# Medida do Tempo de Execução de um Programa

*Livro "Projeto de Algoritmos" – Nívio Ziviani*
*Capítulo 1 – Subseção 1.3.1*
*http://www2.dcc.ufmg.br/livros/algoritmos/*
*(slides adaptados)*

# Objetivos

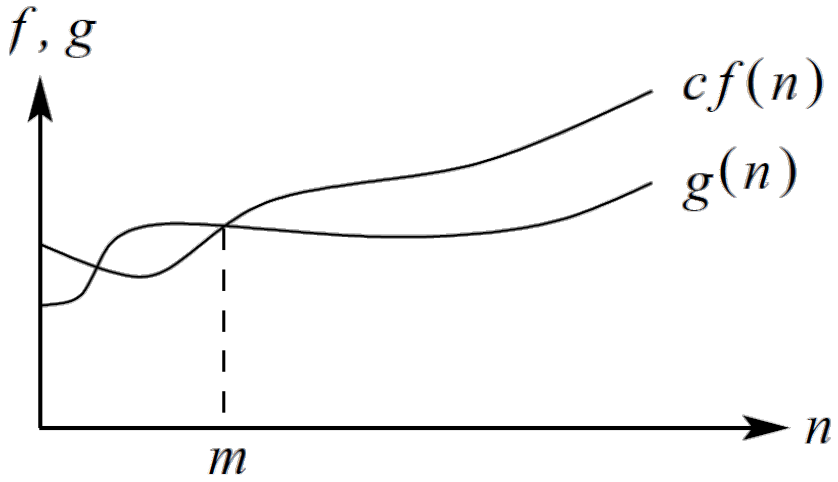
- Comportamento Assintótico
- Notações O, Ω, Θ
- Principais Classes de Problemas
- Algoritmos Polinomiais vs. Exponenciais

# Comportamento Assintótico de Funções

- O parâmetro n fornece uma medida da dificuldade para se resolver o problema.
- Algoritmos são parecidos para n pequenos – escolha não é crítica.
- Análise de algoritmos é realizada para **valores grandes de n**.
- Comportamento assintótico de f(n): limite do comportamento do custo quando n cresce.
- **Interesse**: como o custo aumenta quando n vai para o limite.

# Dominação 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|f(n)|.



# Dominação Assintótica

**Exemplo 1**:

f(n) domina assintóticamente g(n) se
|g(n)| ≤ c|f(n)| para n ≥ m e c > 0

- Sejam $f(n) = n^2$ e $g(n) = n$

- f(n) domina assintoticamente g(n) desde que

  |g(n)| ≤ c|f(n)| para n ≥ m e c > 0

  c = 1 e m = 1

| n | \|g(n)\| | \|f(n)\| |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 4 |
| 3 | 3 | 9 |
| ... | ... | ... |

# Dominação Assintótica

**Exemplo 2**:

f(n) domina assintóticamente g(n) se $|g(n)| \leq c|f(n)|$ para $n \geq m$ e $c > 0$

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

- f(n) domina assintoticamente g(n)?

$c = 4$ e $n \geq 1$

$g(n) \leq cf(n)$

$(n+1)^2 \leq 4n^2$

$n^2 + 2n + 1 \leq 4n^2$

$n + 2 + 1/n \leq 4n$

# Dominação Assintótica

**Exemplo 2**:

f(n) domina assintóticamente g(n) se
$|g(n)| \leq c|f(n)|$ para $n \geq m$ e $c > 0$

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

- g(n) domina assintoticamente f(n): c = 1 e n ≥ 1

$n^2 \leq (n+1)^2$

$n^2 \leq n^2 + 2n + 1$

$1 \leq 1 + 2/n + 1/n^2$

f(n) e g(n) dominam assintoticamente uma a outra

# Dominação Assintótica

**Exemplo 3**:

f(n) domina assintóticamente g(n) se
$|g(n)| \leq c|f(n)|$ para $n \geq m$ e $c > 0$

- Sejam $f(n) = n$ e $g(n) = n^2$

- f(n) domina assintoticamente g(n) desde que

  $|g(n)| \leq c|f(n)|$ para $n \geq m$ e $c > 0$

  $n^2 \leq cn$

  $n \leq c$ (x) (ou seja, f(n) não domina assintoticamente g(n))

# Notação Assintótica

- O - especifica um limite superior para g(n).

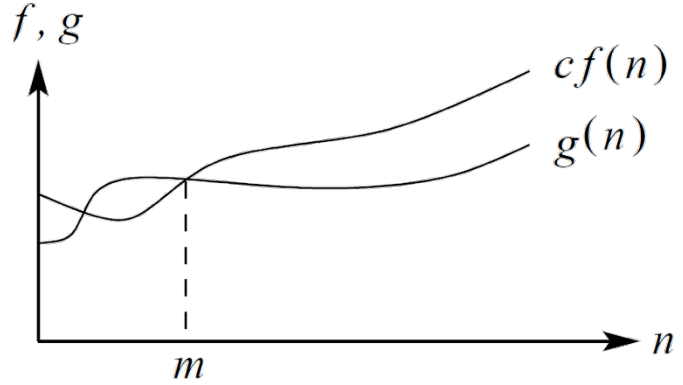
  **g(n) = O(f(n))**

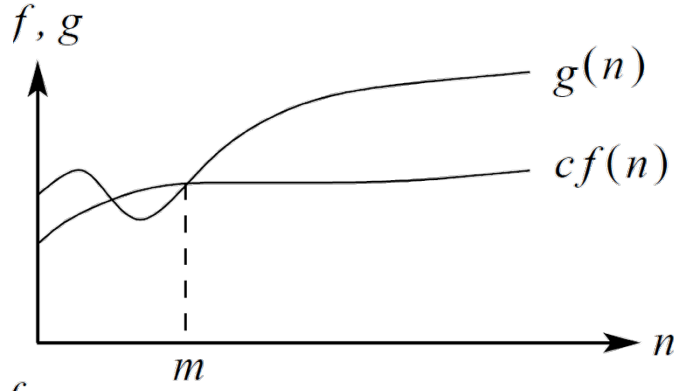
- Ω - especifica um limite inferior para g(n).

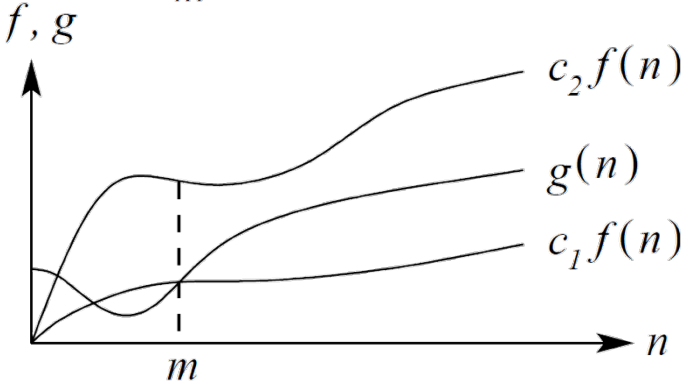
  **g(n) = Ω (f(n))**

- Θ - especifica um limite firme para g(n).

  **g(n) = Θ(f(n))**



# Notação O

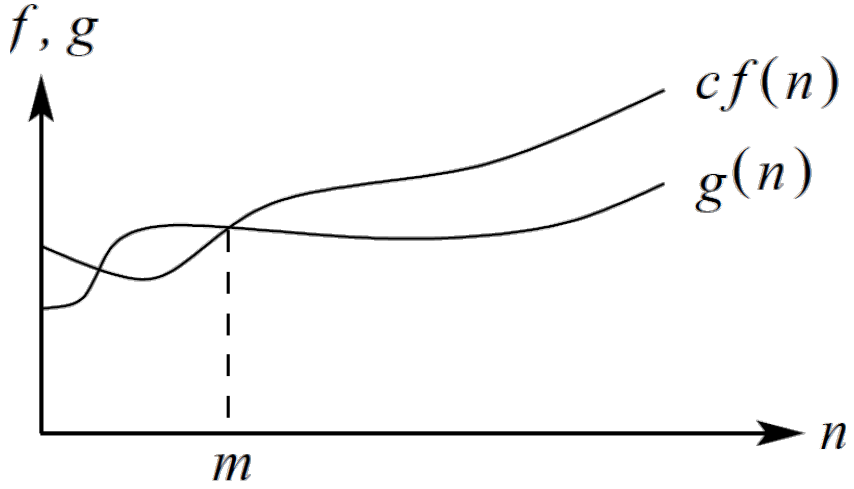
- Escrevemos **g(**n**) = O(f(**n**))** se
  - **f(**n**)** domina assintoticamente **g(**n**)**. Lê-se **g(**n**)** é da ordem no máximo **f(**n**)**.

- Exemplo: quando dizemos que o tempo de execução **f(**n**)** de um programa é **$O(n^2)$**, significa que existem constantes **c** e **m** tais que, para valores de **$n \geq m$**, **$f(n) \leq cn^2$**.

# Notação O

- **Definição**: Uma função $g(n)$ é $O(f(n))$ se existem duas constantes positivas c e m tais que

$$g(n) \leq cf(n), \text{ para todo } n \geq m.$$



# Exemplos de Notação O

$g(n)$ é $O(f(n))$ se $g(n) \leq cf(n)$, para todo $n \geq m$, *onde m e c são positivas*

- **Exemplo 4**: $g(n) = (n + 1)^2$.
  - Logo $g(n)$ é $O(n^2)$, quando $m = 1$ e $c = 4$.
  - Isto porque $(n + 1)^2 \leq 4n^2$ para $n \geq 1$.

# Exemplos de Notação O

*g*(*n*) é O(*f*(*n*)) se *g*(*n*) ≤ *cf*(*n*), para todo *n* ≥ *m*, *onde m e c são positivas*

- **Exemplo 4**: $g(n) = (n + 1)^2$.
  - Logo $g(n)$ é $O(n^2)$, quando $m = 1$ e $c = 4$.
  - Isto porque $(n + 1)^2 \leq 4n^2$ para $n \geq 1$.

$(n+1)^2 \leq cn^2$
$n^2 + 2n + 1 \leq cn^2$
$1 + 2/n + 1/n^2 \leq c$

- **Exemplo 5**: $g(n) = n$ e $f(n) = n^2$.
  - Sabemos que $g(n)$ é $O(n^2)$, pois para $n \geq 0$, $n \leq n^2$.

# Exemplos de Notação O

*g*(*n*) é *O*(*f*(*n*)) se *g*(*n*) ≤ *cf*(*n*), para todo *n* ≥ *m*, *onde m e c são positivas*

- **Exemplo 4**: $g(n) = (n + 1)^2$.
  - Logo $g(n)$ é $O(n^2)$, quando $m = 1$ e $c = 4$.
  - Isto porque $(n + 1)^2 \leq 4n^2$ para $n \geq 1$.

$(n+1)^2 \leq cn^2$
$n^2 + 2n + 1 \leq cn^2$
$1 + 2/n + 1/n^2 \leq c$

- **Exemplo 5**: $g(n) = n$ e $f(n) = n^2$.
  - Sabemos que $g(n)$ é $O(n^2)$, pois para $n \geq 0$, $n \leq n^2$.
  - Entretanto $f(n)$ não é $O(n)$.

# Exemplos de Notação O

*g*(*n*) é *O*(*f*(*n*)) se *g*(*n*) ≤ *cf*(*n*), para todo *n* ≥ *m*, *onde m e c são positivas*

■ **Exemplo 4**: $g(n) = (n + 1)^2$.

– Logo $g(n)$ é $O(n^2)$, quando $m = 1$ e $c = 4$.

– Isto porque $(n + 1)^2 \leq 4n^2$ para $n \geq 1$.

$(n+1)^2 \leq cn^2$
$n^2 + 2n + 1 \leq cn^2$
$1 + 2/n + 1/n^2 \leq c$

■ **Exemplo 5**: $g(n) = n$ e $f(n) = n^2$.

– Sabemos que $g(n)$ é $O(n^2)$, pois para $n \geq 0$, $n \leq n^2$.

– Entretanto $f(n)$ não é $O(n)$.

– Suponha que existam constantes $c$ e $m$ tais que para todo $n \geq m$, $n^2 \leq cn$.

– Logo, $c \geq n$ para qualquer $n \geq m$, e não existe uma constante $c$ que possa ser maior ou igual a $n$ para todo $n$.

# Exemplos de Notação O

■**Exemplo 6**: $g(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$.

*g(n) é O(f(n)) se g(n) ≤ cf(n), para todo n ≥ m, onde m e c são positivas*

– Basta mostrar que $3n^3 + 2n^2 + n \leq 6n^3$, para $n \geq 1$.

– 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 $O(n^3)$.

$3n^3 + 2n^2 + n \leq cn^4$

$3 + 2/n + 1/n^2 \leq cn$, $c=6$, $m=1$

■ **Exemplo 7**: $2^{n+1}$ *é* $O(2^n)$*?*

# Operações com a Notação O

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

# Operações com a Notação O

**Exemplo 8**: regra da soma $O(f(n)) + O(g(n))$.

■ Suponha três trechos cujos tempos de execução são $O(n)$, $O(n^2)$ e $O(n \log n)$.

■ O tempo de execução dos dois primeiros trechos é $O(\max(n, n^2))$, que é $O(n^2)$.

■ O tempo de execução de todos os três trechos é então $O(\max(n^2, n \log n))$, que é $O(n^2)$.

# Operações com a Notação O

**Exemplo 9**: $[n + O(1)][n + O(\log n) + O(1)]$

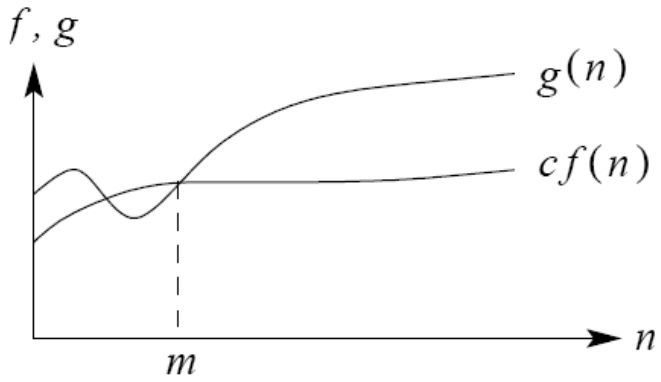
$n^2 + O(n \log n) + O(n) + O(n) + O(\log n) + O(1)$

$O(n^2)$

# Notação Ω

- Especifica um limite inferior para g(n).
- **Definição**: Uma função g(n) é Ω(f(n)) se existirem duas constantes c e m tais que

$$g(n) \geq cf(n), \text{ para todo } n \geq m.$$



# Notação Ω

*g*(*n*) é Ω(*f*(*n*)) se *g*(*n*) ≥ *cf*(*n*), para todo *n* ≥ *m, onde m e c são positivas*

- **Exemplo 10**: Mostrar que $g(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$

basta fazer c = 1, e então $3n^3 + 2n^2 \geq n^3$ para $n \geq 1$.

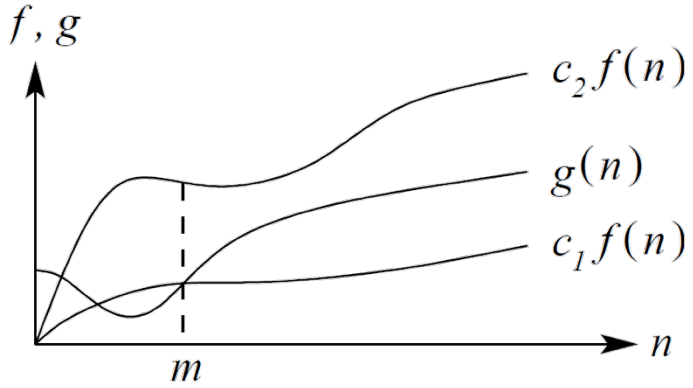
$3 + 2/n \geq 1$ para $n \geq 1$

.

# Notação Θ

- **Definição**: Uma função *g(n)* é Θ*(f(n))* se existirem constantes positivas $c_1$, $c_2$ e m tais que

$$0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n), \text{ para todo } n \geq m$$




- Para todo n ≥ m, a função *g(n)* é igual a *f(n)* a menos de uma constante.
- Neste caso, *f(n)* é um **limite assintótico firme**.

# Notação Θ

G(n) é Θ(f(n)) se $0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$, para todo $n \geq m$, onde c1 e c2 são positivas.

**Exemplo 11**:

- Mostre que $g(n) = n^2/2 - 3n$ é $\Theta(n^2)$.

$$c_1 n^2 \leq n^2/2 - 3n \leq c_2 n^2$$

$$c_1 \leq 1/2 - 3/n \leq c_2$$

O lado direito é verdadeiro para $N \geq 1$ com $c_2 \geq 1/2$ e o lado esquerdo para $n \geq 7$ com $c_1 \leq 1/14$.

Portanto, escolhendo $c_1 = 1/14$, $c_2 = 1/2$ e $m = 7$, $g(n) = \Theta(n^2)$.

# Classes de Comportamento Assintótico

• Se *f* é uma **função de complexidade** para um algoritmo F, então *O(f)* é considerada a **complexidade assintótica**.

• Dominação assintótica permite comparar funções de complexidade

• Se as funções *f* e *g* dominam assintoticamente uma a outra, então os algoritmos associados são equivalentes.

- O comportamento assintótico não serve para comparar esses algoritmos

  Exemplo: $f(n) = n^2$ e $g(n) = (n+1)^2$

# Classes de Comportamento Assintótico

■ Por exemplo, considere dois algoritmos F e G aplicados à mesma classe de problemas, sendo que F leva três vezes o tempo de G ao serem executados, isto é, $f(n) = 3g(n)$, sendo que $O(f(n)) = O(g(n))$.

■ Logo, o comportamento assintótico não serve para comparar os algoritmos F e G, porque eles diferem apenas por uma constante.

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

# Comparação de Programas

■ 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 $100n$.

# Principais Classes de Problemas

- ***f(n) = O(1)***
  - **Complexidade constante**.
  - Uso do algoritmo independe de n.
  - As instruções do algoritmo são executadas um número fixo de vezes.

```
void algoritmo1(int *v, int n){
int i, j, aux;

  for(i = 0; i < 10; i++){
     for(j = 0; j < 9; j++){
         if(v[j]>v[j+1]) {
           aux=v[j]; v[j]=v[j+1]; v[j+1]=aux;
         }
      }
   }
}
```

# Principais Classes de Problemas

- $f(n) = O(n)$
  - **Complexidade linear.**
  - Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada
  - Cada vez que *n* dobra de tamanho, o tempo de execução dobra.

```
int algoritmo2(int *v, int n, int k){
int i;

   for(i = 0; i < n; i++){
      if(v[i] == k)
         return i;
   }
   return -1;
}
```

# Principais Classes de Problemas

- ***f(n) = O(log n)***
  - **Complexidade logarítmica**.
  - Típico em algoritmos que transformam um problema em outros menores.
  - Quando n é mil, $\log_2 n$ 10, quando n é 1 milhão, $\log_2 n$ 20.
  - Exemplo: pesquisa binária

# Principais Classes de Problemas

- ***f(n) = O(n log n)***
  - Típico em algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e ajuntando as soluções depois.
  - Exemplo: algoritmos de ordenação (mergesort, heapsort)

# Principais Classes de Problemas

- ***f(n) = O(n²)***
  - **Complexidade quadrática**.
  - Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro de outro.
  - Úteis para problemas de tamanhos pequenos.

```
void algoritmo(int *v, int n){
int i, j, aux;

  for(i = 0; i < n; i++){
     for(j = 0; j < n-1; j++){
         if(v[j]>v[j+1]) {
           aux=v[j]; v[j]=v[j+1]; v[j+1]=aux;
         }
      }
   }
}
```

# Principais Classes de Problemas

- $f(n) = O(n^3)$
  - **Complexidade cúbica**.
  - Úteis apenas para resolver pequenos problemas.
  - Quando n é 100, o número de operações é da ordem de 1 milhão.
  - Exemplo: multiplicação de matrizes (algoritmo simples)

# Principais Classes de Problemas

- $f(n) = O(2^n)$
  - **Complexidade exponencial**.
  - Geralmente não são úteis sob o ponto de vista prático.
  - Ocorrem na solução de problemas quando se usa **força bruta** para resolvê-los.
  - Quando n é 20, o tempo de execução é cerca de 1 milhão. Quando n dobra, o tempo fica elevado ao quadrado.

# Principais Classes de Problemas

- ***f(n) = O(n!)***
  - Um algoritmo de complexidade O(n!) é dito ter complexidade exponencial, apesar de O(n!) ter comportamento muito pior do que $O(2^n)$.
  - Geralmente ocorrem quando se usa **força bruta** para na solução do problema.
  - n = 20 → 20! = 2432902008176640000, um número com 19 dígitos.
  - n = 40 → 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 séc. |
| $3^n$ | 0,059 s | 58 min | 6,5 anos | 3855 séc. | $10^8$ séc. | $10^{13}$ séc. |

# Comparação de Funções de Complexidade

| Função de custo de tempo | Computador atual (tamanho) | Computador 100 vezes mais rápido | Computador 1.000 vezes mais rápido |
|---|---|---|---|
| $n$ | $t_1$ | $100\ t_1$ | $1000\ t_1$ |
| $n^2$ | $t_2$ | $10\ t_2$ | $31,6\ t_2$ |
| $n^3$ | $t_3$ | $4,6\ t_3$ | $10\ t_3$ |
| $2^n$ | $t_4$ | $t_4 + 6,6$ | $t_4 + 10$ |

# Algoritmos Exponenciais x Polinomiais

- **Algoritmo exponencial** no tempo de execução tem função de complexidade $O(c^n)$; $c > 1$.
- **Algoritmo polinomial** no tempo de execução tem função de complexidade $O(p(n))$, onde $p(n)$ é um polinômio.
- A distinção entre estes dois tipos de algoritmos torna-se significativa quando o tamanho do problema a ser resolvido cresce.
- Por isso, os algoritmos polinomiais são muito mais úteis na prática do que os exponenciais.

# Algoritmos Exponenciais x Polinomiais

- Algoritmos exponenciais são geralmente simples variações de pesquisa exaustiva.
- Algoritmos polinomiais são geralmente obtidos mediante entendimento mais profundo da estrutura do problema.
- Um problema é considerado:
  - **intratável**: se não existe um algoritmo polinomial para resolvê-lo.
  - **bem resolvido**: quando existe um algoritmo polinomial para resolvê-lo.

# Algoritmos Exponenciais x Polinomiais - Exceções

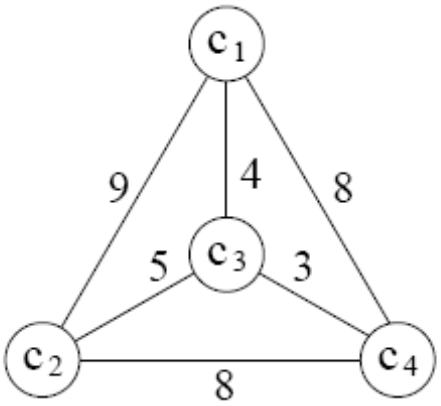
- A distinção entre algoritmos polinomiais eficientes e algoritmos exponenciais ineficientes possui várias exceções.
- **Exemplo**: um algoritmo com função de complexidade $f(n) = 2^n$ é mais rápido que um algoritmo $g(n) = n^5$ para valores de n menores ou iguais a 20.

# Exemplo de Algoritmo Exponencial

- Um **caixeiro viajante** deseja visitar n cidades de tal forma que sua viagem inicie e termine em uma mesma cidade, e cada cidade deve ser visitada uma única vez
- Supondo que sempre há uma estrada entre duas cidades quaisquer, o problema é encontrar a menor rota para a viagem.

# Exemplo de Algoritmo Exponencial

- A figura ilustra o exemplo para quatro cidades *c1*, *c2*, *c3*, *c4*, em que os números nos arcos indicam a distância entre duas cidades.




- O percurso < *c1*, *c3*, *c4*, *c2*, *c1*> é uma solução para o problema, cujo percurso total tem distância 24.

# Exemplo de Algoritmo Exponencial

- **Algoritmo simples:** verificar todas as rotas e escolher a menor delas.
- Há $(n - 1)!$ rotas possíveis e a distância total percorrida em cada rota envolve $n$ adições, logo o número total de adições é $n!$.
- No exemplo anterior teríamos 24 adições.
- Suponha agora 50 cidades: o número de adições seria $50! \approx 10^{64}$.
- Em um computador que executa $10^9$ adições por segundo, o tempo total para resolver o problema com 50 cidades seria maior do que $10^{45}$ séculos só para executar as adições.

# Exercícios

*g*(*n*) é *O*(*f*(*n*)) se *g*(*n*) ≤ *cf*(*n*), para todo *n* ≥ *m*, *onde m e c são positivas*

**Exercício 1**:

- Mostre que $g(n) = \log_5 n$ é $O(\log n)$.

$\log_5 n \leq c \log n$ (temos que $\log_5 n = \log n / \log 5$)

$\log n / \log 5 \leq c \log n$

$1/\log 5 \leq c$, $c = 1$ e m = 1