Algoritimos e Estruturas de Dados III

CIC210

Divisão e Conquista

Haroldo Gambini Santos

Concurso Universidade Federal de Ouro Preto - UFOP

3 de setembro de 2009

Conteúdo

Seção

Técnica de Divisão e Conquista

Passos Básicos

Dividir

Dividir o problema em uma ou mais subproblemas

Técnica de Divisão e Conquista

Passos Básicos

Dividir

Dividir o problema em uma ou mais subproblemas

Conquistar

Resolver os subproblemas caso esses sejam suficientemente pequenos, caso contrário continuar a divisão

Técnica de Divisão e Conquista

Passos Básicos

Dividir

Dividir o problema em uma ou mais subproblemas

Conquistar

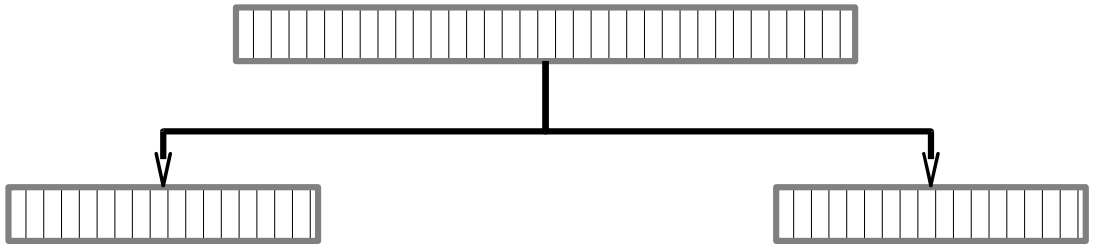
Resolver os subproblemas caso esses sejam suficientemente pequenos, caso contrário continuar a divisão

Combinar

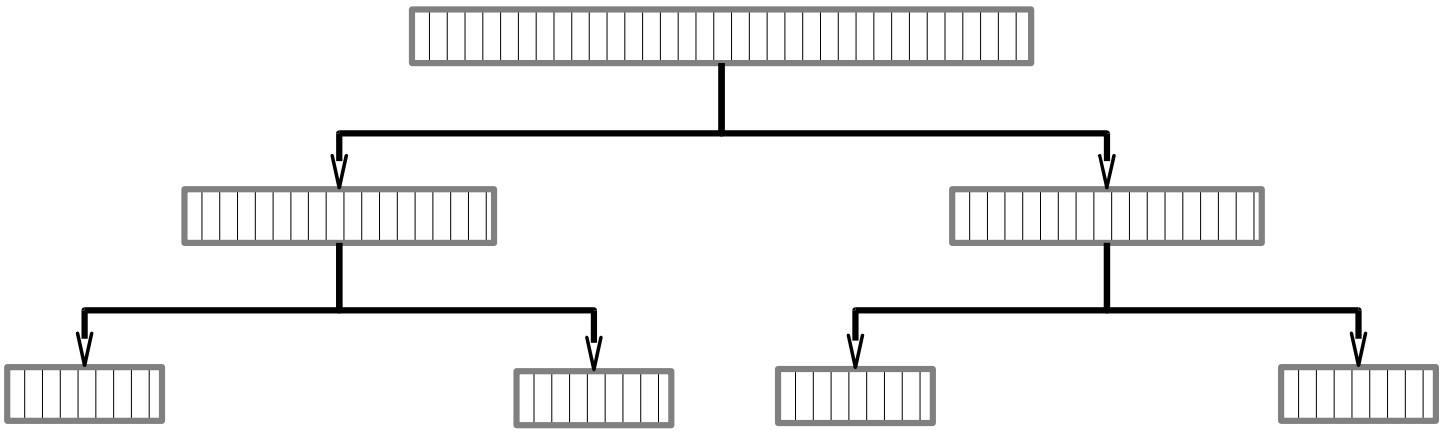
Soluções dos subproblemas para a solução do problema original

Divisão e Conquista

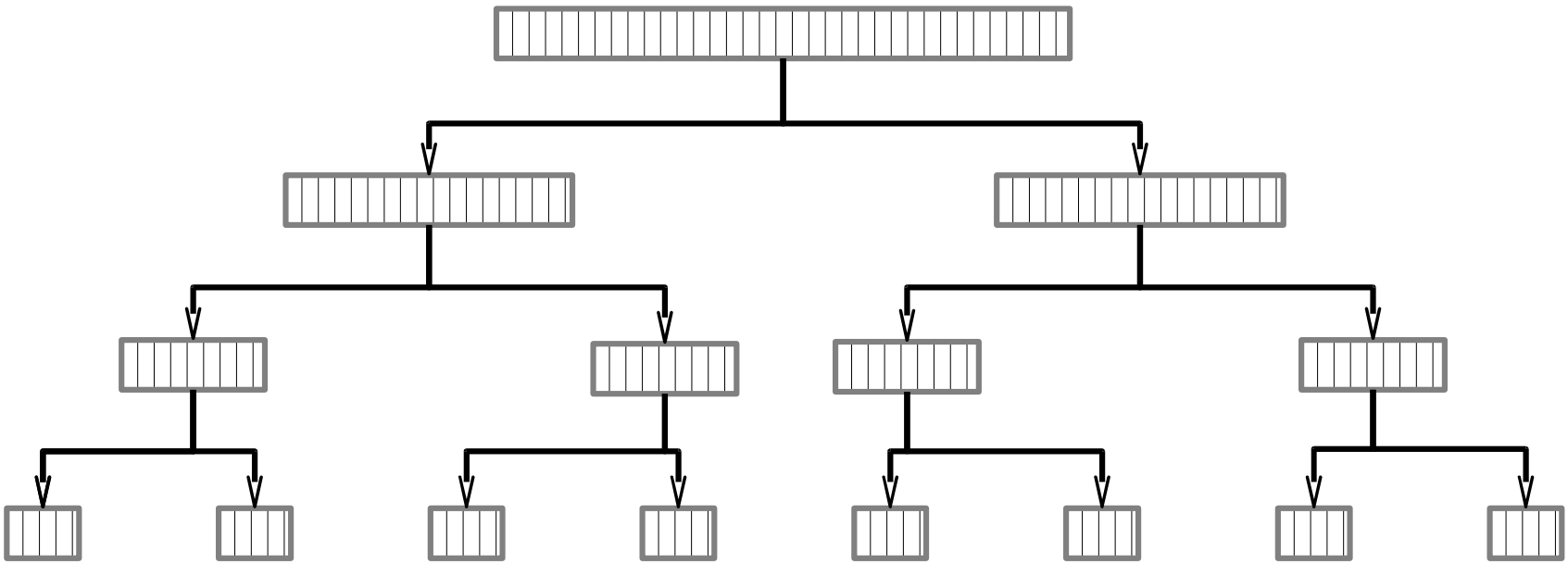
Divisão e Conquista



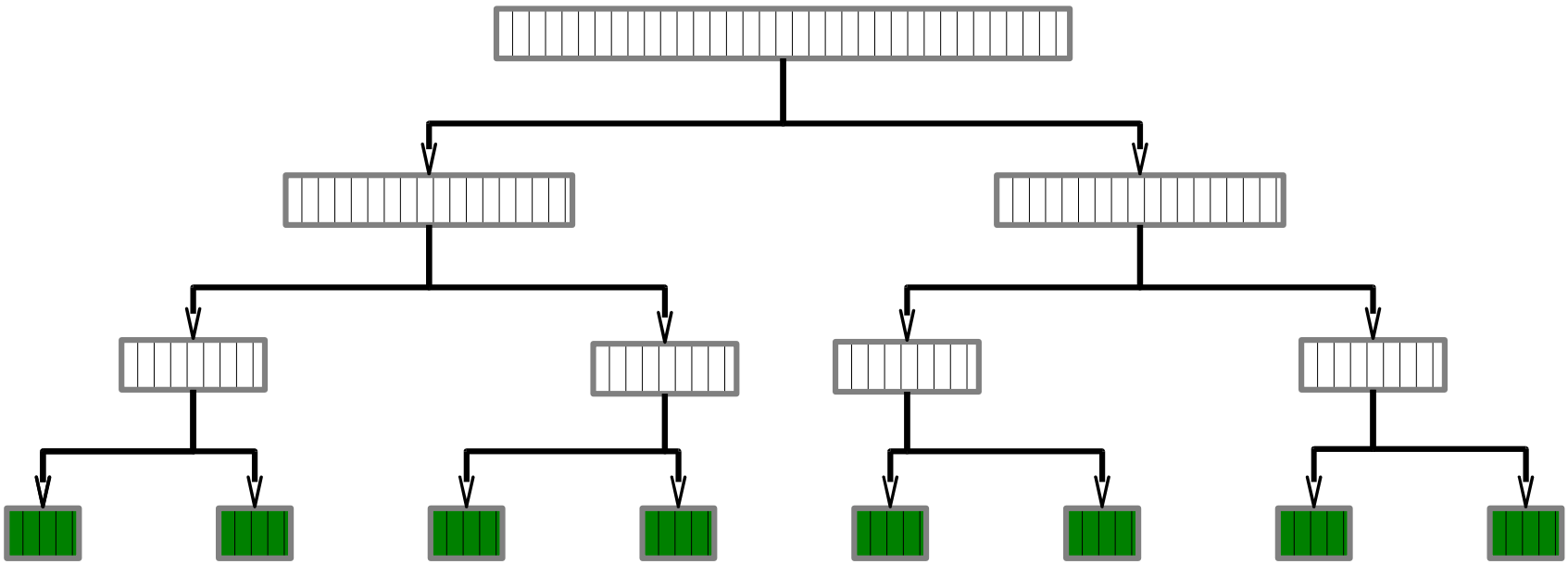
Divisão e Conquista



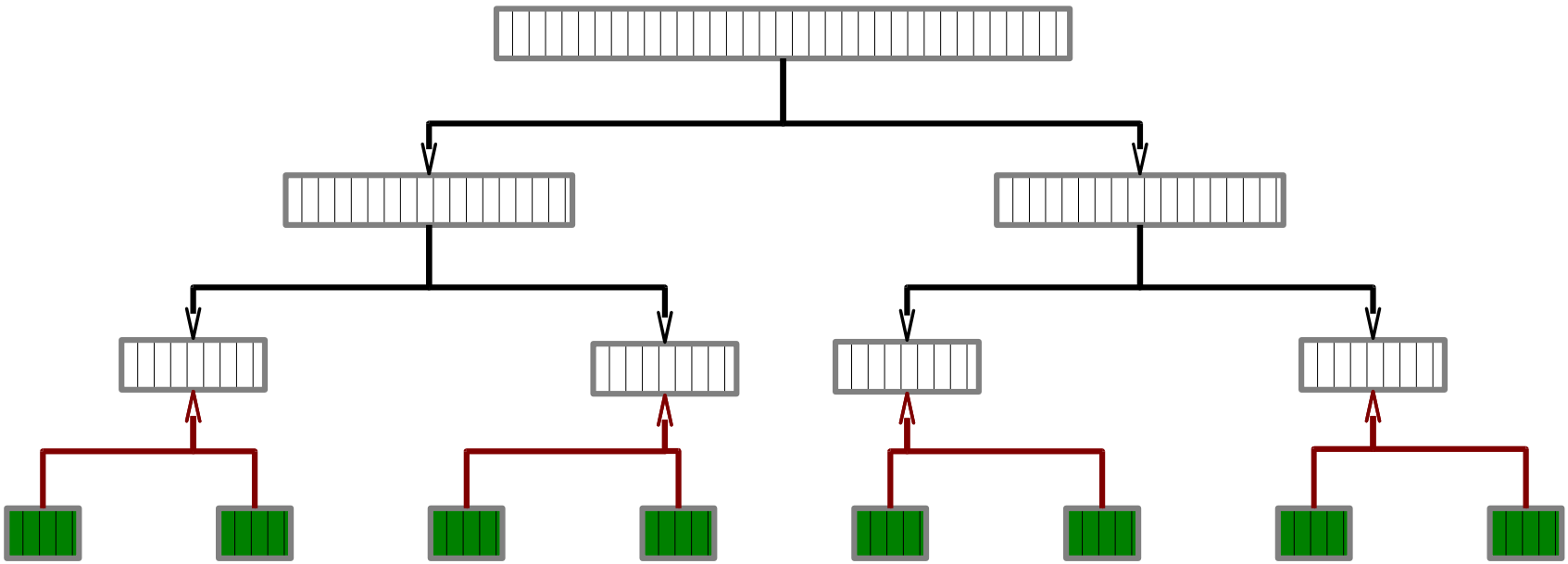
Divisão e Conquista



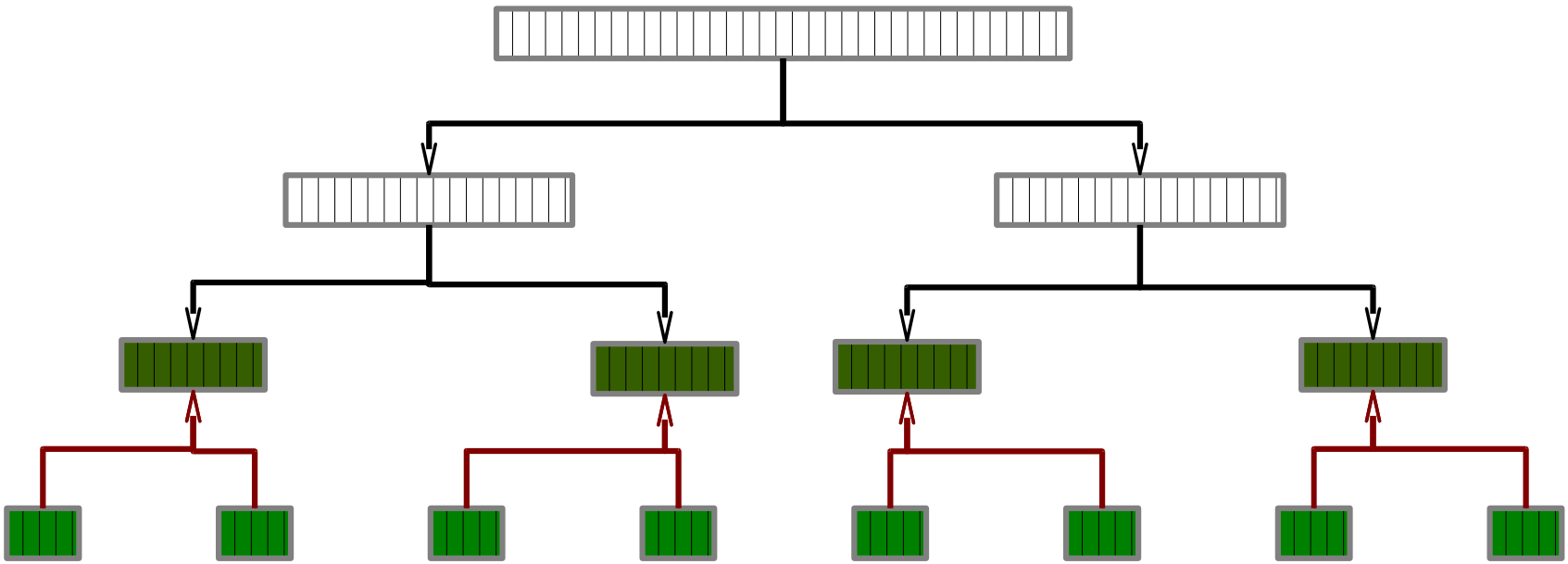
Divisão e Conquista



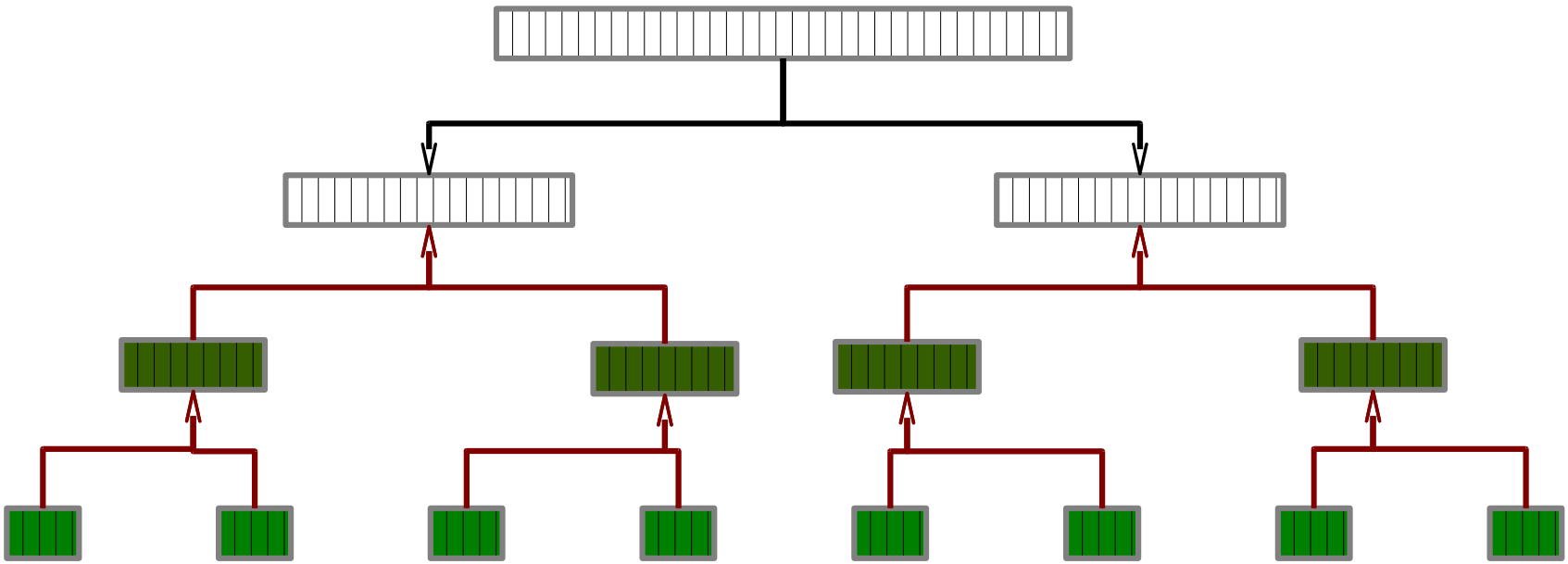
Divisão e Conquista



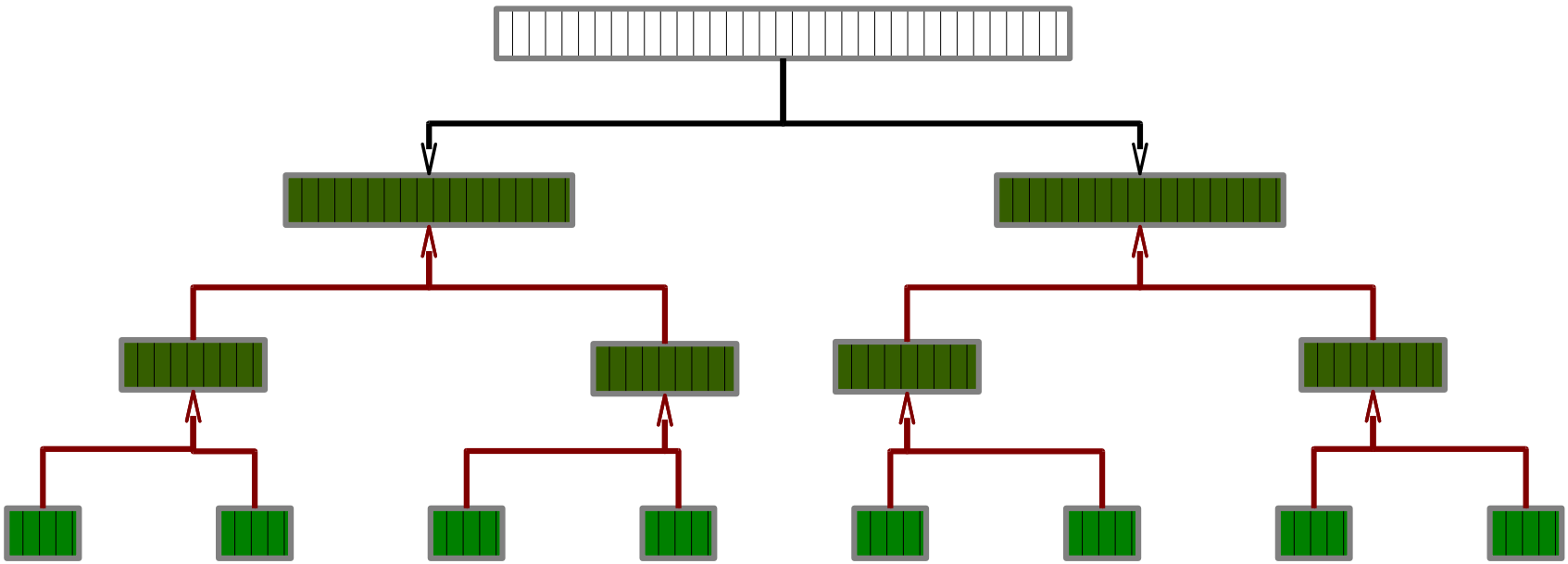
Divisão e Conquista



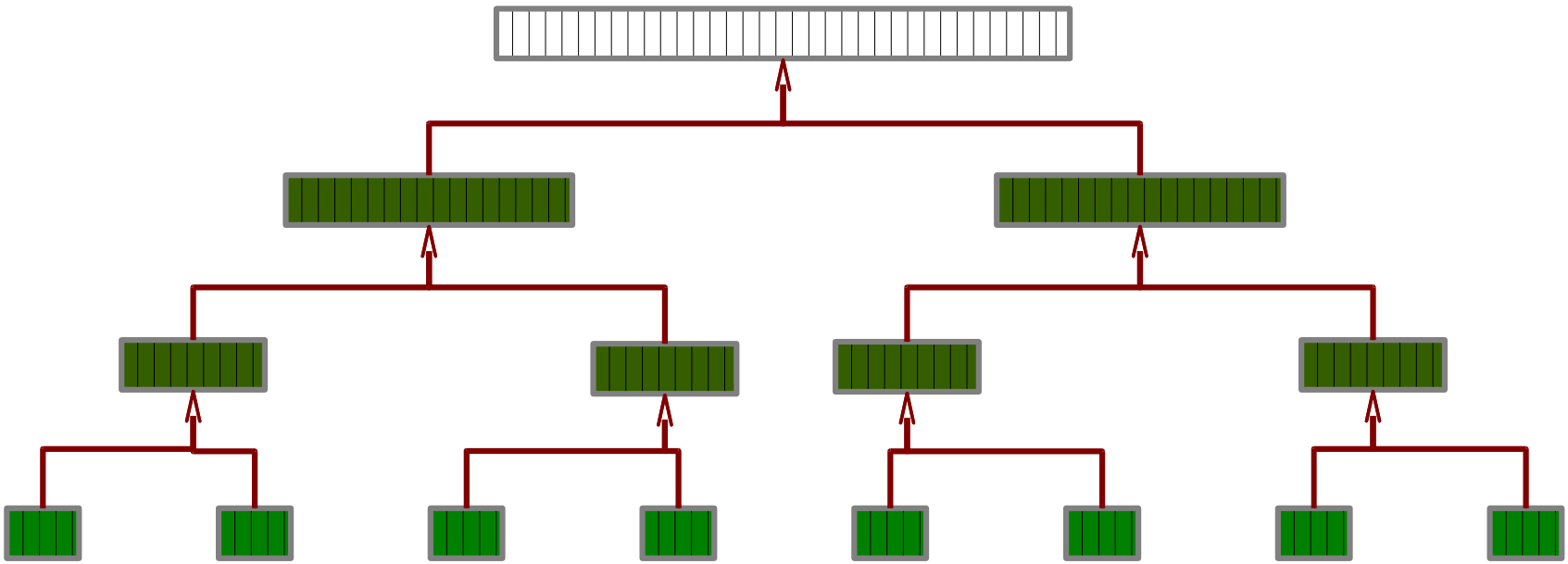
Divisão e Conquista



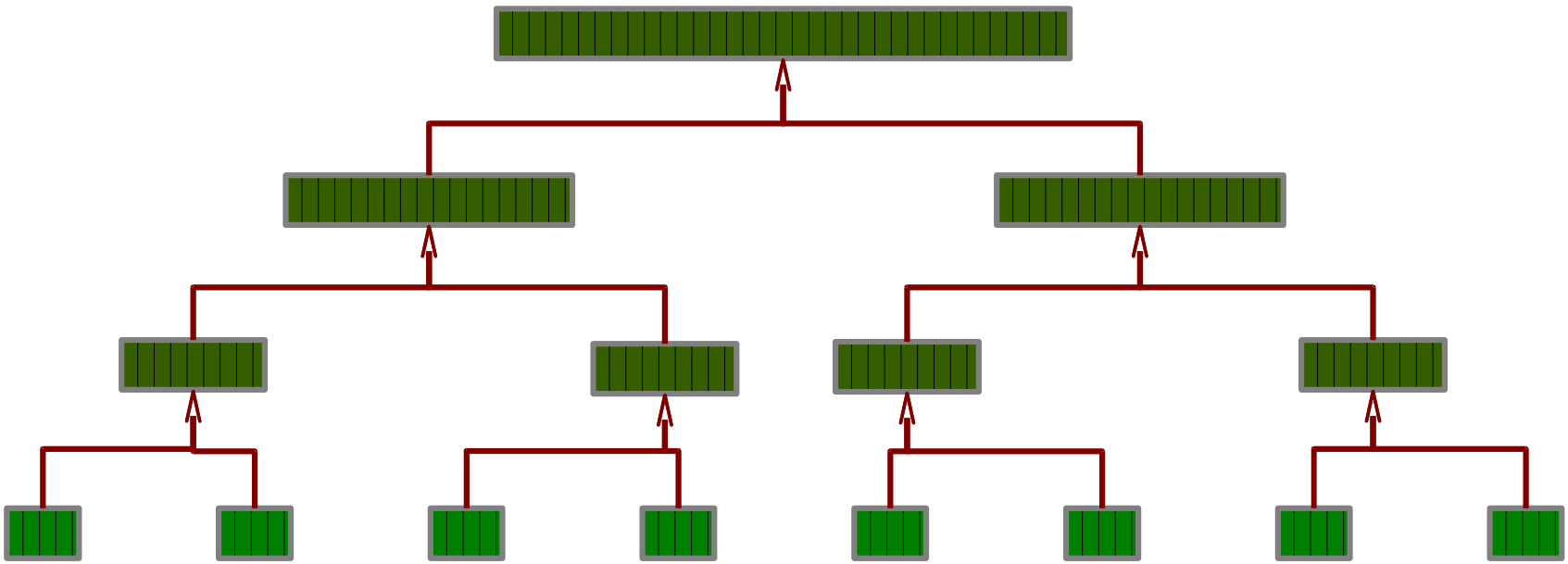
Divisão e Conquista



Divisão e Conquista



Divisão e Conquista



Divisão e Conquista

```
1  função divConq( x )
2  se  x é suficientemente pequeno então
3      retorna resolver(x) ;
4  senão
5      decomponha x em k conjuntos menores: x_1, x_2, ..., x_k;
6      para  i=1 até k faça
7          y_i = divConq( x_i );
8      fim
9      combine o resultados de y_1, y_2, ..., y_k;
10     retorne y;
11 fim
```

Divisão e Conquista

Vantagens

- Eficiência do algoritmo

Divisão e Conquista

Vantagens

- Eficiência do algoritmo
- Tratamento da complexidade

Divisão e Conquista

Vantagens

- Eficiência do algoritmo
- Tratamento da complexidade
- Paralelismo

Divisão e Conquista

Vantagens

- Eficiência do algoritmo
- Tratamento da complexidade
- Paralelismo
- Acesso à memória

Seção

Subseção

Multiplicação de Inteiros

Entrada

Entrada: 2 inteiros, x e y representados em n bits.

Algoritmo trivial

$$\Theta(n^2)$$

Multiplicação de Inteiros

Entrada

Entrada: 2 inteiros, x e y representados em n bits.

Algoritmo trivial

$$\Theta(n^2)$$

Pode-se melhorar ?

Multiplicação de Inteiros

Entrada

Entrada: 2 inteiros, x e y representados em n bits.

Algoritmo trivial

$$\Theta(n^2)$$

Pode-se melhorar ?

SIM, ao menos comportamento assintotico.

Multiplicação de Inteiros

Passo 1

Dividir os números x e y em duas partes:

$$x = 2^{\frac{n}{2}}a + b$$

$$y = 2^{\frac{n}{2}}c + d$$

Multiplicação de Inteiros

Passo 1

Dividir os números x e y em duas partes:

$$x = 2^{\frac{n}{2}}a + b$$

$$y = 2^{\frac{n}{2}}c + d$$

Obtemos

$$xy = 2^n ac + 2^{\frac{n}{2}}(ad + bc) + bd$$

Multiplicação de Inteiros

Continuando

$$xy = 2^n ac + 2^{\frac{n}{2}}(ad + bc) + bd$$

4 multiplicações

Multiplicação de Inteiros

Continuando

$$xy = 2^n ac + 2^{\frac{n}{2}}(ad + bc) + bd$$

4 multiplicações

Alternativa

$$\begin{aligned} x_1 &= multiplica(a, c) \\ x_2 &= multiplica(b, d) \\ x_3 &= multiplica(a + b, c + d) - x_1 - x_2 \end{aligned}$$

Multiplicação de Inteiros

Alternativa

$$\begin{aligned} x_1 &= multiplica(a, c) \\ x_2 &= multiplica(b, d) \\ x_3 &= multiplica(a + b, c + d) - x_1 - x_2 \end{aligned}$$

Recorrência

$$T(n) = 3T(\frac{n}{2}) + \Theta(n)$$

$$\Theta(n^{1.59})$$

Multiplicação de Inteiros

1 **função** `multiplicaInteiros`(x, y, n)
2 // x e y são vetores binários de tamanho n
Saída: vetor binário z com o resultado da multiplicação
3 **se** $n == 1$ **então**
4 retorne xy;
5 **fim**
6 $x_L = \lceil \frac{n}{2} \rceil$ bits à esquerda de x;
7 $x_R = \lfloor \frac{n}{2} \rfloor$ bits à direita de x;
8 $y_L = \lceil \frac{n}{2} \rceil$ bits à esquerda de y;
9 $y_R = \lfloor \frac{n}{2} \rfloor$ bits à direita de y;
10 $P_1 =$ `multiplicaInteiros`$(x_L, y_L, \lceil \frac{n}{2} \rceil)$;
11 $P_2 =$ `multiplicaInteiros`$(x_R, y_R, \lfloor \frac{n}{2} \rfloor)$;
12 $P_3 =$ `multiplicaInteiros`$(x_L + y_R, x_L + y_R, \lceil \frac{n}{2} \rceil)$;
13 retorne $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{\frac{n}{2}} + P_2$

Subseção

Busca Binária

```
1  função buscaBin( V[], n, chave )
2  // V deve ser um vetor ordenado com posições 0, ..., n − 1
   Saída: posição de V com valor igual a chave ou -1
3  l = 0; r = n − 1;
4  enquanto l ≤ r faça
5      m = ⌊(l+r)/2⌋;
6      se V[m] == chave então
7          retorne m;
8      senão
9          se chave < V[m] então
10             r = m - 1;
11         senão
12             l = m + 1;
13         fim
14     fim
15 fim
16 retorne −1;
```

Busca Binária

Exemplo

```
chave 31
V[i] =  3  14  27  31  39  42  55  70  74  81  85  93  98
i       0   1   2   3   4   5   6   7   8   9  10  11  12

it. 1   l                       m                       r

it. 2   l       m               r

it. 3               l   m   r

it. 4               l
                    m
                    r
```

Busca Binária

Comentários

- número de possíveis índices: $r - l + 1$
  - reduz-se pela metade (ou mais) a cada iteração
- nr. de vezes que n é divido por 2 até que $n < 1$
  - $\lg n$

Subseção

Quicksort

```
1 função quickSort( V[l,...,r] )
2 se l < r então
3       p = partition( V[l,...,r] );
4       quickSort(V[l,...,p-1]);
5       quickSort(V[p+1,...,r]);
6 fim
```

Quicksort

```
1  função partition( V[l, ..., r] )
2  pivot = V[l];
3  i = l;
4  j = r + 1;
5  repita
6      repita i = i + 1 até que V[i] ≥ pivot;
7      repita j = j − 1 até que V[j] ≤ pivot;
8      se i < j então
9          troque V[i] e V[j]
10     fim
11 até  i ≥ j ;
12 troque V[l] e V[j];
13 retorne j;
```

Quicksort

Exemplo

5 3 1 9 8 2 7 4

2 3 1 4 *5 8 7 9

1 *2 3 4 7 *8 8

3 4

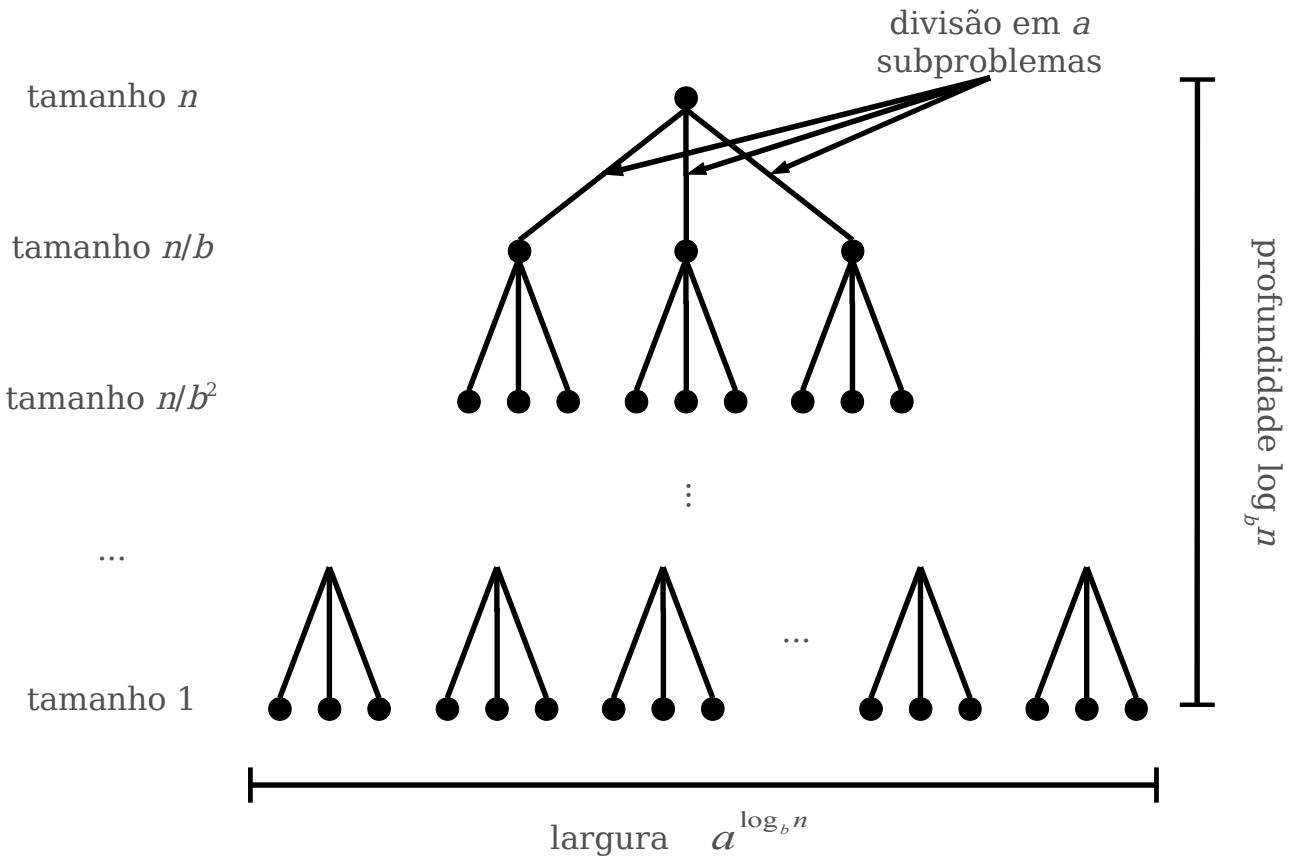
Seção

Resolvendo Recorrências

Padrão Genérico

- Problema é sucessivamente:
  - Dividido em a subproblemas
  - Cada subproblema de tamanho $\frac{n}{b}$

Resolvendo Recorrências



Subseção

O Método Mestre

Utilidade

- Caracterizar assintoticamente um algoritmo recursivo
- Permite resolver recorrências que aparecem em algoritmos do tipo D e C

Formato

$$T(n) = aT(\frac{n}{b}) + O(n^d)$$

- a: número de subproblemas (> 1)
- $\frac{n}{b}$: tamanho de cada subproblema (b > 1)
- custo de dividir o problema e de se combinar os resultados $(d \geq 0)$

O Método Mestre

Utilidade

- Caracterizar assintoticamente um algoritmo recursivo
- Permite resolver recorrências que aparecem em algoritmos do tipo D e C

Formato

$$T(n) = aT(\frac{n}{b}) + O(n^d)$$

- a: número de subproblemas (> 1)
- $\frac{n}{b}$: tamanho de cada subproblema (b > 1)
- custo de dividir o problema e de se combinar os resultados ($d \geq 0$)

O Método Mestre

Utilidade

- Caracterizar assintoticamente um algoritmo recursivo
- Permite resolver recorrências que aparecem em algoritmos do tipo D e C

Formato

$$T(n) = aT(\frac{n}{b}) + O(n^d)$$

- a: número de subproblemas (> 1)
- $\frac{n}{b}$: tamanho de cada subproblema (b > 1)
- custo de dividir o problema e de se combinar os resultados ($d \geq 0$)

O Método Mestre

Utilidade

- Caracterizar assintoticamente um algoritmo recursivo
- Permite resolver recorrências que aparecem em algoritmos do tipo D e C

Formato

$$T(n) = aT(\frac{n}{b}) + O(n^d)$$

- a: número de subproblemas (> 1)
- $\frac{n}{b}$: tamanho de cada subproblema (b > 1)
- custo de dividir o problema e de se combinar os resultados ($d \geq 0$)

O Método Mestre

$T(n)$

$$T(n) = aT(\frac{n}{b}) + O(n^d)$$

Limitação assintotica de $T(n)$

$$T(n) = \begin{cases} O(n^d) & \text{se } d > \log_b a \\ O(n^d \log n) & \text{se } d = \log_b a \\ O(n^{\log_b a}) & \text{se } d < \log_b a \end{cases}$$