UNIVERSIDADE
FEDERAL DO CEARÁ

Análise de Algoritmos

TÓPICOS ESPECIAIS EM COMPUTAÇÃO II e ESTUDOS ESPECIAIS

UFC – Engenharia da Computação – 2023-2

Prof. Fischer Jônatas Ferreira

Modelo de Computação

- Analisar um algoritmo permite prever seu comportamento e desempenho sem que seja necessário implementá-lo.
- A análise de um algoritmo depende do modelo computacional adotado.
- É de acordo com o modelo computacional que se define quais são os recursos disponíveis e quanto custam.
- O Modelo RAM - Random Access Machine simula máquinas convencionais.

Modelo RAM

- Composta por um único processador que executa instruções sequencialmente;
- A memória única é infinita;
- As operações são todas executadas sequencialmente, uma única por vez;
- A execução de toda e qualquer operação toma uma unidade de tempo;
  - Operações aritméticas básicas (somas, subtrações, multiplicações e divisões), atribuições e comparações são feitas em tempo constante
- O programa não é armazenado na memória, então o espaço que ele ocupa não é contabilizado no gasto de memória.

Complexidade de tempo e espaço

- A complexidade de tempo não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.
- A complexidade de espaço representa a quantidade de memória (numa unidade qualquer) que é necessário para armazenar as estruturas de dados associadas ao algoritmo.
- Usa-se a notação assintótica para representar essas complexidades:

  - O (O grande);
  - Ω (Ômega grande);
  - Θ (Teta);
  - o (o pequeno);
  - ω (ômega pequeno).

Classes e Comportamento Assintótico

- Em geral, é interessante agrupar os algoritmos em Classes de Comportamento Assintótico, que vão determinar a complexidade inerente do algoritmo

- O comportamento assintótico é medido quando o tamanho da entrada (n) tende a infinito, com isso, as constantes são ignoradas e apenas o comportamento mais significativo da função de complexidade é considerado

- Quando dois algoritmos fazem parte da mesma classe de comportamento assintótico, eles são ditos equivalentes

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)

- Algoritmos de complexidade O(log n) é dito de complexidade logarítmica.
- Típico em algoritmos que transformam um problema em outros menores.
- A base pouco importa para a análise de complexidade;

Principais Classes de Problemas

f(n) = O(n)

- Algoritmos de complexidade O(log n) é dito de complexidade linear.
- Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
- É melhor situação para um algoritmo que tem de processar/produzir n elementos de entrada/saída;

Principais Classes de Problemas

f(n) = O(n log n)

- Típico em algoritmos que quebram um problema em outros menores. resolve, cada um deles independente e ajuntando as soluçõe depois

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 em laço de repetição dentro de outro laço.
- Quando n é mil, o número de operações é da ordem de 1 milhão.
- 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.
- Ocorrem quando os itens de dados são processados em dois laços de repetição dentro de outro laço.
- Quando n é cem, o número de operações é da ordem de 1 milhão.
- Úteis para resolver problemas de tamanhos relativamente pequenos;

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 de ponto de vista prático.
- Ocorrem na solução de problemas quando se usa força bruta para resolver.
- Quando n é 20, o número de operações é da ordem de 1 milhão;

Principais Classes de Problemas

f(n) = O(n!)

- Um algoritmo de complexidade O(n!) é dito ter complexidade fatorial, bem pior que os exponenciais
- Geralmente não são úteis de ponto de vista prático.

- Ocorrem na solução de problemas quando se usa força bruta para resolver.

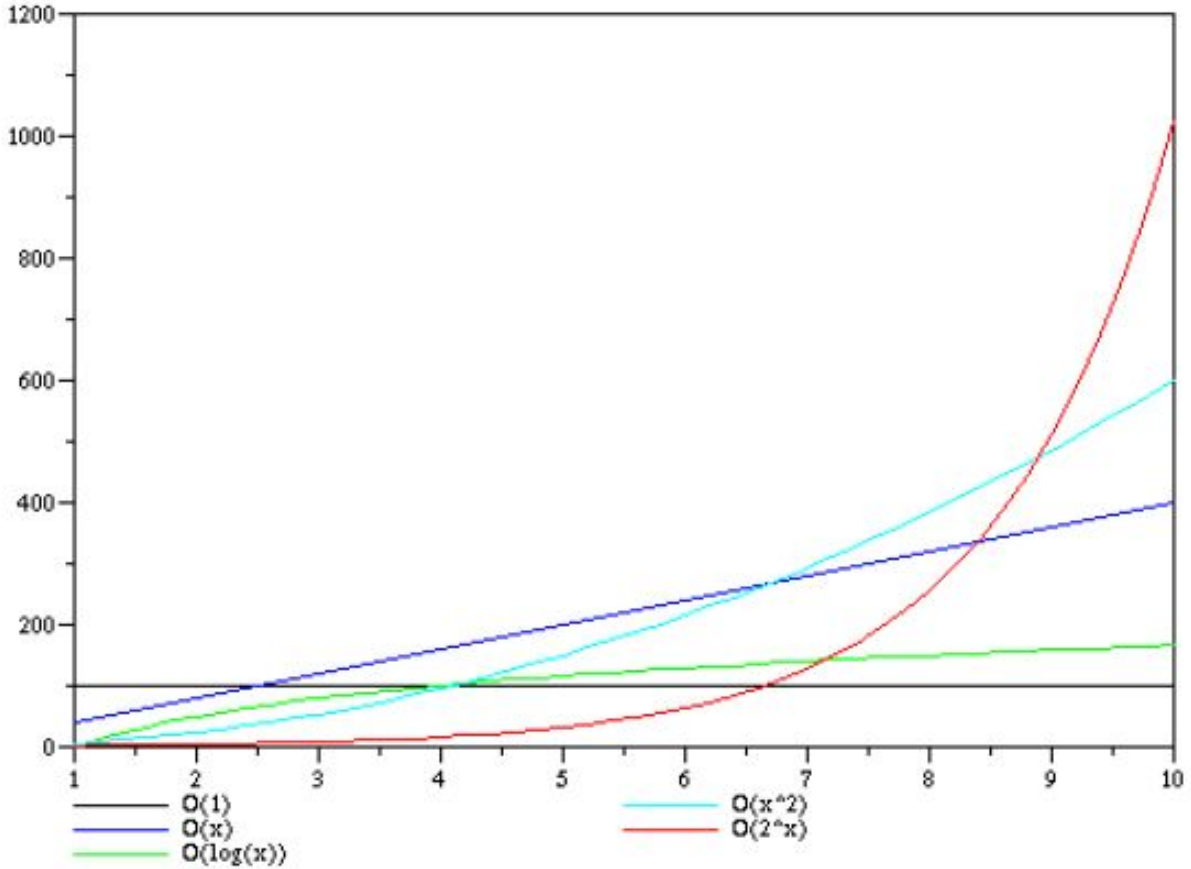
- n =20 => 20! 2432902008176640000, um número com 19 dígitos.

- n =40 => un número com 48 dígitos;

Principais Classes de Problemas

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 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

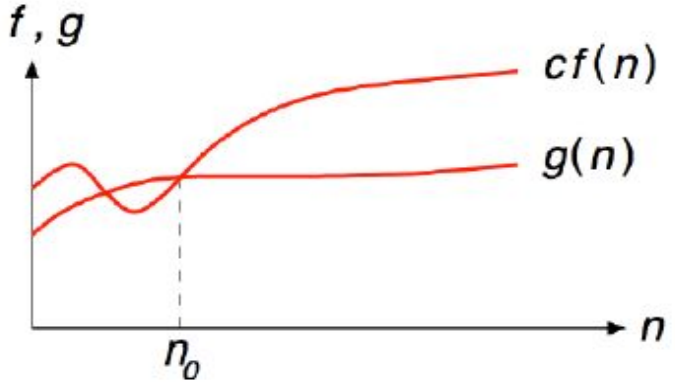
Principais Classes de Problemas



Dominação assintótica

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares
- 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_0 tais que, para $n \geq n_0$, temos

$$|g(n)| \leq c \times |f(n)|$$



- Exemplo:
  - Sejam $g(n) = (n+1)^2$ e $f(n) = n^2$
  - As funções $g(n)$ e $f(n)$ dominam assintoticamente uma a outra, já que
  - $|(n+1)^2| \leq 4|(n^2)|$ para $n \geq 1$ e
  - $|(n^2)| \leq |(n+1)^2|$ para $n \geq 0$

Custo Assintótico de Funções

- **Função de Custo** ou **Função de Complexidade**
  - T(n) = medida de custo necessário para executar um algoritmo para um problema de tamanho n
  - Se T(n) é uma medida da quantidade de tempo necessário para executar um algoritmo para um problema de tamanho n, então T é chamada função de complexidade de tempo de algoritmo
  - Se T(n) é uma medida da quantidade de memória necessária para executar um algoritmo para um problema de tamanho n, então T é chamada função de complexidade de espaço de algoritmo

Observação: TEMPO NÃO É TEMPO!

É importante ressaltar que a complexidade de tempo na realidade não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.

Custo Assintótico de Funções

- É interessante comparar algoritmos para valores grandes de n
- O *custo assintótico* de uma função T(n) representa o limite do comportamento de custo quando n cresce
- Em geral, o custo aumenta com o tamanho n do problema

Observação:

Para valores pequenos de n, mesmo um algoritmo ineficiente não custa muito para ser executado

Melhor caso, pior caso e caso médio

Melhor Caso:

- Menor tempo de execução sobre todas as entradas de tamanho n

Pior Caso:

- Maior tempo de execução sobre todas as entradas de tamanho n.
- Se f é uma função de complexidade baseada na análise de pior caso, o custo de aplicar o algoritmo nunca é maior do que f(n).

Caso Médio:

- Média dos tempos de execução de todas as entradas de tamanho n.

Melhor caso, pior caso e caso médio

- Aproveitamentos dos alunos do curso de estudos especiais.

Melhor caso, pior caso e caso médio

- Considere o problema de acessar os registros de um arquivo.
- Cada registro contém uma chave única que é utilizada para recuperar registros do arquivo.
- O problema: dada uma chave qualquer, localize o registro que contenha esta chave.
- O algoritmo de pesquisa mais simples é o que faz a pesquisa sequencial.
- Seja **f** uma função de complexidade tal que **f(n)** é o número de registros consultados no arquivo (número de vezes que a chave de consulta é comparada com a chave de cada registro).

– Melhor caso: $f(n) = 1$ (registro procurado é o primeiro consultado);

– Pior caso: $f(n) = n$ (registro procurado é o último consultado ou não está presente no arquivo);

– Caso médio: $f(n) = \frac{n+1}{2}$.

Melhor caso, pior caso e caso médio

- Considere o problema de encontrar o maior e o menor elemento de um vetor de inteiros A[1 .. n]; n >= 1.

```
procedure MaxMin1 (var A: Vetor; var Max, Min: integer);
var i: integer;
begin
  Max := A[1];  Min := A[1];
  for i := 2 to n do
    begin
      if A[i] > Max then Max := A[i]; {Testa se A[i] contém o maior elemento}
      if A[i] < Min then Min := A[i]; {Testa se A[i] contém o menor elemento}
    end;
end;
```

- Pior caso, melhor caso e caso melhor: $f(n) = n$

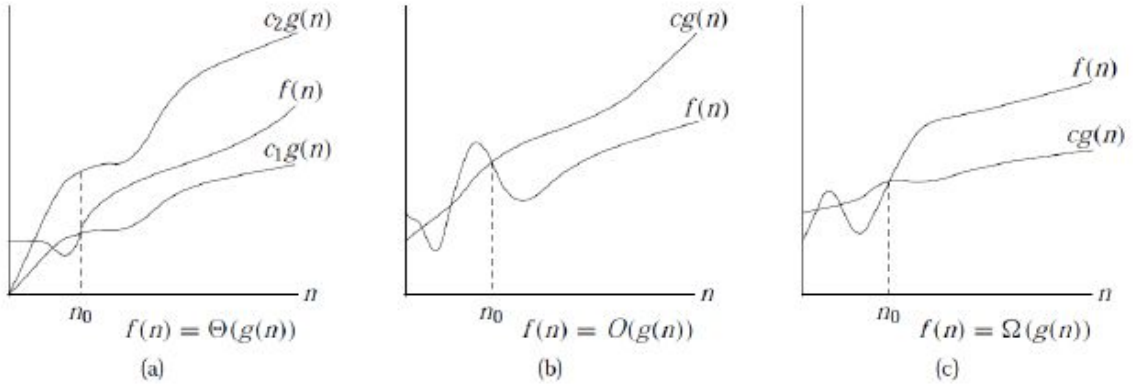
Melhor caso, pior caso e caso médio

```
INSERTION-SORT(A)
1 for j ← 2 to comprimento[A]
2       do chave ← A[j]
3          ▷ Inserir A[j] na seqüência ordenada A[1..j − 1].
4          i ← j − 1
5          while i > 0 e A[i] > chave
6             do A[i + 1] ← A[i]
7                i ← i − 1
8          A[i + 1] ← chave
```

Melhor caso: O(n^2)
Pior caso: Ω(n)

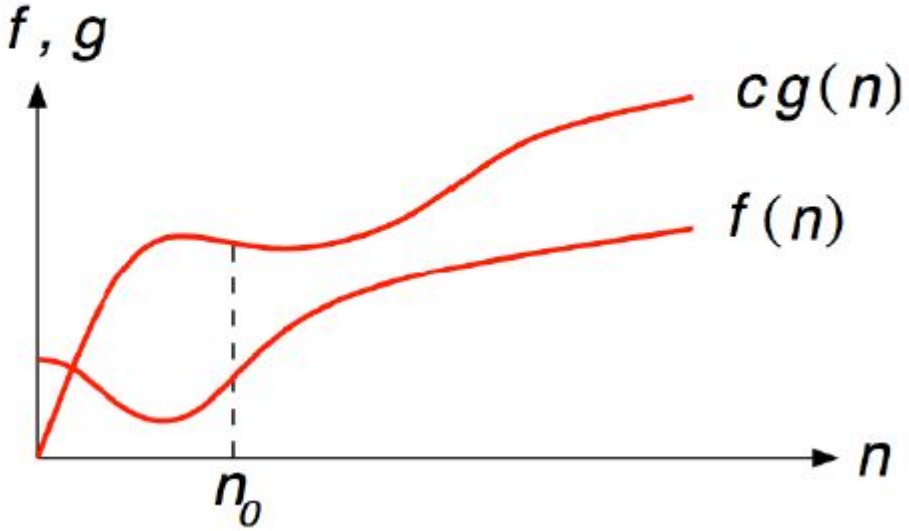
Notação assintótica de funções

- Existem três notações principais na análise de assintótica de funções:
  - Notação O (“O” grande)
  - Notação Ω (Ômega)
  - Notação Θ (Teta)



Notação O – ***Big-O***

- f(n) = O(g(n))



Notação O ***Big-O***

Seja n um inteiro positivo e sejam f(n) e g(n) funções positivas dizemos que:

f(n) = Og(n)

se existem constantes positivas C e n0 tais que $f(n) \leq c.g(n)$ para todo $n \geq n0$

Limite assintótico superior.

• Escrevemos f(n) = O(g(n)) para expressar que g(n) domina assintoticamente f(n). Lê-se f(n) é da ordem no máximo g(n).

Notação O ***Big-O***

- Seja *f(n) = (n + 1)2*
  - Logo *f(n)* é $O(n^2)$, quando n0 = 1 e c = 4, já que

$$(n+1)^2 \leq 4n^2 \text{ para } n \geq 1$$

- Seja *f(n) = n* e *g(n)* = n^2. Mostre que *g(n)* não é *O(n)*.
  - Sabemos que *f(n)* é $O(n^2)$, pois para $n \geq 0$, $n \leq n^2$.
  - Suponha que existam constantes c e n0 tais que para todo $n \geq n_0$, $n^2 \leq cn$.
  - Assim, $c \geq n$ para qualquer $n \geq n_0$.
  - No entanto, não existe uma constante *c* que possa ser maior ou igual a *n* para todo *n*.

Notação O ***Big-O***

- Quando a notação O é usada para expressar o tempo de execução de um algoritmo no pior caso, está se definindo também o limite superior do tempo de execução desse algoritmo para *todas* as entradas

- Por exemplo, o algoritmo de ordenação por inserção é $O(n^2)$ no pior caso
  - Este limite se *aplica* para *qualquer* entrada

Notação O ***Big-O***

- Tecnicamente é um abuso dizer que o tempo de execução do algoritmo de ordenação por inserção é $O(n^2)$ (i.e., sem especificar se é para o pior caso, melhor caso, ou caso médio)
  - O tempo de execução desse algoritmo depende de como os dados de entrada estão arranjados.
  - Se os dados de entrada já estiverem ordenados, este algoritmo tem um tempo de execução de $O(n)$, ou seja, o tempo de execução do algoritmo de ordenação por inserção no *melhor caso* é $O(n)$.

- O que se quer dizer quando se fala que "o tempo de execução é $O(n^2)$" é que no pior caso o tempo de execução é $O(n^2)$
  - ou seja, não importa como os dados de entrada estão arranjados, o tempo de execução em qualquer entrada é $O(n^2)$

Notação Ômega (notação Ω)

Notação que limita funções inferiormente - *Notação Ômega (notação Ω)*

f(n)= Ωg(n)

se existem constantes positivas C e n0 tais que c.g(n) <= f(n) para todo n ≥ n0

Notação Ômega (notação Ω)

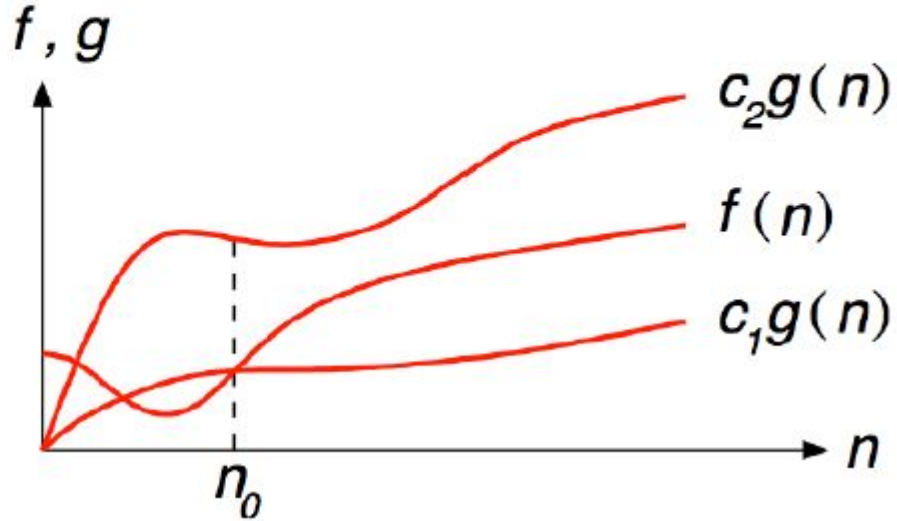
- Quando a notação Ω é usada para expressar o tempo de execução de um algoritmo no melhor caso, está se definindo também o limite (inferior) do tempo de execução desse algoritmo para todas as entradas

- Por exemplo, o algoritmo de ordenação por inserção é Ω(n) no melhor caso
  - O tempo de execução do algoritmo de ordenação por inserção é Ω(n)

- O que significa dizer que "o tempo de execução" (i.e., sem especificar se é para o pior caso, melhor caso, ou caso médio) é Ω(g(n))?
  - O tempo de execução desse algoritmo é pelo menos uma constante vezes g(n) para valores suficientemente grandes de n

Notação Θ (Teta)

- f(n) = Θ(g(n))



Notação Θ (Teta)

Notação que limita funções inferiormente e superiormente comitantemente - N*otação Theta (notação Θ)*

f(n) = Θg(n)

se existem constantes positivas c1, c2 e n0 tais que c1.g(n) ≤ f(n) ≤ c2.g(n) para todo n ≥ n0

Notação Θ (Teta) - Exemplo

- Mostre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- Para provar esta afirmação, devemos achar constantes $c_1 > 0$, $c_2 > 0$, $n_0 > 0$, tais que:

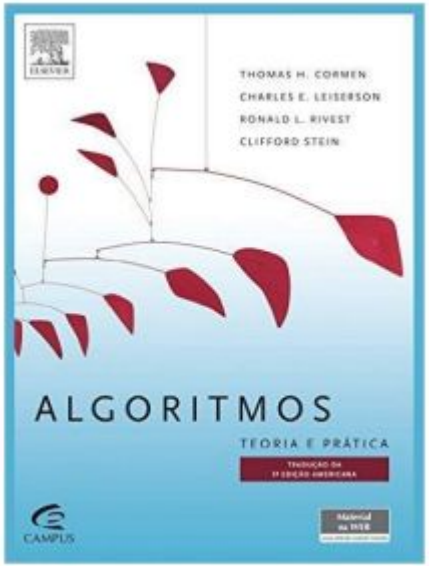
$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

- para todo $n \geq n_0$
- Se dividirmos a expressão acima por n^2 temos:

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

Bibliografia principal

Cormen, Thomas H., et al. "Algoritmos: teoria e prática." Editora Campus 2 (2002).



Bibliografia alternativa

- Dasgupta, Sanjoy, Christos Papadimitriou, and Umesh Vazirani. Algoritmos. AMGH Editora, 2009.

- Ziviani, Nivio. Projeto de algoritmos: com implementações em Pascal e C. Vol. 2. Thomson, 2004.

- Lintzmayer, Carla. Análise de Algoritmos e de Estruturas de Dados http://professor.ufabc.edu.br/~carla.negri/cursos/materiais/Livro-Analise.de.Algoritmos.pdf