# Comportamento Assintótico

Algoritmos e Estruturas de Dados 2
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

# Até Agora

- Falamos de complexidade de algoritmos com base no número de passos

- Vamos generalizar mais um pouco com classes de complexidade

- Na prática:
    - Vamos ter um embasamento mais matemático
    - Vamos ignorar as constantes

# Comportamento Assintótico

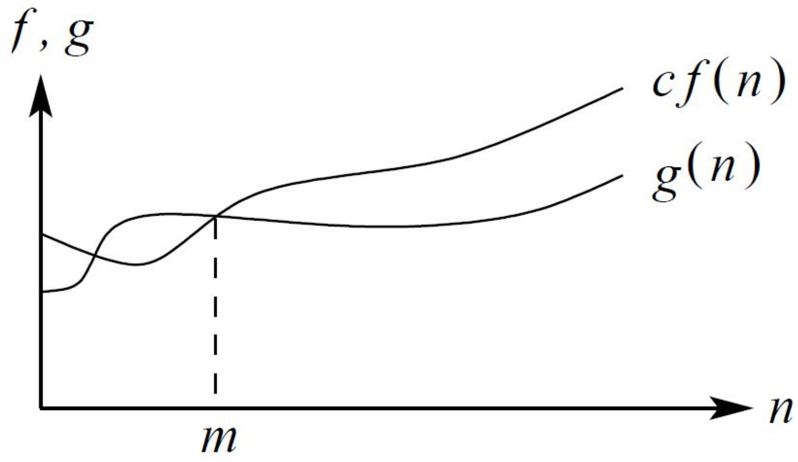
- Para valores suficientemente pequenos de *n*, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes
  - [Geralmente] Escolha de um algoritmo não é um problema crítico com n pequeno

- Logo, analisamos algoritmos para grandes valores de *n*
  - Estudamos o comportamento assintótico das funções de complexidade de um programa (comportamento para grandes valores de *n*)

- Ao escolher um n *inicial* suficientemente grande, podemos comparar 2 funções utilizando o crescimento das mesmas

# Dominação Assintótica

- Uma função $f(n)$ domina assintoticamente outra função $g(n)$ se existem duas constantes *positivas* $c$ e $m$ tais que, para $n \geq m$, temos $|g(n)| \leq c|f(n)|$.



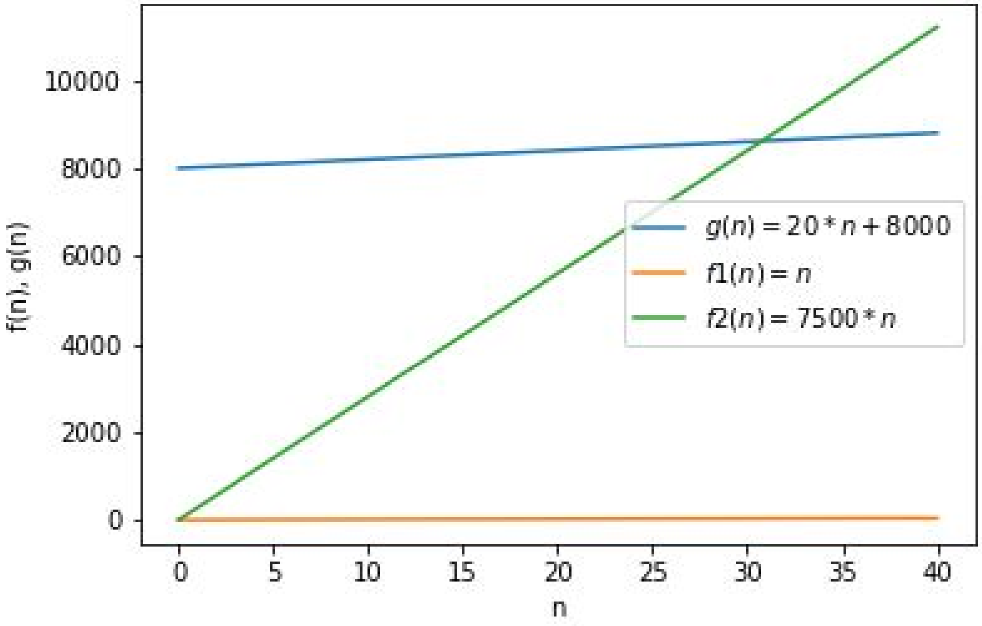
# Vamos Comparar 2 Funções

- $g(n) = 20 * n + 8000$
- $f(n) = c * n$

- Olhe a função ao lado

- Existe um ponto onde
  $g(n) \leq c * f(n)$
  sim para algum c (280)

- Qual ponto é este?



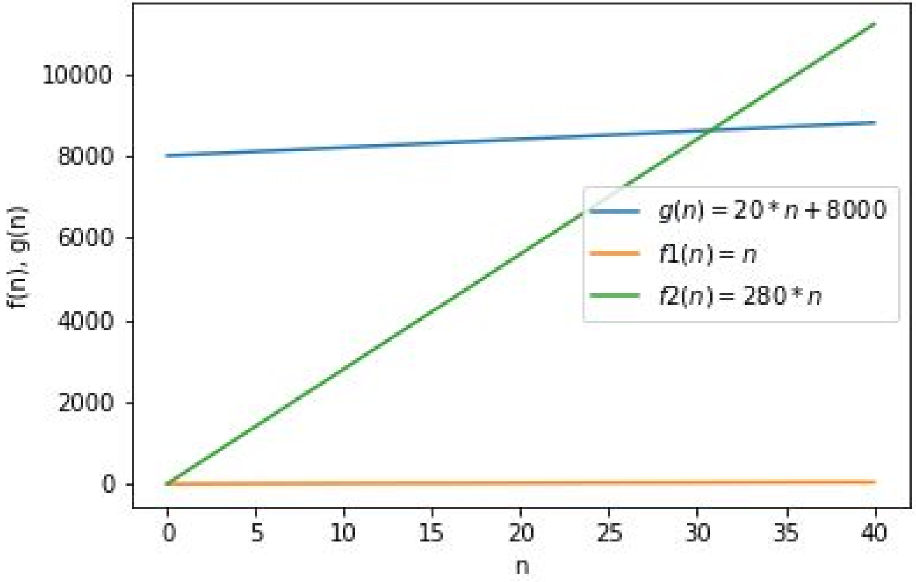
# Vamos Comparar 2 Funções

- $g(n) = 20 * n + 8000$
- $f(n) = n$

- Olhe a função ao lado

- Existe um ponto onde
  $g(n) \leq 7500 * f(n)$

- Qual ponto é este?
  - f(n) = g(n)



# Exemplo 2

- $g(n) = 6 * n^4 + 2 * n^3 + 5$
- $f(n) = n^4$
- Temos que achar c e m, para qualquer n >= m
  - $|g(n)| \leq c|f(n)|$
- $m=1, c=13$
- $6 * n^4 + 2 * n^2 + 5 \leq 13 * n^4$

  $6 * n^4 + 2 * n^2 + 5 \leq 6 * n^4 + 2 * n^4 + 5 n^4$

  é fácil ver que
- $6 * n^4 \leq 6 * n^4$

  $2 * n^2 \leq 2 * n^4$

  $5 \leq 5 * n^4$

  cada fator individual do lado esquerdo tem valor menor do que os do lado direito. A soma sempre será menor

# Exemplo 3

- $f(n) = n^2$, $g(n) = n$

  f($n$) domina assintoticamente g($n$)
  $c = 1$, $m = 1$

  $|g(n)| \leq 1|f(n)|$ para todo $n \geq m = 0$

- Qual a implicação?

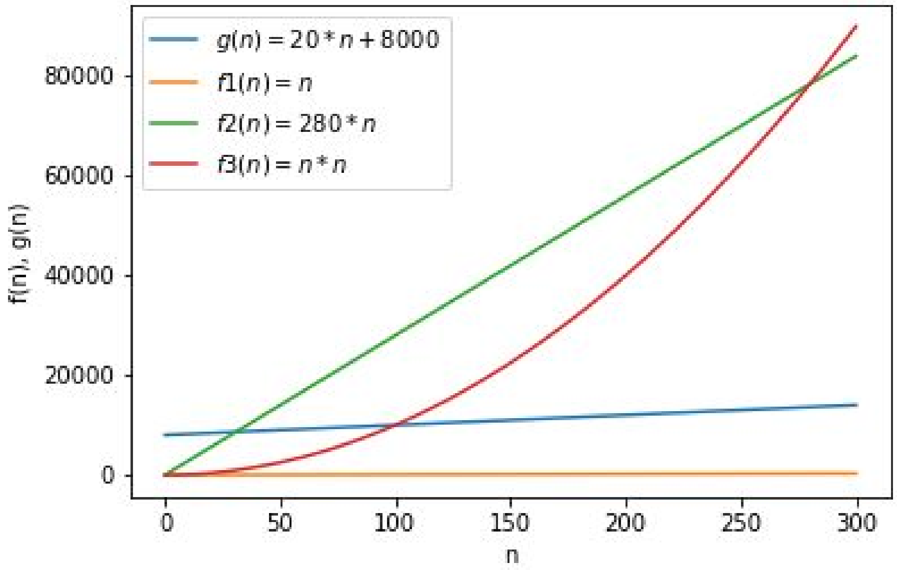
# Exemplo 3

- $f(n) = n^2, g(n) = n$
  f($n$) domina assintoticamente g($n$)
  $c = 1, m = 1$
  $|g(n)| \leq 1|f(n)|$ para todo $n \geq m = 0$
- Qual a implicação?
  - Todo algoritmo de n passos tem um limite superior em $n^2$ também
  - Isso é importante?
  - Sim, pois o inverso não é verdade
    - Não conseguimos mostrar que $n^2$ é $O(n)$
      - Nunca existe c e m para este caso
  - n é $O(n)$ **e** $O(n^2)$; $n^2$ é $O(n^2)$ mas **não é** $O(n)$
  - nos preocupamos com um limite superior justo

# Limites superiores justos

- Tanto n*n quanto 280 * n limitam g(n)

- Porém:
  - 280 * n é um limite mais justo
  - Note que n * n passa de 280 *n em algum momento
  - Quando falamos que um algoritmo é O(n), queremos que n seja um limite justo!



# Outro Exemplo

- $f(n) = n^2$, $g(n) = (n+1)^2$

  $f(n)$ e $g(n)$ dominam assintoticamente uma à outra

  $|f(n)| \leq 1\,|g(n)|$ para todo $n \geq m = 0$

  $|g(n)| \leq 4\,|f(n)|$ para todo $n \geq m = 1$

- Aqui podemos falar que as funções são da mesma classe

# Voltando Para os MinMax

- Falamos de 3 algoritmos MinMax nas últimas aulas
- A tabela abaixo mostra o número de passos dos 3
- Exercício: Algum algoritmo MinMax é assintoticamente melhor do que outro?
  - Olhar apenas o pior caso
  - Comparar MinMax3 com MinMax1

| Os três algoritmos | $f(n)$ | | |
|---|---|---|---|
| | Melhor caso | Pior caso | Caso médio |
| MaxMin1 | $2(n-1)$ | $2(n-1)$ | $2(n-1)$ |
| MaxMin2 | $n-1$ | $2(n-1)$ | $3n/2-3/2$ |
| MaxMin3 | $3n/2-2$ | $3n/2-2$ | $3n/2-2$ |

# Comparando os MinMax

- Os três algoritmos são equivalentes assintoticamente

- Qual a implicação disto?
  - Existe um ganho constante em usar MinMax3
  - Porém a complexidade do mesmo cresce igual a MinMax2 e MinMax

- Às vezes vale a pena pagar o preço da constante
  - O MinMax3 é mais ou menos 50% (constante) mais rápido do que os outros
  - Para vetores muito muito grande pode ver a pena

- Às vezes não
  - Algoritmo mais complicado
  - Pouco ganho real

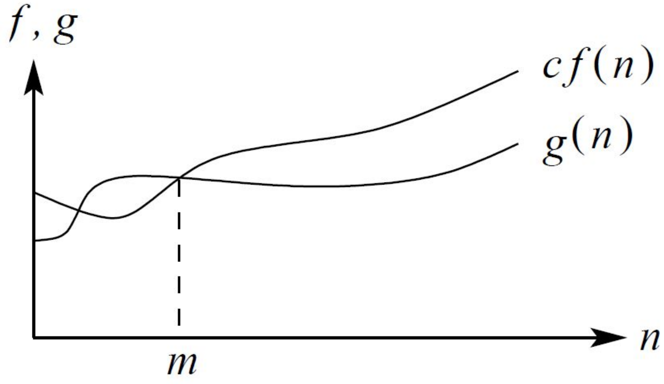
# Notação O

- Definimos $g(n) = O(f(n))$ se $f(n)$ domina assintoticamente $g(n)$

- Lê se $g(n)$ é da ordem no máximo $f(n)$

- Quando dizemos que o tempo de execução de um programa $T(n) = O(n^2)$, existem constantes c e $m$ tais que $T(n) \leq cn^2$ para $n \geq m$

- Geralmente:
  - O comportamento até antes de m não importa
  - Porém:
    - Existem casos onde chaveamos os algoritmos dependendo de n. Alguns algoritmos de ordena de uso específico
    - Fora do escopo da disciplina

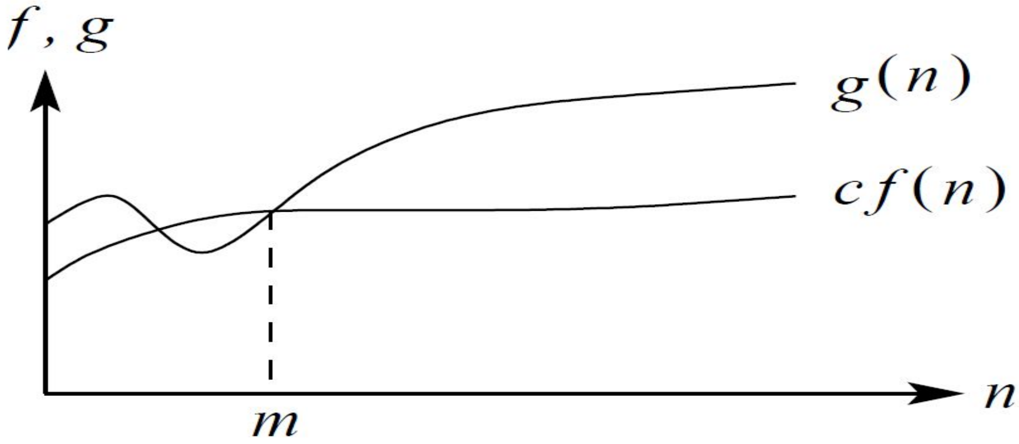


# Propriedades

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

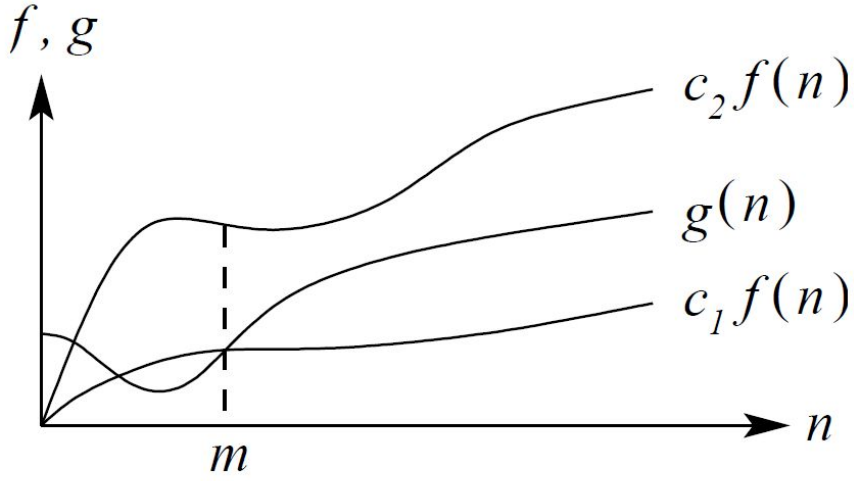
# Notação $\Omega$

- Invertemos o caso anterior
- Estamos olhando um limit inferior agora
- Uma função $f(n)$ é dominada assintoticamente outra função $g(n)$ se existem duas constantes ***positivas*** $c$ e $m$ tais que, para $n \geq m$ ,
temos $c\,|\,f(n)\,| \leq |\,g(n)\,|$.



# Notação Θ

- Limite firme (superior e inferior ao mesmo tempo)
- Estamos olhando um limite inferior agora
- $c1|f(n)| \leq |g(n)| \leq c2|f(n)|$.



# Classes de Funções Comuns com Exemplos

- $O(1)$
  - Constante
- $O(\log n)$
  - Algoritmos de busca
- $O(n \log n)$
  - Ordenação
- $O(n^2)$
  - Matrizes
- $O(n^3)$
  - Matrizes
- $O(2^n)$
  - Problemas exponenciais. Temos que enumerar todas as respostas.

| | *constant* | *logarithmic* | *linear* | *N-log-N* | *quadratic* | *cubic* | *exponential* |
|---|---|---|---|---|---|---|---|
| $n$ | $O(1)$ | $O(\log n)$ | $O(n)$ | $O(n \log n)$ | $O(n^2)$ | $O(n^3)$ | $O(2^n)$ |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |
| 2 | 1 | 1 | 2 | 2 | 4 | 8 | 4 |
| 4 | 1 | 2 | 4 | 8 | 16 | 64 | 16 |
| 8 | 1 | 3 | 8 | 24 | 64 | 512 | 256 |
| 16 | 1 | 4 | 16 | 64 | 256 | 4,096 | 65536 |
| 32 | 1 | 5 | 32 | 160 | 1,024 | 32,768 | 4,294,967,296 |
| 64 | 1 | 6 | 64 | 384 | 4,069 | 262,144 | $1.84 \times 10^{19}$ |

# Propriedades

- Imagine um programa com três fases
  - A primeira com custo $O(n)$
  - A segunda com custo $O(n^2)$
  - A terceira com custo $O(n \log(n))$

- Aplicando a regra da soma
  - O tempo de execução total do programa é $O(n^2)$

# Ordenação de Dados

- Encontre o Menor Elemento do Vetor
- Troque com o Primeiro Elemento
- Mova para o Segundo Elemento
- Repita até Percorrer o Vetor Todo

# Ordenação de Dados

Qual a complexidade
do algoritmo ao lado?

[https://goo.gl/qTyJep](https://goo.gl/qTyJep)

```c
void ordena(int *dados, int n) {
  int i;
  int j;
  int min_index;
  int aux;

  for(i = 0; i < n - 1; i++) {
    min_index = i;
    for(j = i + 1; j < n; j++)
      if(dados[j] < dados[min_index])
        min_index = j;

    /* troca A[min_index] e A[i]: */
    aux = dados[min_index];
    dados[min_index] = dados[i];
    dados[i] = aux;
  }
}
```

# TP1: Banco AEDS (Use o Esqueleto da Aula Passada)

- Seu Banco AEDS deve:
  - Ordenar as transações por tempo
  - Imprimir as transações ordenadas por tempo
  - Ordenar as transações por valor
  - Imprimir as transações ordenadas por valor
  - Suportar qualquer número de transações
  - Documente a complexidade de todas as funções

- Use o módulo time.h para mensurar o tempo de suas funções com diferentes tamanhos (n=1, n=2, n=3...., n=10000). Apenas das funções com laços

- Gere dados aleatórios para testar se necessário
  - Ver exemplo do sort acima

# Exercícios

- Prove que $4\log_2(n) + 16 = O(n)$

- Prove que $4\log_2(n) + 16 = O(\log_2 n)$

- $2^{n+1} = O(2^n)$. Verdadeiro ou falso?

- $2^{2n} = O(2^n)$. Verdadeiro ou falso?

# Exercícios

- Prove que $4\log_2(n) + 16 = O(n)$
  - $4\log_2(n) + 16 \le n$ para $n \ge m = 64 = 2^6$

- Prove que $4\log_2(n) + 16 = O(\log_2 n)$
  - $4\log_2(n) + 16 \le 5\log_2(n)$ para $n \ge m = 2^{17}$

- $2^{n+1} = O(2^n)$. Verdadeiro ou falso?
  - Verdadeiro, faça $\mathrm{c} = 2$ e $\mathrm{m} = 0$

- $2^{2n} = O(2^n)$. Verdadeiro ou falso?
  - Falso.