Projeto e Análise de Algoritmos

Aula 2:
Função de Complexidade
Notação Assintótica (GPV 0.3)

DECOM/UFOP
2013/1 – 5º. Período
Anderson Almeida Ferreira
Adaptado do material feito por
Andréa Iabrudi Tavares

BCC 241/2012-2

Qual o modelo para medir desempenho?

- Medida precisa de um programa executando em um ambiente específico?
  - muitas variáveis externas

- Algoritmo em pseudo-código para RAM
  - Capturar os aspectos essenciais
    - Abstração de hardware, sistema operacional, compilador, linguagem, etc
  - Medir número de passos (operações básicas)
  - Custo constante para passo

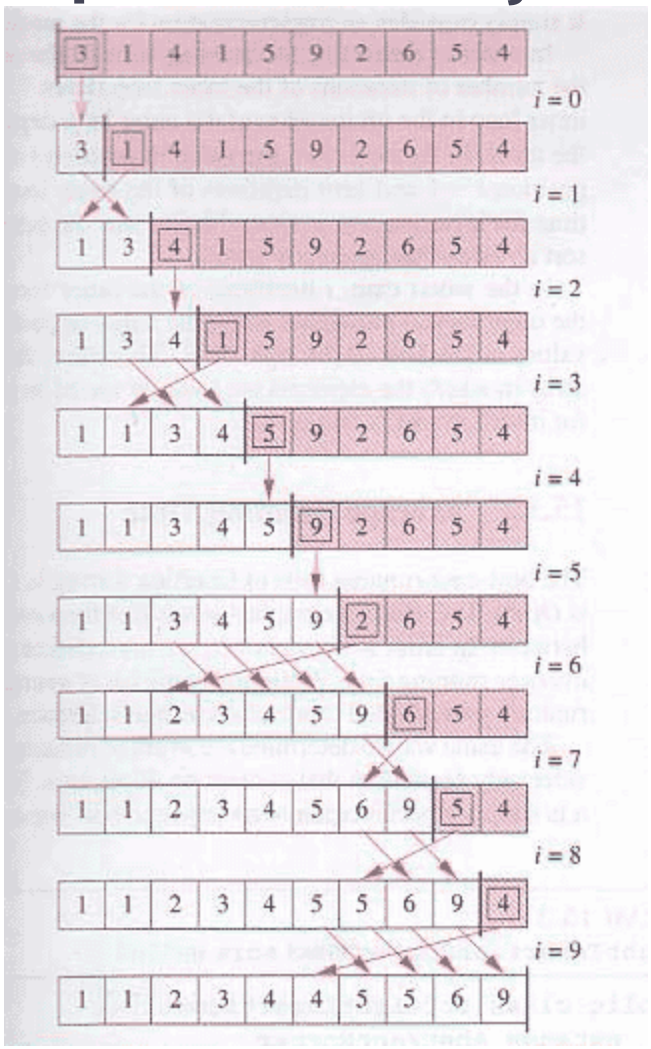
Função de Complexidade $T(n)$

- Número de passos/itens de memória
- Depende do tamanho da entrada
  - Número de elementos
  - Número de bits
  - Número de arestas + vértices
- Depende da entrada: melhor/pior caso/ caso médio

decom
departamento de computação

Exemplo: Ordenação por Inserção

```
function insertion(A[n])
1. for i=2..n:
2.     elem = A[i]
3.     j = i
4.     while (j>1) and (elem< A[j-1]):
5.         A[j] = A[j-1]
6.         j = j -1
7.     A[j] = elem
```



Inserção – Função de complexidade

```
function insertion(A[n])
1. for i=2..n:
2.    elem = A[i]
3.    j = i
4.    while (j>1) and (elem< A[j-1]):
5.       A[j] = A[j-1]
6.       j = j -1
7.    A[j] = elem
```

1		
2		
3		
4		
5		
6		
7		

decom
departamento
de computação

Função de Complexidade

- Caso médio: tempo esperado
- Melhor caso: menor tempo de execução
- Pior caso : maior tempo de execução

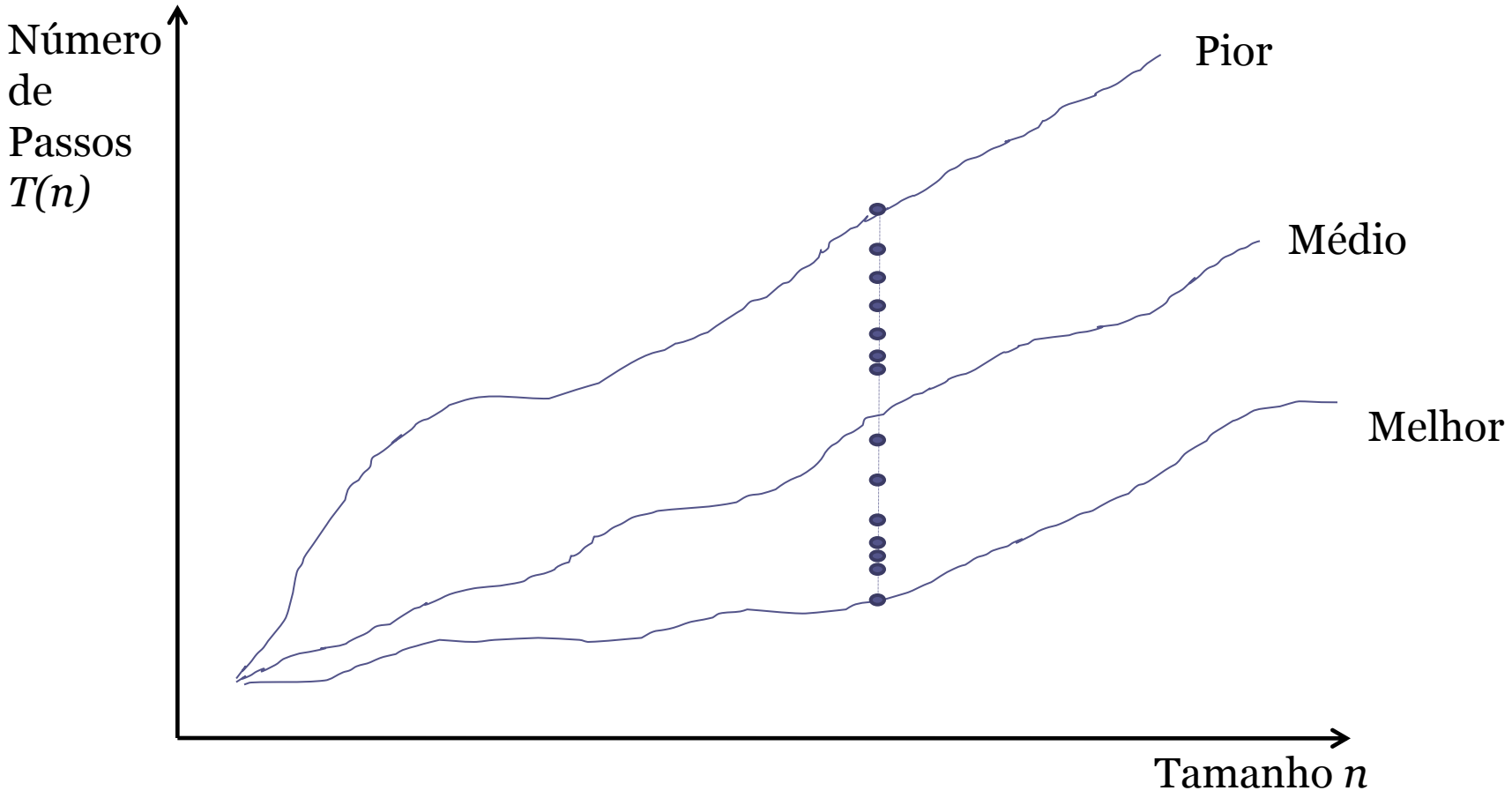
Função de Complexidade

- Exercício. Forneça a função de complexidade de tempo, considerando o número de comparações, do melhor caso, pior caso e caso médio para:

a)
```
function sequencial(V,n, c)
    V[0] = c;
    i = n;
    while V[i] != c:
        i= i – 1;
    return i;
```

b)
```
function Max(A, n)
    Max := A[1];
    for i = 2 .. n:
        if A[i] > Max:
            Max := A[i] ;
    return Max;
```

Funções de Complexidade



Diferentes medidas

- Caso médio: tempo esperado
  - Não se sabe a distribuição da entrada
  - Pode ser difícil de calcular
- Melhor caso: menor tempo de execução
  - Normalmente, pouquíssimas instâncias
  - Pode enganar
- Pior caso : maior tempo de execução
  - garantia de tempo de execução

decom
departamento de computação

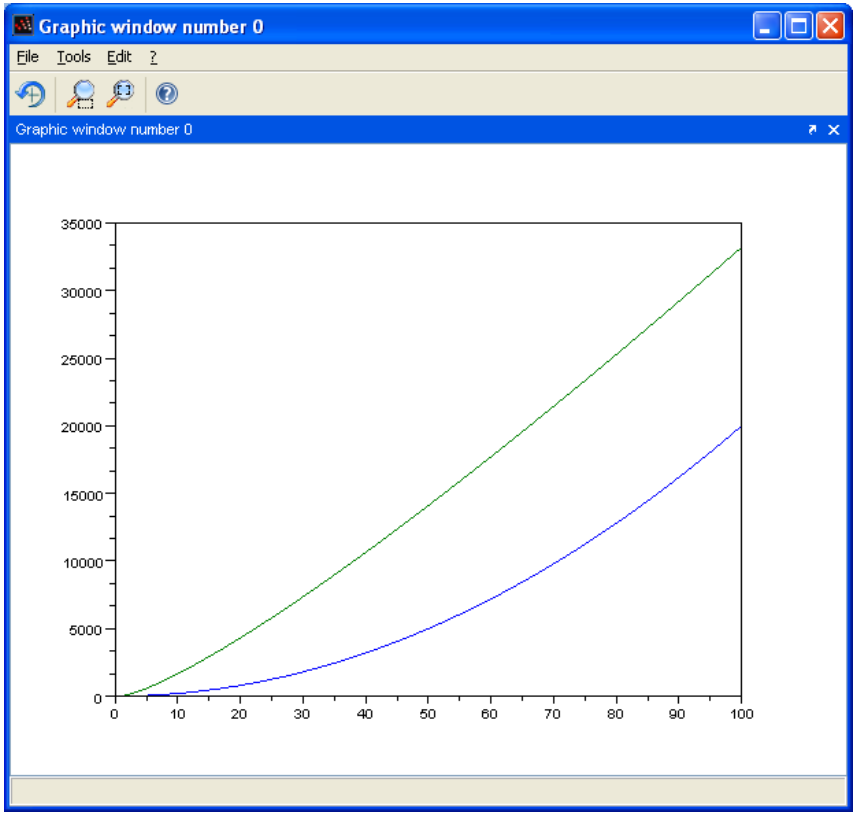
Comparação Efetiva

- Inserção com melhor programador: $T_1(n) = 2n^2$
- MergeSort com programador medíocre: $T_2(n) = 50n \log n$

- No mesmo computador, qual é o melhor se a entrada for 100? E se for 2000?

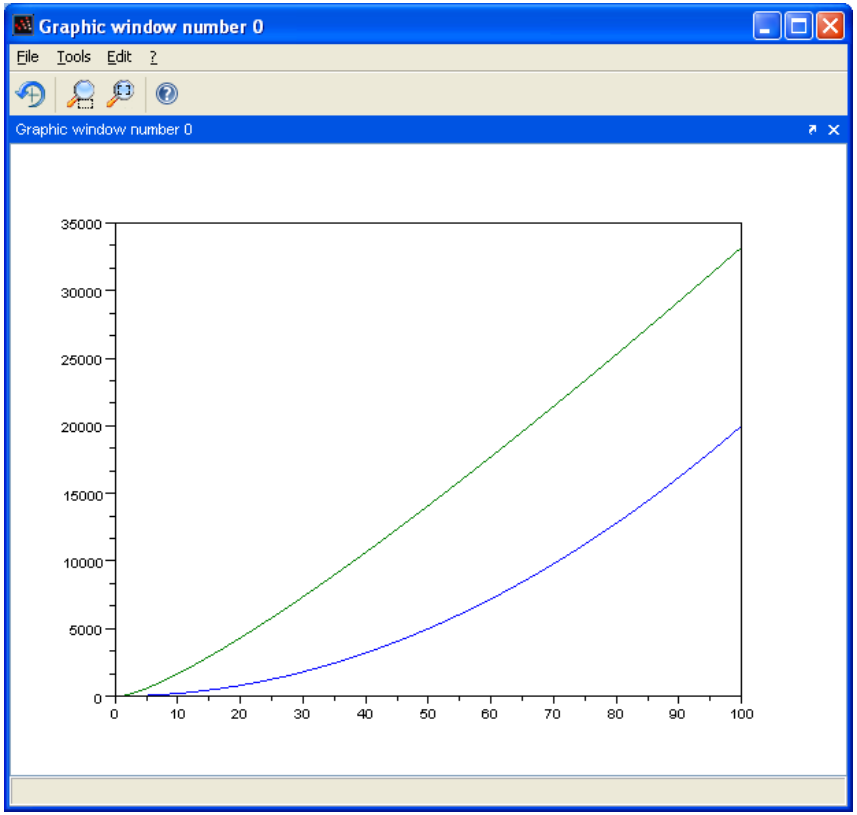
Execução num mesmo computador

Entradas Pequenas

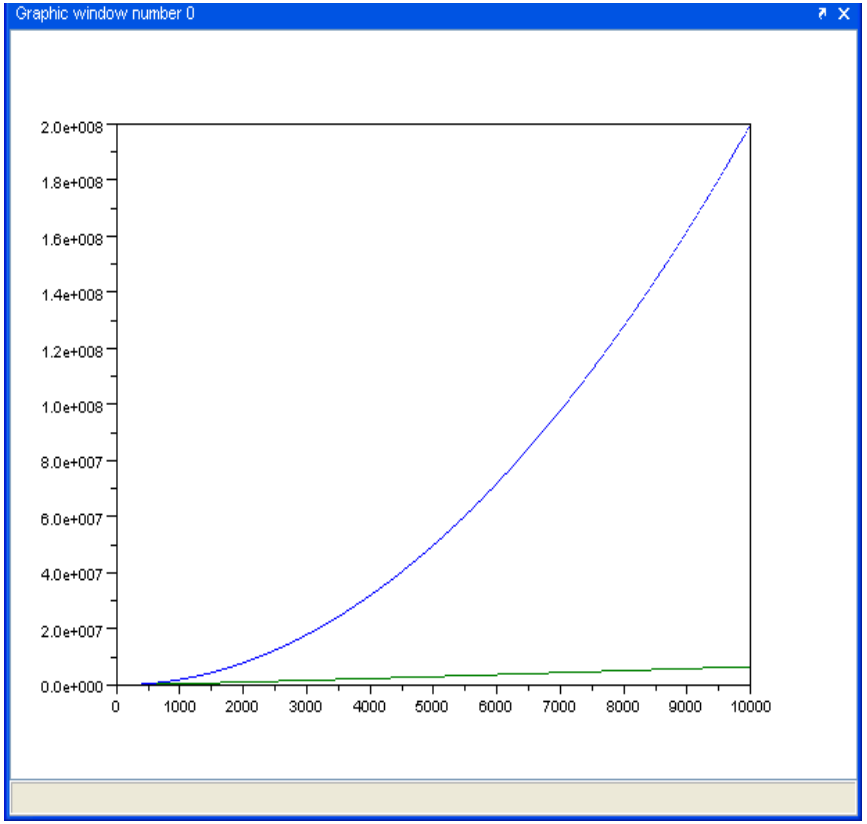


Execução num mesmo computador

Entradas Pequenas



Tendência



Análise Assintótica – "A Idéia"

- Ignorar constantes dependentes de máquina
- Olhar a taxa de crescimento do tempo de execução, expressa em função do tamanho da entrada.
  - Funções: Θ, O, Ω, o, ω

Capturando essência da complexidade...

- Captura a tendência real do desempenho de um algoritmo, quando o tamanho da entrada cresce.
- Em termos práticos:
  - Ignorar constantes e remover termos dominados da função de complexidade

Comportamento assintótico de funções

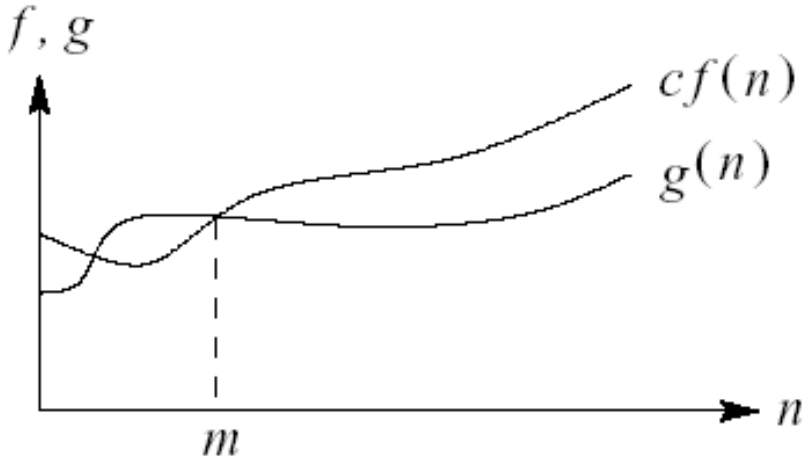
- O parâmetro *n* fornece uma medida da dificuldade para se resolver um problema, já que o tempo necessário para resolvê-lo cresce quando *n* cresce.
  - Para valores suficientemente pequenos de *n*, qualquer algoritmo custa pouco para ser executado, mesmo os ineficientes.
  - A escolha do algoritmo não é um problema crítico para problemas de tamanho pequeno.
- A análise de algoritmos é realizada para valores grandes de *n*.
  - Estuda-se o comportamento assintótico das funções de custo.
  - O comportamento assintótico de *f(n)* representa o limite do comportamento do custo quando *n* cresce.

Comportamento assintótico de funções

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
  - A medida de custo ou medida de complexidade deve relatar o crescimento assintótico da(s) operação(ões) considerada(s).

- A seguinte definição relaciona o comportamento assintótico de duas funções distintas:

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



Comportamento assintótico de funções

- Exemplo 1: Seja $g(n) = n$ e $f(n) = -n^2$.
  - A função $f(n)$ domina assintoticamente a função $g(n)$, já que $|n| \leq 1 \times |-n^2|$ para $n \geq 0$.
  - A função $g(n)$ não domina assintoticamente a função $f(n)$, já que $|-n^2| > c \times |n|$ para todo $n > 1$ e qualquer valor de c.

- Exemplo 2: Seja $g(n) = (n+1)^2$ e $f(n) = n^2$.
  - A função $g(n)$ e $f(n)$ dominam assintoticamente uma a outra, já que:

    $|n^2| \leq 1 \times |(n+1)^2|$ para $n \geq 0$;

    $|(n+1)^2| \leq 4 \times |n^2|$ para $n \geq 1$.

Comportamento assintótico de funções

- Para expressar que $f(n)$ domina assintoticamente $g(n)$, uma notação sugerida por *Knuth* é $g(n) = O(f(n))$, onde se lê "$g(n)$ é da ordem no máximo $f(n)$".
  - Assim, se o tempo de execução $T(n)$ de um programa é $O(n^2)$, existem constantes c e m tais que, para valores de $n \geq m$, $T(n) \leq c \times n^2$.
- **Definição** notação O: Uma função $g(n)$ é $O(f(n))$ se existem duas constantes positivas c e m tais que, para todo $n \geq m$, tem-se $|g(n)| \leq c \times |f(n)|$.
- Exemplo 1: Seja $g(n) = (n+1)^2$.
  - Logo $g(n)$ é $O(n^2)$, já que $|(n+1)^2| \leq 4 \times |n^2|$ para $n \geq 1$.

Comportamento assintótico de funções

- Exemplo 2: Seja $g(n) = 3n^3 + 2n^2 + n$
  - Logo $g(n)$ é $O(n^3)$, já que $|3n^3+2n^2+n| \leq 6 \times |n^3|$ para $n \geq 0$.
  - É importante notar que $g(n)$ também é $O(n^4)$, porém tal fato é mais fraco do que dizer que $g(n)$ é $O(n^3)$.
- Exemplo 3: Seja $g(n) = n^2$
  - Claramente, $g(n)$ é $O(n^2)$, já que $|n^2| \leq 1 \times |n^2|$ para $n \geq 0$.
  - Porém, $g(n)$ não é $O(n)$. Suponha que existam constantes c e m tais que para todo $n \geq m$, $n^2 \leq c \times n$. Logo $c \geq n$ para qualquer $n \geq m$. Mas, não existe $c \geq n$ para todo n.
- Exemplo 4: Seja $g(n) = \log_5 n$
  - $g(n)$ é $O(\log n)$, já que $\log_b n$ difere de $\log_c n$ pela const. $\log_b c$.
  - Como , tomando $\log_b$ em ambos os lados, tem-se:

$$n = c^{\log_c n}$$

$$\log_b n = \log_b c^{\log_c n} = \log_c n \times \log_b c.$$

Comportamento assintótico de funções

- Algumas operações realizadas com a notação *O* são:

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

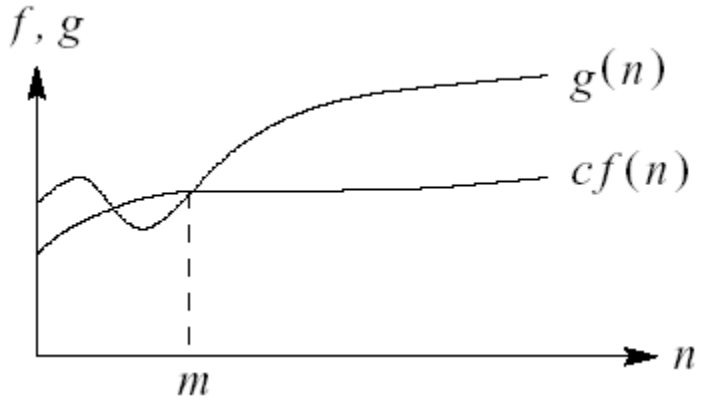
- A regra da soma $O(f(n)) + O(g(n))$ pode ser usada para calcular o tempo de execução de uma sequência de trechos de programa.
  - Suponha trechos cujo tempo de execução são $O(n)$, $O(n^2)$ e $O(n \log n)$. Assim, o tempo de execução do programa é $O(n^2)$.
- O produto de $[\log n + k + O(1/n)]$ por $[n + O(\sqrt{n})]$ é:

$$n \log n + kn + O(\sqrt{n} \log n).$$

Comportamento assintótico de funções

- Dizer que $g(n)$ é $O(f(n))$ significa que $f(n)$ é um limite superior para $g(n)$. A notação Ω especifica o limite inferior.

- **Definição** notação Ω: Uma função $g(n)$ é $\Omega(f(n))$ se existem duas constantes positivas c e m tais que, para todo $n \geq m$, tem-se $|g(n)| \geq c \times |f(n)|$.

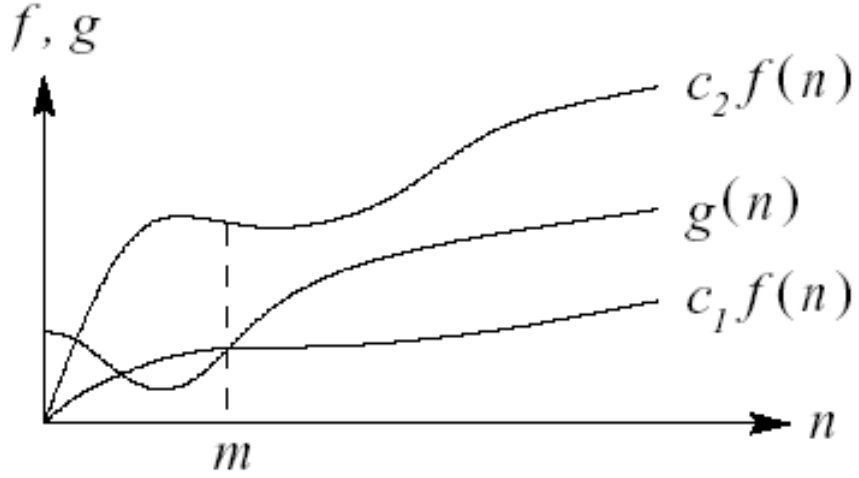


- Exemplo 1: Seja $g(n) = 3n^3 + 2n^2$
  - Logo $g(n)$ é $\Omega(n^3)$, já que $|3n^3+2n^2| \geq 1 \times |n^3|$ para $n \geq 0$.

- Exemplo 2: Seja $g(n) = n$ (n ímpar) e $g(n) = n^2/10$ (n par)
  - Logo $g(n)$ é $\Omega(n^2)$, já que $|n^2/10| \geq 1/10 \times |n^2|$ para n par ≥ 0.

Comportamento assintótico de funções

- **Definição** notação Θ: Uma função $g(n)$ é $\Theta(f(n))$ se existirem constantes positivas c_1, c_2 e m tais que, para todo $n \geq m$, tem-se $0 \leq c_1 \times f(n) \leq g(n) \leq c_2 \times f(n)$.



- Para todo $n \geq m$, $g(n)$ é igual a $f(n)$ a menos de uma constante. Nesse caso, $f(n)$ é considerado um **limite assintótico firme**.

Comportamento assintótico de funções

- Exemplo: Seja $g(n) = n^2/3 - 2n$.
  - Para mostrar que $g(n) = \Theta(n^2)$, deve-se determinar valores para as constantes c_1, c_2 e m tais que: $c_1 n^2 \leq \frac{1}{3}n^2 - 2n \leq c_2 n^2$ para todo $n \geq m$.
  - Dividindo por n^2, tem-se: $c_1 \leq \frac{1}{3} - \frac{2}{n} \leq c_2$.
  - O lado direito da desigualdade será sempre válido para qualquer valor de $n \geq 1$, ao se escolher $c_2 \geq 1/3$.
  - O lado esquerdo da desigualdade será válido para qualquer valor de $n \geq 7$, ao se escolher $c_1 \leq 1/21$.
  - Logo, escolhendo $c_1 = 1/21$, $c_2 = 1/3$ e $m = 7$, é possível verificar que $g(n) = \Theta(n^2)$.
  - Outras constantes podem existir, mas o importante é que existe alguma escolha para as três constantes.

Comportamento assintótico de funções

- A notação *o* é usada para definir um limite superior que não é assintoticamente firme.
- **Definição** notação *o*: Uma função $g(n)$ é $o(f(n))$ se, para qualquer constante $c > 0$, então $0 \leq g(n) < c \times f(n)$ para todo $n \geq m$.
- Exemplo: $2n = o(n^2)$, mas $2n^2 \neq o(n^2)$.
- As notações *O* e *o* são similares.
  - A diferença é que em $g(n) = O(f(n))$, a expressão "$0 \leq g(n) < c \times f(n)$" é válida para alguma constante $c > 0$, mas em $g(n) = o(f(n))$, tal expressão é válida para todas as constantes $c > 0$.
- Na notação *o*, a função $g(n)$ tem um crescimento muito menor que $f(n)$ quando n tende para infinito.
  - Alguns autores usam o seguinte limite para definição da notação *o*:

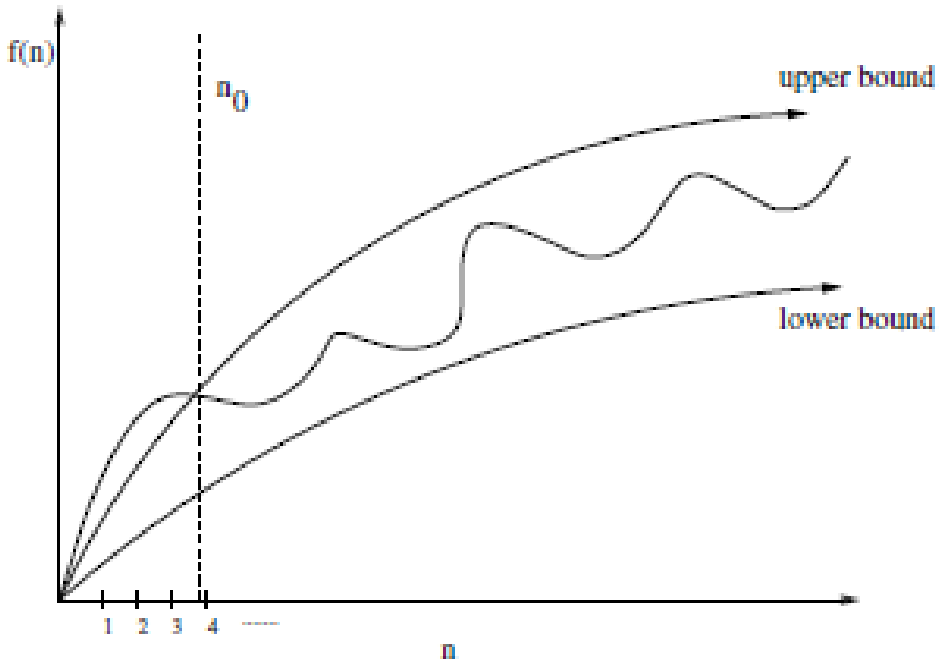
$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

Comportamento assintótico de funções

- A notação ω é usada para definir um limite inferior que não é assintoticamente firme.
- **Definição** notação ω**:** Uma função $g(n)$ é $\omega(f(n))$ se, para qualquer constante $c > 0$, então $0 \leq c \times f(n) < g(n)$ para todo $n \geq m$.
- Exemplo: $n^2/2 = \omega(n)$, mas $n^2/2 \neq \omega(n^2)$.
- As notações Ω e ω são similares.
  - A diferença é que em $g(n) = \Omega(f(n))$, a expressão "$0 \leq c \times f(n) < g(n)$" é válida para alguma constante $c > 0$, mas em $g(n) = \omega(f(n))$, tal expressão é válida para todas as constantes $c > 0$.
- Na notação ω, a função $f(n)$ tem um crescimento muito menor que $g(n)$ quando n tende para infinito.
  - Alguns autores usam o seguinte limite para definição da notação ω:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$$

Limites são mais suaves...



Revisão: Análise de Comandos Iterativos

- Atribuição: *O(1)*
- Sequência: regra da soma

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$
$$\Omega(f(n)) + \Omega(g(n)) = \Omega(\max(f(n), g(n)))$$

Revisão: Análise de Comandos Iterativos

- Decisão
  - Condição
  - Melhor (mínimo), pior (máximo), médio (?)

Revisão: Análise de Comandos Iterativos

- Repetição: regra da multiplicação
  - Somatórios
  - Dependência de limite
  - While (pior, melhor, médio)

$$O(cf(n)) = O(f(n))$$

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

Revisão: Análise de Comandos Iterativos

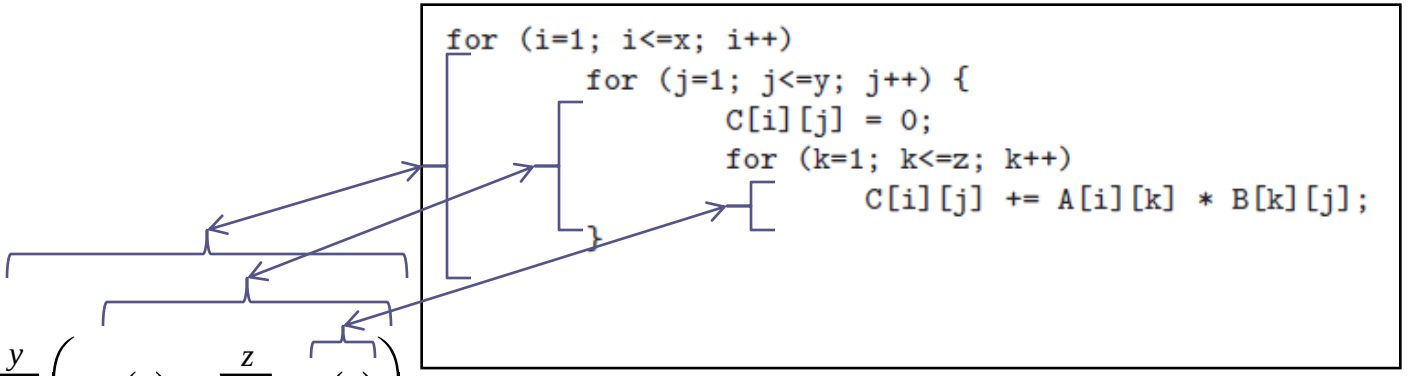
- Funções não-recursivas
  - Calcula complexidade da função e atribui ao comando de chamada
- Funções recursivas:
  - Relações de recorrência
  - Teorema mestre

Multiplicação de Matrizes

```
for (i=1; i<=x; i++)
        for (j=1; j<=y; j++) {
                C[i][j] = 0;
                for (k=1; k<=z; k++)
                        C[i][j] += A[i][k] * B[k][j];
        }
```

- Multiplicação de duas matrizes $C_{x \times z} = A_{x \times y} \times B_{y \times z}$
- Matrizes começam no índice 1

Multiplicação de Matrizes



```
for (i=1; i<=x; i++)
        for (j=1; j<=y; j++) {
                C[i][j] = 0;
                for (k=1; k<=z; k++)
                        C[i][j] += A[i][k] * B[k][j];
                }
```

$$T(x,y,z) = \sum_{i=1}^{x}\sum_{j=1}^{y}\left(\Theta(1) + \sum_{k=1}^{z}\Theta(1)\right)$$

$$T(x,y,z) = \sum_{i=1}^{x}\sum_{j=1}^{y}\left(\Theta(1) + z\Theta(1)\right)$$

$$T(x,y,z) = \sum_{i=1}^{x}\sum_{j=1}^{y}\left(\Theta(1) + \Theta(z)\right) = \sum_{i=1}^{x}\sum_{j=1}^{y}\left(\Theta(z)\right)$$

$$T(x,y,z) = \sum_{i=1}^{x} y\Theta(z) = \sum_{i=1}^{x}\Theta(yz)$$

$$T(x,y,z) = x\Theta(yz) = \Theta(xyz)$$

decom
departamento
de computação

Pesquisa em texto

```
int findmatch(char *p, char *t)
{
        int i,j;                                /* counters */
        int m, n;                               /* string lengths */

        m = strlen(p);
        n = strlen(t);

        for (i=0; i<=(n-m); i=i+1) {
                j=0;
                while ((j<m) && (t[i+j]==p[j]))
                        j = j+1;
                if (j == m) return(i);
        }

        return(-1);
}
```

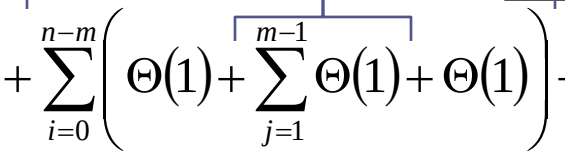
Pesquisa em texto

```
int findmatch(char *p, char *t)
{
        int i,j;                        /* counters */
        int m, n;                       /* string lengths */

        m = strlen(p);
        n = strlen(t);

        for (i=0; i<=(n-m); i=i+1) {
                j=0;
                while ((j<m) && (t[i+j]==p[j]))
                        j = j+1;
                if (j == m) return(i);
        }

        return(-1);
}
```



$$T(n,m) = \Theta(n) + \Theta(m) + \sum_{i=0}^{n-m} \left(\Theta(1) + \sum_{j=1}^{m-1} \Theta(1) + \Theta(1) \right) + \Theta(1)$$

$$T(n,m) = \Theta(n) + \sum_{i=0}^{n-m} \left(\Theta(1) + (m-1)\Theta(1) \right)$$

$$T(n,m) = \Theta(n) + \sum_{i=0}^{n-m} \left(\Theta(1) + \Theta(m) \right) = \Theta(n) + \sum_{i=0}^{n-m} \left(\Theta(m) \right)$$

$$T(n,m) = \Theta(n) + (n-m+1)\Theta(m) = \Theta(n) + \Theta(nm - m^2 + m)$$

$$T(n,m) = \Theta(n) + \Theta(nm) = \Theta(nm)$$