Projeto e Análise de Algoritmos

Aula 4: Dividir para Conquistar ou Divisão e Conquista (2.1-2.2)

DECOM/UFOP
2012/12– 5º. Período
Anderson Almeida Ferreira
Material desenvolvido por Andréa Iabrudi Tavares

BCC241/2012-2

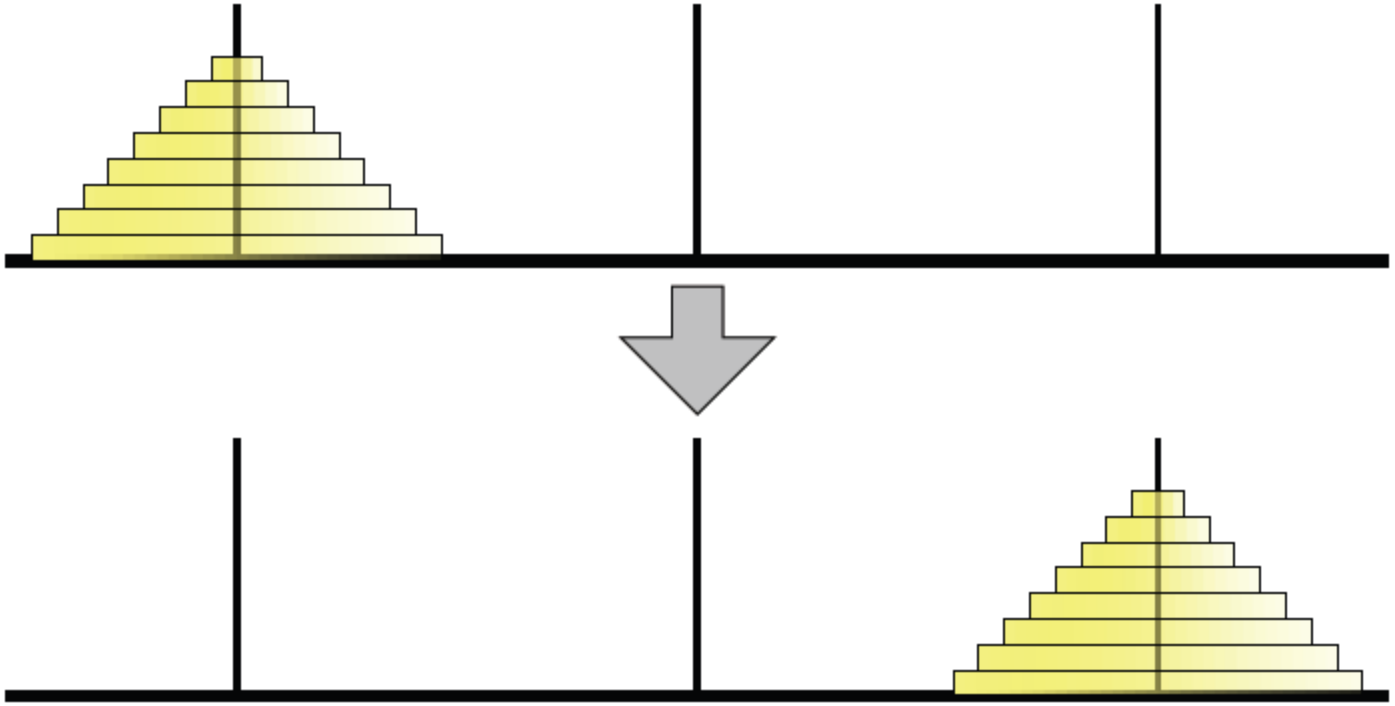
Estratégias para problemas tratáveis

- Estruturas de Dados
  - Use uma estrutura adequada
- Espaço por Tempo
  - Gaste mais espaço para economizar tempo
- Algoritmos Probabilísticos
  - Use aleatoriedade para conseguir eficiência
- Dividir para conquistar (top-down)
  - Divida em subproblemas semelhantes e disjuntos, resolva e combine
- Programação Dinâmica (bottom-up)
  - Comece com subproblemas e componha um maior, reusando solução de subproblemas compartilhados
- Algoritmos Gulosos
  - Sempre pegue o “melhor”

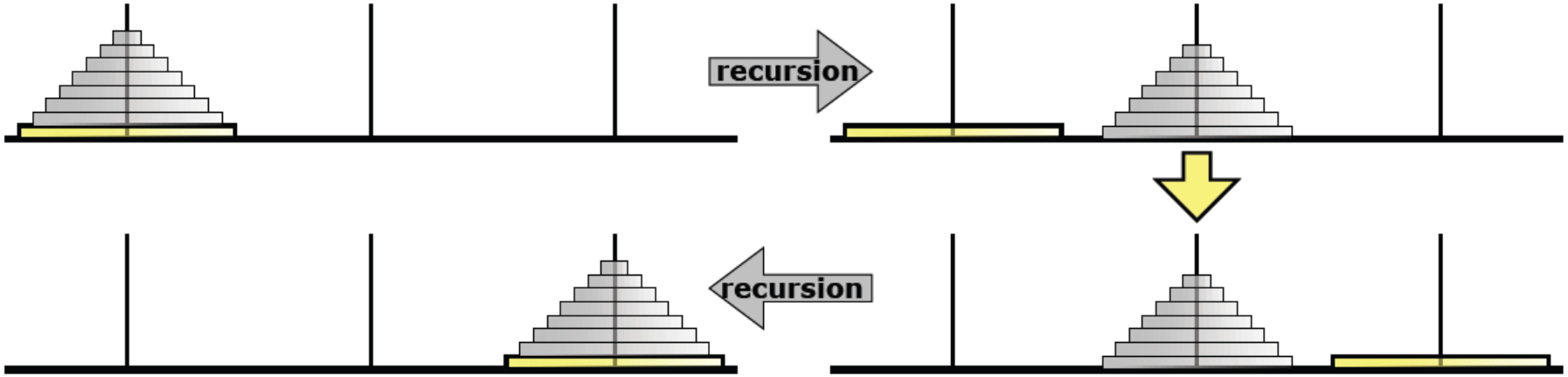
Ordenação:
Um problema, vários algoritmos

- Recursão: Inserção
- Dividir-para-conquistar: MergeSort e QuickSort
- Estrutura de dados: HeapSort
- Tempo por espaço: Counting Sort

Torre de Hanoi



Recursão: Diminuir para conquistar




```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)
```

Recursão: Equação de Recorrência

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)
```

Recursão: Análise

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)
```

Tempo de Execução

Dividir para Conquistar (*Divide and Conquer*)

- Problema (instância) pode ser dividido em subproblemas menores (parecidos e independentes) que são resolvidos recursivamente e combinados.
  - Análise por equação de recorrência

Dividir para Conquistar

- Diminuir complexidade (em geral de linear para logarítmico) de algoritmos polinomiais
  - Busca
  - Ordenação
  - Multiplicação
  - Exponenciação

- Apresentar melhor função de complexidade de pior caso
  - Mínimo/máximo

Busca Sequencial

```
function BuscaSequencial(A, x,e,d)
//Entrada: sub-vetor A[e..d]
//Saída: x pertence ao sub-vetor

1. i = e;
2. while (i≤d) & (A[i]≠x)
3.    i = i +1;
4. return (i ≤ d);
```

Lembre-se, nada sendo dito, utilizaremos sempre análise assintótica de pior caso!

Busca Sequencial com Vetor Ordenado

```
function BuscaSequencial2(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

1. i = e;
2. while (i≤d) & (A[i]≤x)
3.    i = i +1;
4. if (i ≤ d): return (A[i]==x);
```

Busca Sequencial Recursiva

```
function BuscaSequencialRec(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

1. if (e = d): return (A[e]= x);
2. if (A[e]=x):
3.     return true;
4. else:
5.   if (A[e]>x):
6.     return false;
7.   else:
8.     return BuscaSequencialRec(A,x,e+1,d);
```

D&C: Busca Binária

```
function BuscaBinária(A,x,e,d)
//Entrada: sub-vetor A[e..d] ordenado
//Saída: x pertence ao sub-vetor

1. if (e=d): return (A[e]=x);
2. m=(e+d)/2;
3. if (A[m]=x):
4.   return true;
5. else:
6.   if (A[m]>x):
7.     return BuscaBinária(A,x,e,m-1);
8.   else:
9.     return BuscaBinária(A,x,m+1,d);
```

D&C: Passos

- Divida
  - problema em subproblemas menores
- Conquiste
  - resolva recursivamente cada subproblema OU
  - limite de tamanho: use outro método
- Combine
  - soluções de subproblemas para resolver problema

D&C: Meta-algoritmo

```
function DividirConquistar(P)
//Entrada: problema P
//Saída: solução S para P

1. if (ÉPequeno(P)):
2.    S = ResolvePequeno(P);
3. else:
4.    Divida(P, P1, … , Pa);
5.    for i=1..a:
6.       Si = DividirConquistar(Pi);
7.    S = Combine(1,…,Sa);
8. return S;
```

Multiplicação de Inteiros Grandes

- x e y de n bits
- n multiplicações de *1*-n bit
- n somas de n-n bits

$$\begin{array}{rcccc} & 1 & 1 & 0 & 1 \\ \times & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Carry:	1			1	1	1		
		1	1	0	1	0	1	(53)
		1	0	0	0	1	1	(35)
	1	0	1	1	0	0	0	(88)

Custo: $\Theta(n^2)$

Multiplicação de Inteiros Grandes

- Podemos fazer melhor usando D&C?
  - Como dividir?

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R.$$

  - Como combinar?

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

  - Quanto custa?

Custo: $\Theta(n^2)$

D&C para Multiplicação

- Gauss em no século XVIII e Karatsuba em 1962:

$$bc + ad = (a+b)(c+d) - ac - bd.$$

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

D&C Multiplicação: algoritmo

```
function multiply(x, y)
Input:  Positive integers x and y, in binary
Output:  Their product

n = max(size of x, size of y)
if n = 1:  return xy

x_L, x_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of x
y_L, y_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of y

P_1 = multiply(x_L, y_L)
P_2 = multiply(x_R, y_R)
P_3 = multiply(x_L + x_R, y_L + y_R)

return P_1 × 2^n + (P_3 − P_1 − P_2) × 2^(n/2) + P_2
```

Parada recursão

Divide

Conquiste: Recursão

Combine

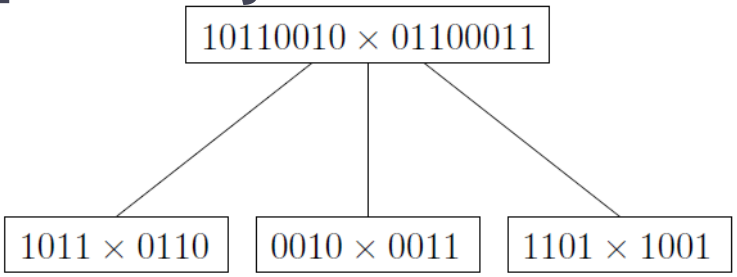
D&C Multiplicação: Complexidade

```
function multiply(x, y)
Input:  Positive integers x and y, in binary
Output:  Their product

n = max(size of x, size of y)
if n = 1:  return xy

x_L, x_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of x
y_L, y_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of y

P_1 = multiply(x_L, y_L)
P_2 = multiply(x_R, y_R)
P_3 = multiply(x_L + x_R, y_L + y_R)
return P_1 × 2^n + (P_3 − P_1 − P_2) × 2^(n/2) + P_2
```

$$T(n) = \Theta\left(n^{\log_2 3}\right) = \Theta\left(n^{1.585}\right)$$

D&C Multiplicação - Análise

(a)



(b)