

# Projeto e Análise de Algoritmos

## Aula 1: Panorama (0.1,0.2)

DECOM/UFOP

2012/2 – 5<sup>o</sup>. Período

Anderson Almeida Ferreira

Material desenvolvido por Andréa  
Iabrudi Tavares

BCC 241 2012/2



# Algumas Fontes

- O material apresentado no curso usa recursos disponíveis na rede, em especial:
  - Skiena (State University of New York)
  - MIT (Vídeos do curso 6.046)
  - Koerich (PUC-PR)
  - Lee, MC-448 (Unicamp)
- Usa material cedido pela profa. Andrea, pelo prof. Elton e prof. David

# Objetivo último da Ciência da Computação

- Projetar e implementar programas corretos (geram a solução esperada) e eficientes (executam em tempo e espaço aceitáveis) para problemas.

# Projeto de algoritmos

- Problema tem subproblemas conhecidos (decomposição).
- Problema se transforma em outro (redução).
- Eficiência e correção.
- Problemas levemente diferentes – complexidade e algoritmos totalmente diferentes.

# Diferenças sutis, complexidades extremas

| Tratável                          | Intratável                          |
|-----------------------------------|-------------------------------------|
| 2-SAT                             | 3-SAT                               |
| Coloração com 2 cores             | Coloração com 3 cores, mesmo planar |
| Menor caminho entre dois vértices | Maior caminho entre dois vértices   |
| Cobertura de arestas              | Cobertura de vértices               |

# Análise de algoritmos

- Calcular os recursos consumidos por um algoritmo.
  - tempo, memória, banda de comunicação...
  - em função do “tamanho” da entrada (instância)
- Comparar diferentes algoritmos, de forma independente da tecnologia.

# Complexidade computacional

- Identificar qual é a dificuldade de resolver um problema.
- Problemas tratáveis (soluções polinomiais)
  - Algoritmos específicos
- Problemas intratáveis
  - Técnicas genéricas de exploração
  - Heurísticas para solução aproximada

# O que é importante além de desempenho?

- robustez,
- modularidade,
- simplicidade,
- manutenabilidade,
- extensibilidade,
- segurança,
- custo (homens-hora),
- usabilidade (user-friendliness)...



# Então, por que desempenho é tão importante?

- Limites estritos de desempenho
- Solução exata ou aproximada
- Permite “pagar” pelas outras características

# O que é um algoritmo?

- Procedimento preciso, não-ambíguo, mecânico, eficiente e correto para achar a solução de um problema.
  - Procedimento = conjunto finito de instruções.
  - Achar a solução = gerar a função.
- Homenagem a Al Khwarizmi (século XII)

# O que é um problema?

- Problema
  - Descreve, de forma geral, o mapeamento desejado de entradas em saídas.
  
- Instância e solução
  - Entrada específica de um problema.
  - Saída (única) associada a uma instância.

# Ordenação: Problema

*Problem:* Sorting

*Input:* A sequence of  $n$  keys  $a_1, \dots, a_n$ .

*Output:* The permutation (reordering) of the input sequence such that  $a'_1 \leq a'_2 \leq \dots \leq a'_{n-1} \leq a'_n$ .

# Fibonacci: Problema

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 . \end{cases}$$

# O que é um algoritmo? (II)

- Como: “Expresso” em pseudo-código
  - Independência de linguagem
  - Convenção (simplicidade, legibilidade)
  - Operações básicas
- Onde: “Executa” em uma máquina virtual
  - RAM (Random Access Machine)
  - Abstração de hardware, sistema operacional, compilador, linguagem, para que seja capturada a essência de desempenho/correção do procedimento.

# Fibonacci: Algoritmo 1

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0. \end{cases}$$

```
function fib1(n)  
if n = 0: return 0  
if n = 1: return 1  
return fib1(n - 1) + fib1(n - 2)
```

# Perguntas a serem feitas:

1. Está correto?
  - Pergunta importantíssima
    - Não é assunto deste curso, mas veremos rapidamente intuições de prova, quando possível.
    - Indução é nossa maior amiga ;).
2. Qual a eficiência (tempo e espaço)?
3. Pode ser feito melhor?



# Fibonacci: Qual a eficiência?

```
function fib1(n)  
if  $n = 0$ : return 0  
if  $n = 1$ : return 1  
return fib1( $n - 1$ ) + fib1( $n - 2$ )
```

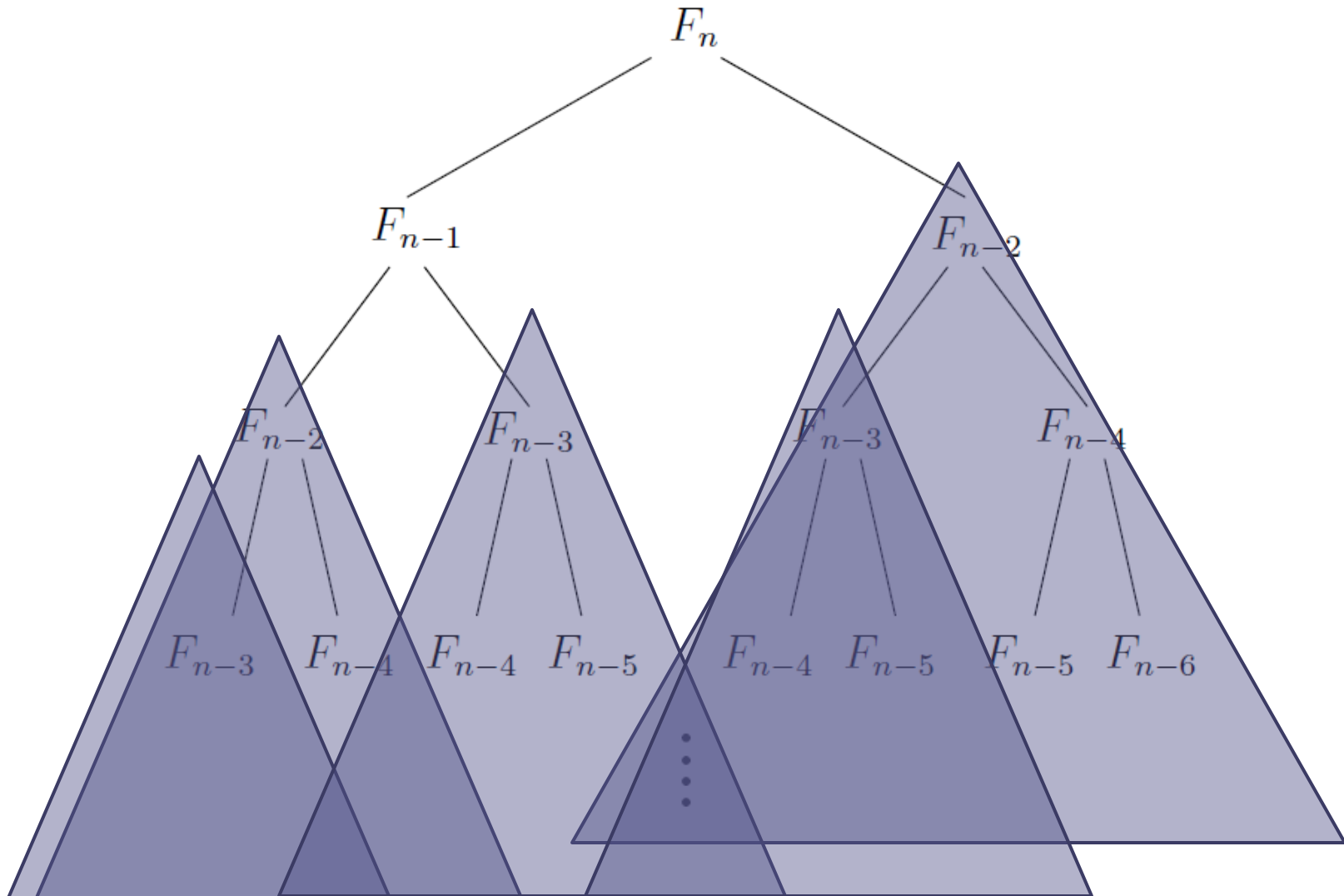
## Paradigma Divisão e Conquista

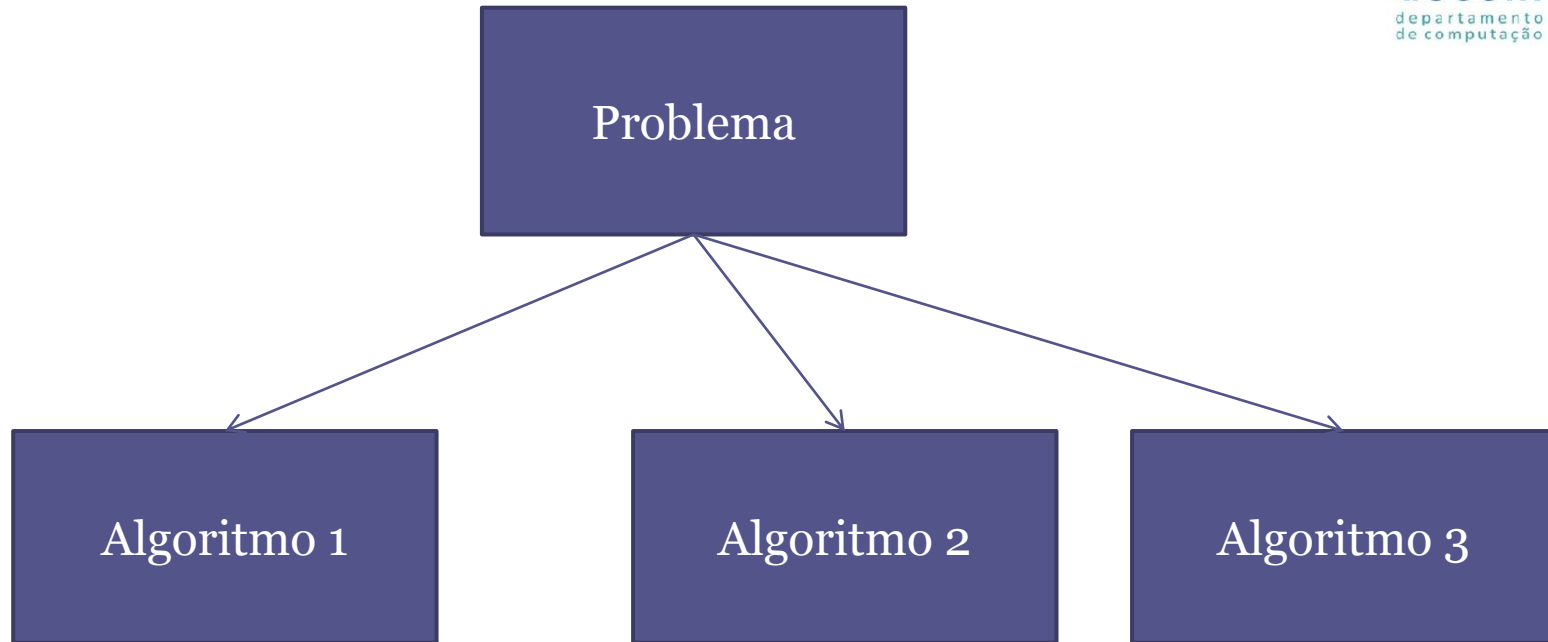
$$T(n) \leq 2 \text{ for } n \leq 1.$$

$$T(n) = T(n - 1) + T(n - 2) + 3 \text{ for } n > 1.$$

$$2^{0.694n} \approx (1.6)^n$$

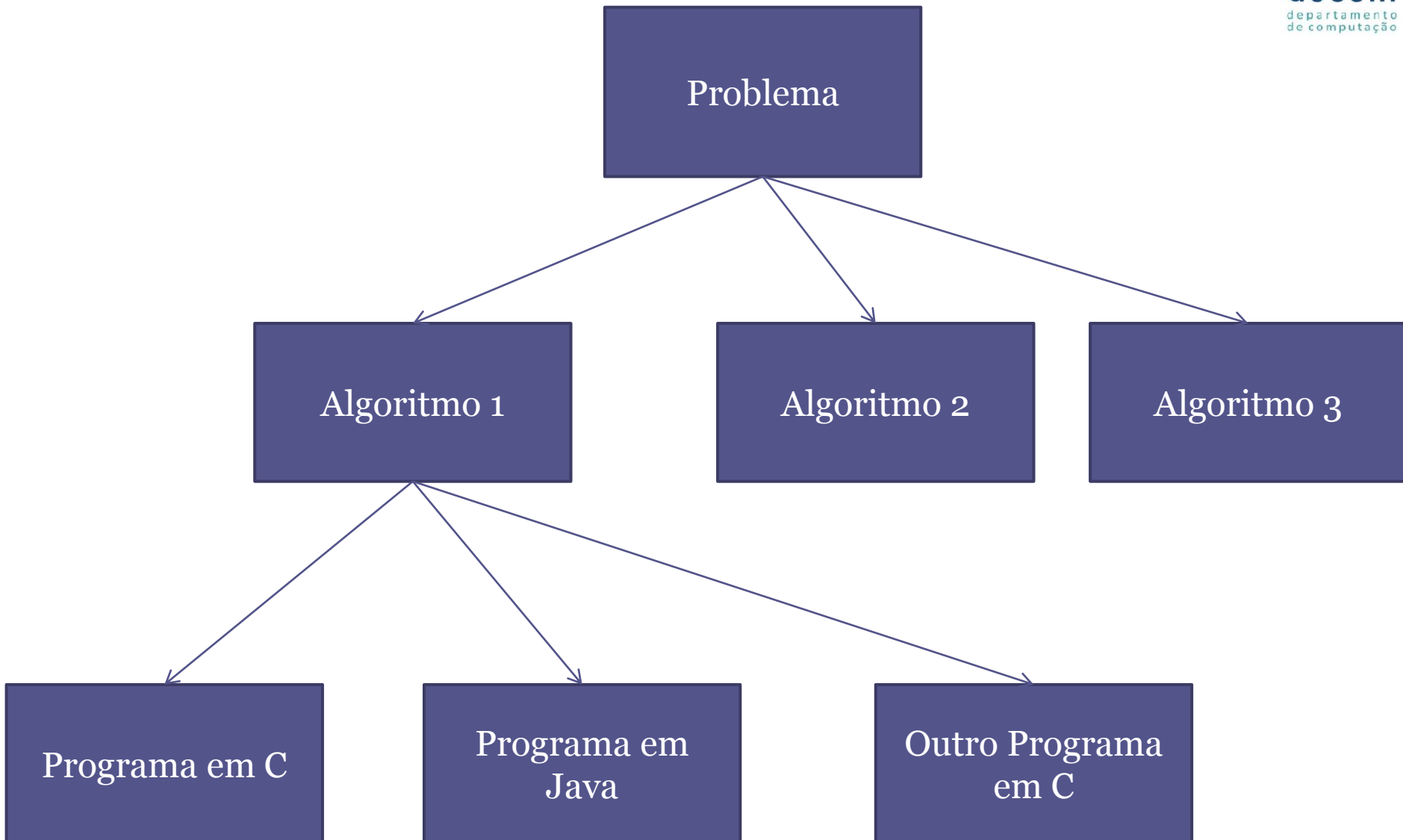
# Por quê? Recálculo





# Algoritmo ou programa?

- Algoritmo é abstrato
  - Preocupação com correção e eficiência teórica.
- Programa é concreto
  - Linguagem, sistema operacional, hardware, etc...
  - Muitas e muitas outras preocupações envolvidas



# Fibonacci: Algoritmo 2

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0. \end{cases}$$

```
function fib2(n)  
if  $n = 0$  return 0  
create an array  $f[0..n]$   
 $f[0] = 0, f[1] = 1$   
for  $i = 2..n$ :  
     $f[i] = f[i-1] + f[i-2]$   
return  $f[n]$ 
```

# Função de complexidade fib2

- Paradigma de Programação Dinâmica
- Polinomial
- Linear ou quadrática?

# Efeito da Complexidade

| Função de complexidade | Tamanho da Instância do Problema |                     |                     |                     |                            |                                 |
|------------------------|----------------------------------|---------------------|---------------------|---------------------|----------------------------|---------------------------------|
|                        | 10                               | 20                  | 30                  | 40                  | 50                         | 60                              |
| $n$                    | 0,00001<br>segundos              | 0,00002<br>segundos | 0,00003<br>segundos | 0,00004<br>segundos | 0,00005<br>segundos        | 0,00006<br>segundos             |
| $n^2$                  | 0,0001<br>segundos               | 0,0004<br>segundos  | 0,0009<br>segundos  | 0,0016<br>segundos  | 0,0025<br>segundos         | 0,0036<br>segundos              |
| $n^3$                  | 0,001<br>segundos                | 0,008<br>segundos   | 0,027<br>segundos   | 0,064<br>segundos   | 0,125<br>segundos          | 0,216<br>segundos               |
| $n^5$                  | 0,1<br>segundos                  | 3,2<br>segundos     | 24,3<br>segundos    | 1,7<br>minutos      | 5,2<br>minutos             | 13,0<br>minutos                 |
| $2^n$                  | 0,001<br>segundos                | 1,0<br>segundos     | 17,9<br>segundos    | 12,7<br>dias        | 35,7<br>anos               | 366<br>séculos                  |
| $3^n$                  | 0,059<br>segundos                | 58<br>minutos       | 6,5<br>anos         | 3855<br>séculos     | $2 \times 10^8$<br>séculos | $1,3 \times 10^{13}$<br>séculos |



# Efeito de Progresso Tecnológico

| Maior instância que um computador resolve em 1 hora |                  |                             |                              |
|---|------------------|-----------------------------|------------------------------|
| Função de complexidade                              | Computador Atual | Computador 100x mais rápido | Computador 1000x mais rápido |
| $n$   | N                | 100 N                       | 1000 N                       |
| $n^2$   | M                | 10 M                        | 31,6 M                       |
| $n^3$   | Z                | 4,64 Z                      | 10 Z                         |
| $n^5$   | W                | 2,5 W                       | 3,98 W                       |
| $2^n$   | X                | X + 6,64                    | X + 9,97                     |
| $3^n$   | Y                | Y + 4,19                    | Y + 6,29                     |