025089 – Projeto e Análise de Algoritmos

Aula 03

lizier@dc.ufscar.br

Tipos de análises

- Melhor caso
  - Determinado pelos dados de entrada que levam ao menor número de passos
  - Provê um limitante inferior, ou seja, o menor número de passos que o algoritmo executará
- Pior caso
  - Determinado pelos dados de entrada que levam ao maior número de passos
  - Provê um limitante superior, ou seja, considerando qualquer entrada possível, este será o maior tempo necessário de processamento

Tipos de análises

- Caso médio
  - Custo esperado em médio (muitas vezes difícil de definir!)

- Abordagens:
  - Apenas a avaliação do pior caso é necessária
    - Busca sequencial: elemento não encontrado!
  - Apenas o caso médio é importante
    - Quicksort
  - Todos os os casos precisam ser avaliados

Análise assintótica

- Útil quando analisamos para N **muito** grande
- Todos os fatores constantes e termos de ordem inferior são eliminados
- Aproximamos a função do modelo de custo por uma das funções “básicas”
- Criação de famílias de funções: constantes, logarítmicas, lineares, quadráticas, exponenciais, etc ...

Notação O
Big-Oh

Obs: Operador "ϵ" e "=" são utilizados como Equivalentes!

- O <u>conjunto</u> de **funções** $O(g(n))$ é definido como:

  $O(g(n)) = \{ f(n)$: existem **constantes** positivas c e n_0, tal que $0 \leq f(n) \leq c\,g(n)$, para todo $n \geq n_0 \}$

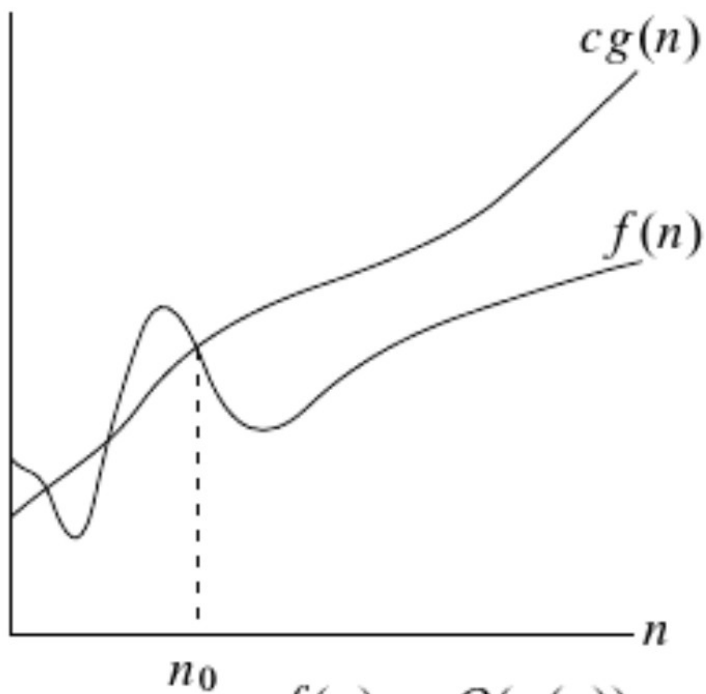
- Exemplos:

  $n^2+n+10 \quad \epsilon \quad O(n^2)$

  $3n^3 + n2 \quad \epsilon \quad O(n^3)$

  $50logN + 30 \quad \epsilon \quad O(logN)$

  $1/3\ NlogN + 4N \quad \epsilon \quad O(NLogN)$



$f(n) = O(g(n))$

Notação Ω
Big-Omega

- O <u>conjunto</u> de **funções** $\Omega(\, g(n)\,)$ é definido como:

  $\Omega(\, g(n)\,) = \{\, f(n)$: existem **constantes** positivas c e n_0, tal que $0 \le c\, g(n) \le f(n)$, para todo $n \ge n_0\, \}$

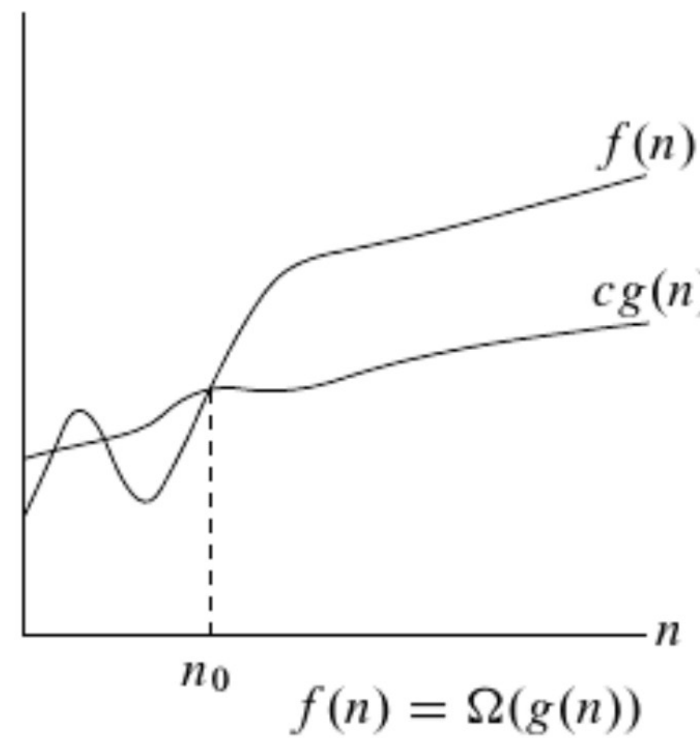
- Exemplos:

  $n^2+n+10 \quad \epsilon \quad \Omega(n^2)$

  $3n^3 + n2 \quad \epsilon \quad \Omega(n^3)$

  $50\log N + 30 \quad \epsilon \quad \Omega(\log N)$

  $1/3\ N\log N + 4N \quad \epsilon \quad \Omega(N\mathrm{Log}N)$



Notação Θ
Theta

- O <u>conjunto</u> de **funções** $\Theta(\ g(n)\ )$ é definido como:

  $\Theta(\ g(n)\ ) = \{\ f(n)$: existem **constantes** positivas c_1, c_2 e n_0, tal que $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, para todo $n \ge n_0\ \}$

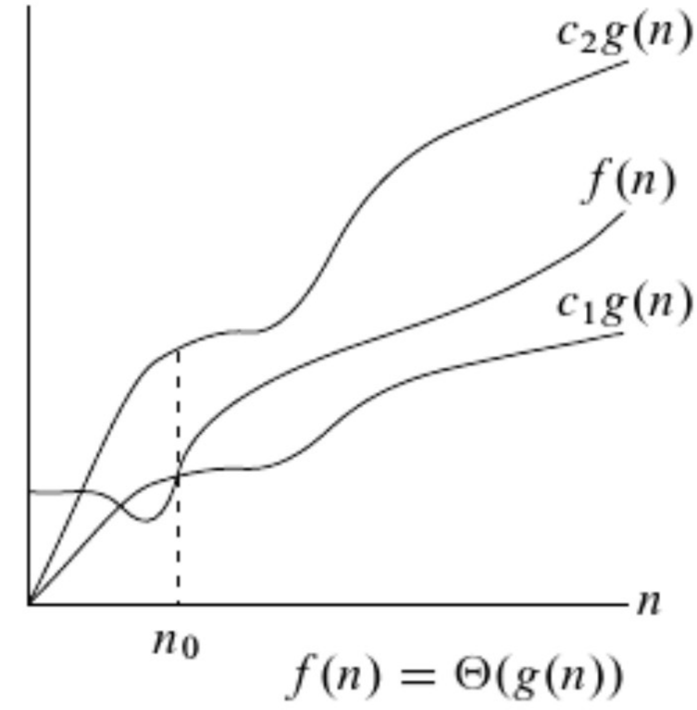
- Exemplos:

  $n^2+n+10 \quad \epsilon \quad \Theta(n^2)$

  $3n^3 + n2 \quad \epsilon \quad \Theta(n^3)$

  $50logN + 30 \quad \epsilon \quad \Theta(logN)$

  $1/3\ NlogN + 4N \quad \epsilon \quad \Theta(NLogN)$



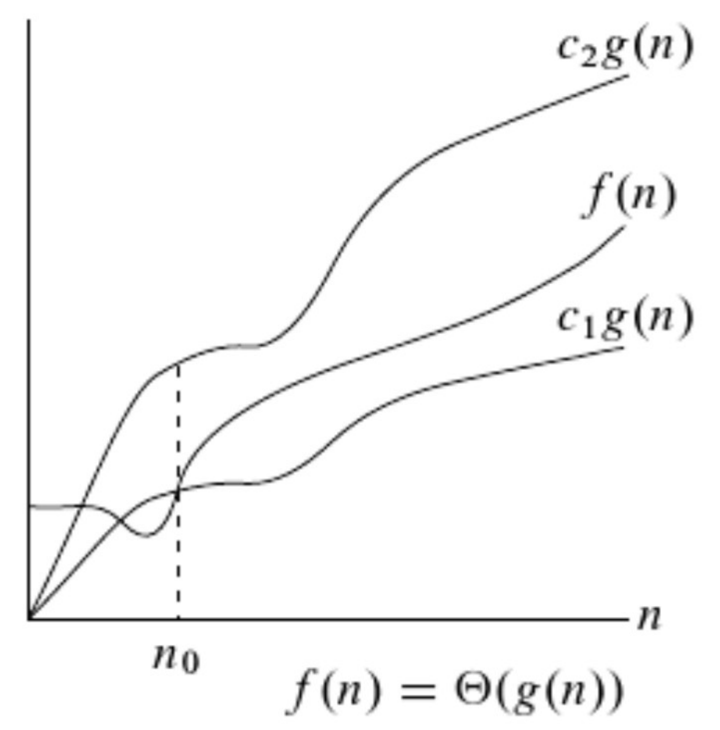
Notação Θ
Theta

- Outra definição:

  Para quaisquer funções *f(n)* e *g(n)*,

  temos $f(n) = \Theta(g(n))$, se e somente se,

  $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.



Observações sobre O, Ω e Θ

- Embora possam parecer “redundantes”, cada problema requer uma notação
- Ao avaliar o pior caso, a utilização de O geralmente é mais apropriada
- Ao avaliar o melhor caso, a utilização de Ω pode ser mais apropriada
- O uso de Θ já indica que conseguimos limitar “por baixo” e “por cima” a função
- As minúsculas o e ω indicam que uma aproximação muito fraca foi feita, para O e Ω respectivamente.

Exemplos *o* e *ω*

$$10N^2 + n + 5 = o(N^3) = O(N^2)$$

$$10N^2 + n + 5 = \omega(N) = \Omega(N^2)$$

- Reparem que pelas definições aqui apresentadas de *O*, *Ω* e *Θ*, *estas possibilidades de "péssima" aproximações não estavam sendo consideradas!*

Análise de algoritmos

- Devemos utilizar a família de função que mais se “ajusta” a função que devemos aproximar

Forte!

- Logo, é errado dizer que:

$$n^2 = O(n^3)$$

$$n^2 = \Omega(n)$$

Propriedades relacionais

Transitivity:

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \quad \text{imply} \quad f(n) = \Theta(h(n)) \, ,$$

$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \quad \text{imply} \quad f(n) = O(h(n)) \, ,$$

$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \quad \text{imply} \quad f(n) = \Omega(h(n)) \, ,$$

$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \quad \text{imply} \quad f(n) = o(h(n)) \, ,$$

$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \quad \text{imply} \quad f(n) = \omega(h(n)) \, .$$

Propriedades relacionais

Reflexivity:

$$
\begin{aligned}
f(n) &= \Theta(f(n)) \,, \\
f(n) &= O(f(n)) \,, \\
f(n) &= \Omega(f(n)) \,.
\end{aligned}
$$

Propriedades relacionais

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$

Transpose symmetry:

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)) ,$$

$$f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)) .$$

Modelo matemático

- <u>Teoria</u>: temos disponível uma modelagem matemática precisa, com maior rigor
- <u>Prática</u>:
  - As fórmulas podem se tornar muito complicadas
  - Utilizamos **aproximações**

Considerações finais

- Os dois modelos são importantes!
  - O modelo matemático é independente do sistema
  - O modelo experimental (análise empírica) é necessária para validar o modelo matemático e fazer um prognóstico futuro

Exemplo: busca sequencial

- <u>Aproximadamente</u> quantas <u>comparações</u> são realizadas?

```
int bsequencial( T vetor[], T key, int n )
{
   int i = 0;
   while( (i < n) && (vetor[i] < key) )         1 até 2*N+1
      i++;
   if ( (i < n) && (vetor[i] == key ) )         1 até 2
      return i;
   else
      return -1;
}
```

Total: 2 até 2*N+2 ~2*N

Exemplo: busca sequencial

- Melhor caso: quando **key** está na posição 0

  2 comparações: f(n) é constante!

- Pior caso: quando **key** não está presente

  ~2*N comparações: f(n) é linear!

- Caso médio: dependerá da aplicação, mas considerando algo totalmente randômico, teríamos ~N comparações, ou seja, linearmente dependente do tamanho do vetor

- Neste caso, é apropriado utilizar O(N) para expressar a complexidade deste algoritmo

Exemplo: busca binária

- <u>Aproximadamente</u> quantas <u>comparações</u> são realizadas?

```
int bbinaria( T vetor[], T key, int n )
{
    int imax = n-1;
    int imin = 0;
    while( imax >= imin )
    {
        int imid = imin + ((imax - imin) / 2);
        if( key > vetor[imid] )
            imin = imid + 1;
        else if( key < vetor[imid])
            imax = imid - 1;
        else
            return imid;
    }
    return -1;
}
```

Exemplo: busca binária

- <u>Melhor caso</u>: o elemento **key** está no meio do vetor: com um número constante de comparações encontramos! (exatamente 3)

- <u>Pior caso</u>: quando não encontramos um elemento, o *loop* executa "por completo", pois não é interrompido no *return imid;*

Exemplo: busca binária

- Pior caso:

  Descemos toda a árvore binária, ou seja, um número de comparações proporcional a $\log_2 N$

Memória

- Além do tempo de processamento, o modelo de custo pode ser o consumo de memória
- Os dois algoritmos apresentados (busca sequencial e binária) não requerem memória proporcional as entradas (constante), excluindo o próprio vetor de dados, que cresce linearmente em relação ao número de elementos
- No exemplo a seguir temos uma versão recursiva da busca binária:

Busca binária: versão recursiva

- Qual a taxa de crescimento do uso de memória?

```
int bbinariarec( T vetor[], T key, int imin, int imax)
{
    if (imax >= imin)
    {
        int imid = imin + ((imax - imin) / 2);
        if ( key > vetor[imid] )
            return bbinariarec(vetor, key, imid+1, imax);
        else if ( key < vetor[imid] )
            return bbinariarec(vetor, key, imin, imid-1);
        else
            return imid;
    }
    return -1;
}
```

Análise de Algoritmos não-recursivos

- Escolha do parâmetro (ou parâmetros) de entrada que define o tamanho do problema (ou entrada)
- Identificação a operação básica (mais frequente) do algoritmo
- Contagem do número de vezes que a operação básica é executada, dependendo apenas do tamanho da entrada. Se depender também do tipo da entrada, analise o melhor e pior caso separadamente
- Construa o somatório que expressa o número de execuções da operação básica
- Faça as manipulações necessárias para aproximar e classificar corretamente a ordem de crescimento do algoritmo

Exemplo 1

ALGORITHM $MaxElement(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

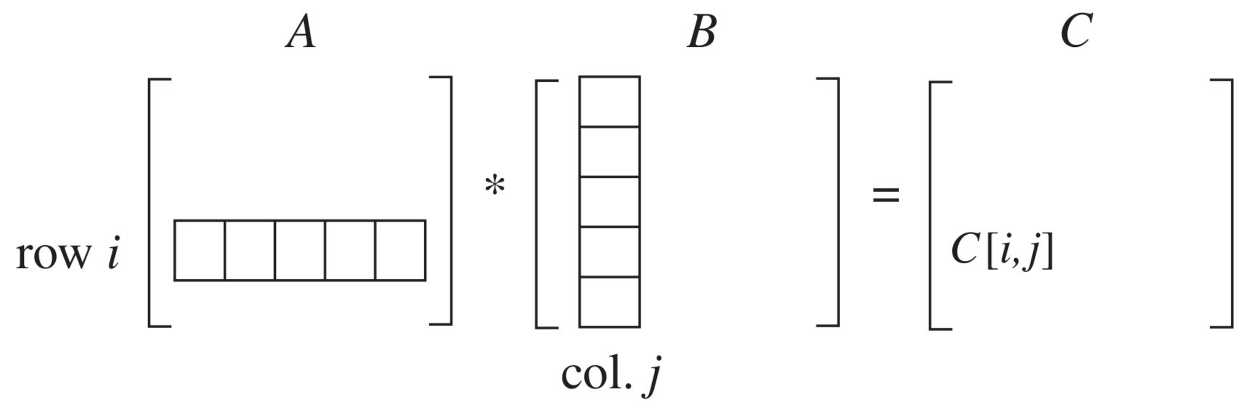
Exemplo 2

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
//   and "false" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] = A[j]$ **return false**
return true

Exemplo 3



ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two $n \times n$ matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
        $C[i, j] \leftarrow 0.0$
        for $k \leftarrow 0$ **to** $n-1$ **do**
            $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
return C

Exemplo 4

ALGORITHM $Binary(n)$

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

$count \leftarrow 1$

while $n > 1$ **do**

  $count \leftarrow count + 1$

  $n \leftarrow \lfloor n/2 \rfloor$

return $count$