[j][i]=temp
```

Solução

```
para i de 0 a n-1 faça
    para j de 0 a n-1 faça
        a[i][j] = b[i][j] + c[i][j]
```

1 atribuição feita uma única vez: i=0
1 subtração feita uma única vez: n-1
1 soma feita n vezes: i+1
1 atribuição feita n vezes: i= i+1
1 atribuição feita n vezes: j=0
1 soma feita n*n vezes: j+1
1 atribuição feita n*n vezes: j = j+1
1 soma feita n*n vezes: b[i][j]+c[i][j]
1 atribuição feita n*n vezes: a[i][j]= b[i][j]+c[i][j]

$$2+3n+4n^2 = O(n^2) \text{ e } \Omega(n^2)$$

Solução

```
para i de 0 a n-2 faça
    para j de i+1 a n-1 faça
        temp = a[i][j]
        a[i][j] = a[j][i]
        a[j][i]=temp
```

1 atribuição feita uma única vez: i=0
1 subtração feita uma única vez: n-2
1 soma feita n-1 vezes: i+1
1 atribuição feita n-1 vezes j=i+1
4 atribuições e uma soma dentro realizadas no laço j:
j = j+1, temp = a[i][j], a[i][j] = a[j][i], a[j][i] = temp
feitas (n-1)+(n-2)+(n-3)+...1 vezes, ou seja $5*\Sigma(n-k)$, k=1 até n-1, = $5*n^2/2 - 5/2$

$$2+2*(n-1)+2.5n^2-2.5$$

$$2.5n^2+2n-2.5 = O(n^2) \text{ e } \Omega(n^2)$$

Exercícios

- Mostre que:
  - $2^{(n+a)} = O(2^n)$
- Suponha que um algoritmo A e um algoritmo B, com funções de complexidade de temp $a(n) = n^2 - n + 549$, e $b(n) = 49n+49$, respectivamente. Determine quais valores de n pertencentes ao conjunto dos números naturais para os quais A leva menos tempo para executar do que B.