



MergeSort

Algoritmos e Programação II
(slides baseados na apostila do Prof Fábio Viduani)

Introdução e motivação

- 0 Operação básica em Computação
- 0 Será que existem métodos mais eficientes de ordenação?
- 0 Ordenação por recursão
- 0 Dividir para conquistar

Dividir para conquistar

| | |
|-------------------|---|
| Dividir | o problema em um número de subproblemas; |
| Conquistar | os subproblemas solucionando-os recursivamente. No entanto, se os tamanhos dos subproblemas são suficientemente pequenos, resolva os subproblemas de uma maneira simples; |
| Combinar | Combinar as soluções dos subproblemas na solução do problema original. |

Problema da intercalação

0 Problema:

0 Dados dois conjuntos crescentes A e B , com m e n elementos, respectivamente, obter um conjunto crescente C a partir de A e B .

0 Problema:

0 Dados dois vetores crescentes $v[p .. q-1]$ e $v[q .. r-1]$, rearranjar $v[p .. r-1]$ em ordem crescente.

Vetor A

p = 0

q = 4

r = 8

| | | | |
|----|----|----|----|
| 18 | 34 | 56 | 78 |
|----|----|----|----|

0

1

2

3

| | | | |
|---|----|----|----|
| 7 | 15 | 22 | 60 |
|---|----|----|----|

4

5

6

7

Vetor w

| | | | | | | | |
|--|--|--|--|--|--|--|--|
| | | | | | | | |
|--|--|--|--|--|--|--|--|

0

1

2

3

4

5

6

7

```

/* Recebe os vetores crescentes v[p..q-1] e v[q..r-1]
   e rearranja v[p..r-1] em ordem crescente */
void intercala(int p, int q, int r, int v[MAX])
{
    int i, j, k, w[MAX];

    i = p; j = q; k = 0;
    while (i < q && j < r) {
        if (v[i] < v[j]) {
            w[k] = v[i]; i++; }
        else {
            w[k] = v[j]; j++; }
        k++;
    }
    while (i < q) {
        w[k] = v[i]; i++; k++; }
    while (j < r) {
        w[k] = v[j]; j++; k++; }
    for (i = p; i < r; i++)
        v[i] = w[i-p];
}

```

Vetor v

$p = 4$

| | |
|---|----|
| 7 | 22 |
|---|----|

4

5

$q = 6$

| | |
|----|----|
| 15 | 60 |
|----|----|

6

7

$r = 8$

Vetor w

| | | | |
|---|----|----|----|
| 7 | 15 | 22 | 60 |
|---|----|----|----|

0

1

2

3

Vetor v

| | | | |
|--|--|--|--|
| | | | |
|--|--|--|--|

4

5

6

7

TEMPO DE EXECUÇÃO - INTERCALA

Vetor v

$p = 0$

$q = 4$

$r = 8$

| | | | | | | | |
|----|----|----|----|---|----|----|----|
| 18 | 34 | 56 | 78 | 7 | 15 | 22 | 60 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Vetor w

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Problema da intercalação

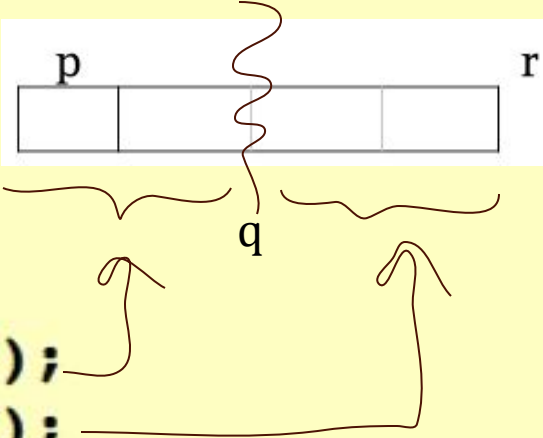
- 0 Tempo de execução de pior caso proporcional ao número de comparações entre os elementos do vetor, isto é, $r-p$.
- 0 Isto é, o consumo de tempo no pior caso da função **intercala** é proporcional ao número de elementos do vetor de entrada.

Ordenação por intercalação

- 0 Divida ao meio um vetor v com $r-p$ elementos.
- 0 Ordenamos recursivamente essas duas metades de v .
- 0 Intercalamos essas metades

Ordenação por Intercalação

```
void mergesort(int p, int r, int v[MAX])  
{  
    int q;  
  
    if (p < r - 1) {  
        q = (p + r) / 2;  
        mergesort(p, q, v);  
        mergesort(q, r, v);  
        intercala(p, q, r, v);  
    }  
}
```



Para ordenar um vetor $v[0..n - 1]$ basta chamar a função **mergesort** com os seguintes argumentos:

mergesort(0, n, v);

entrada

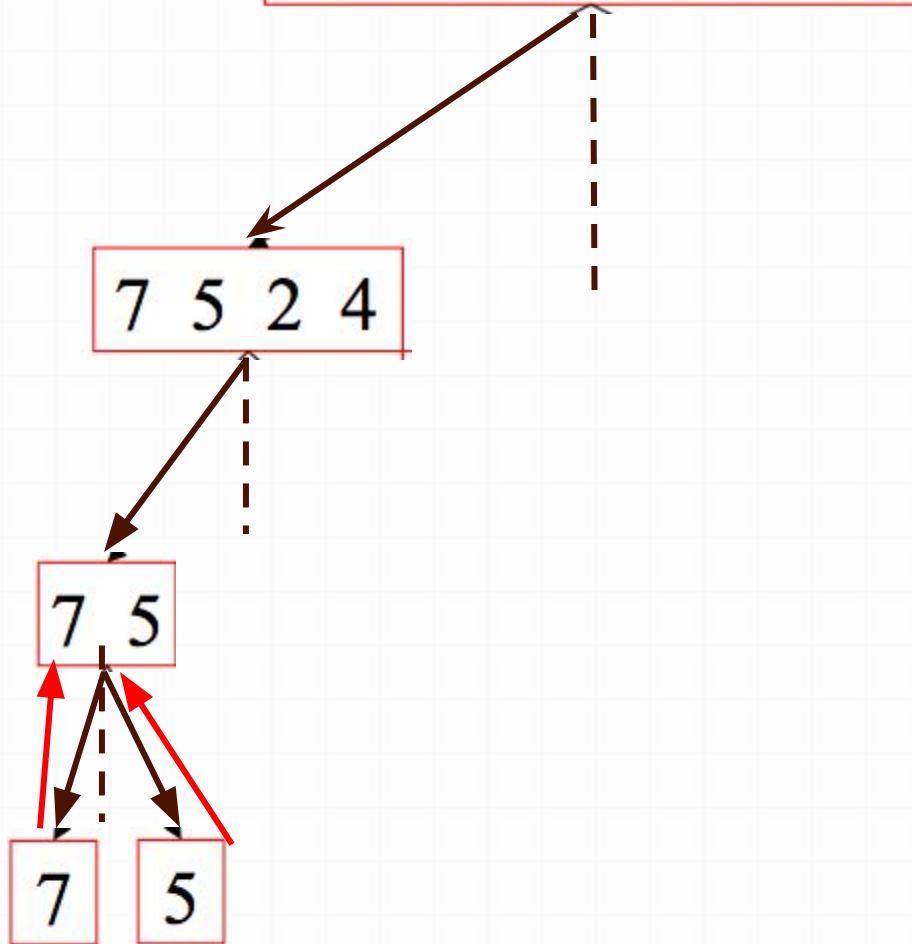
7 5 2 4 1 6 3 0

7 5 2 4

7 5

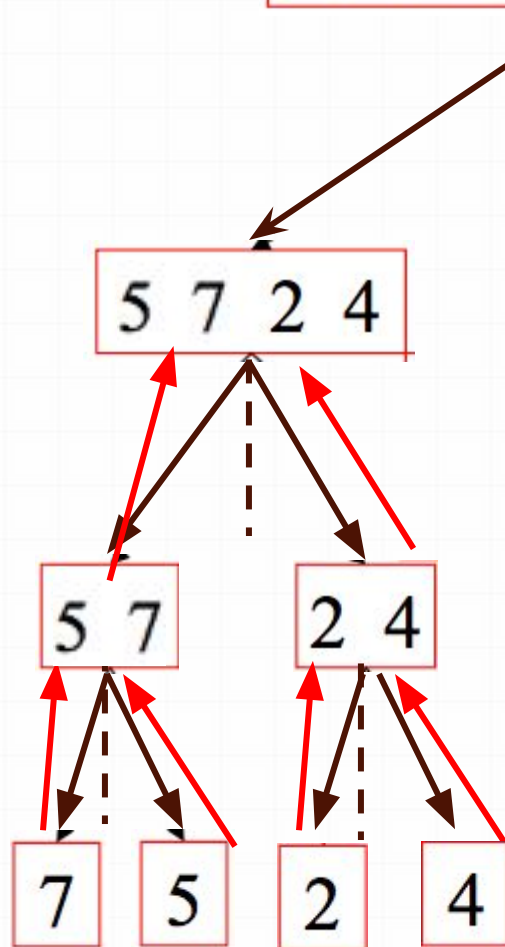
7

5

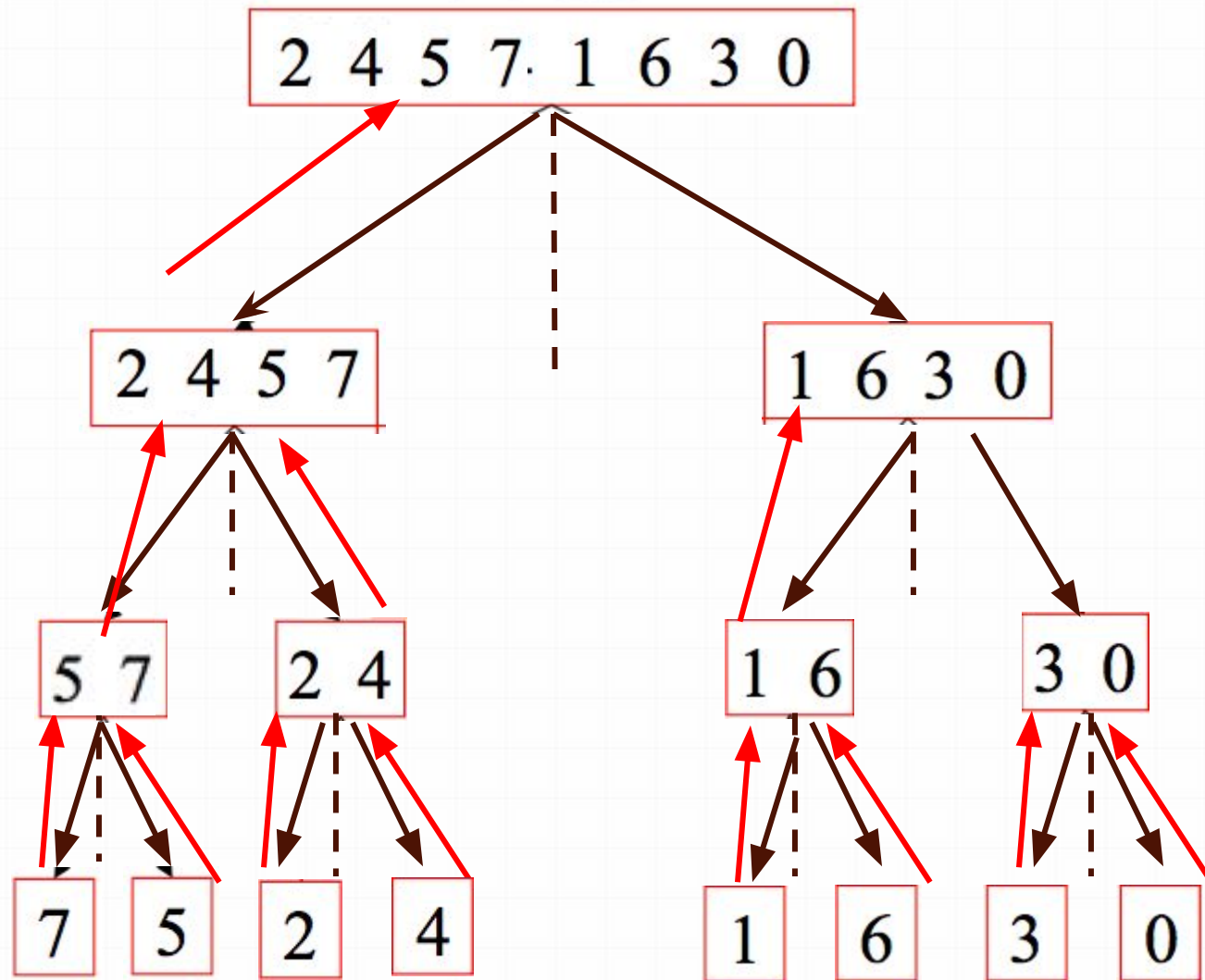


entrada

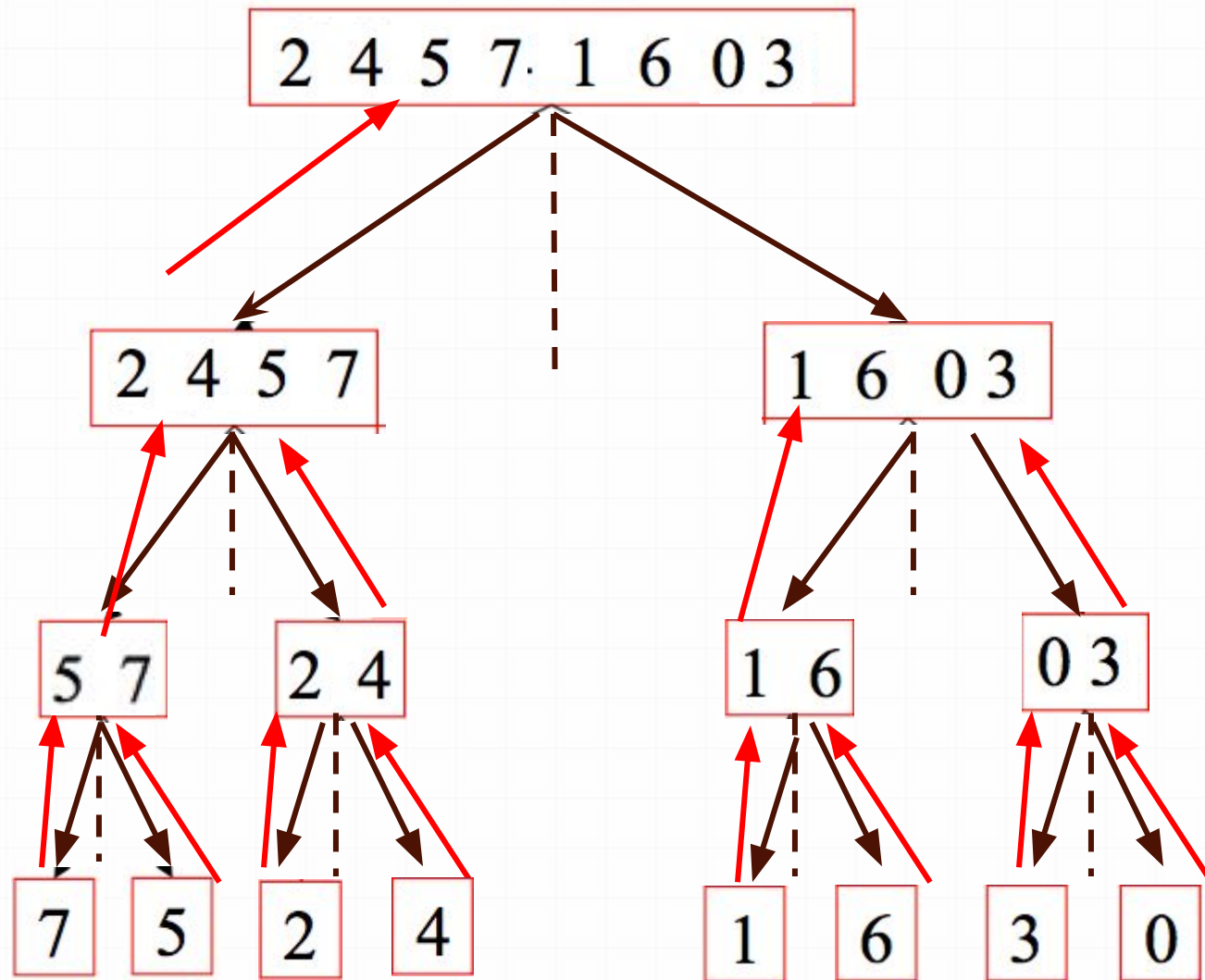
5 7 2 4 1 6 3 0



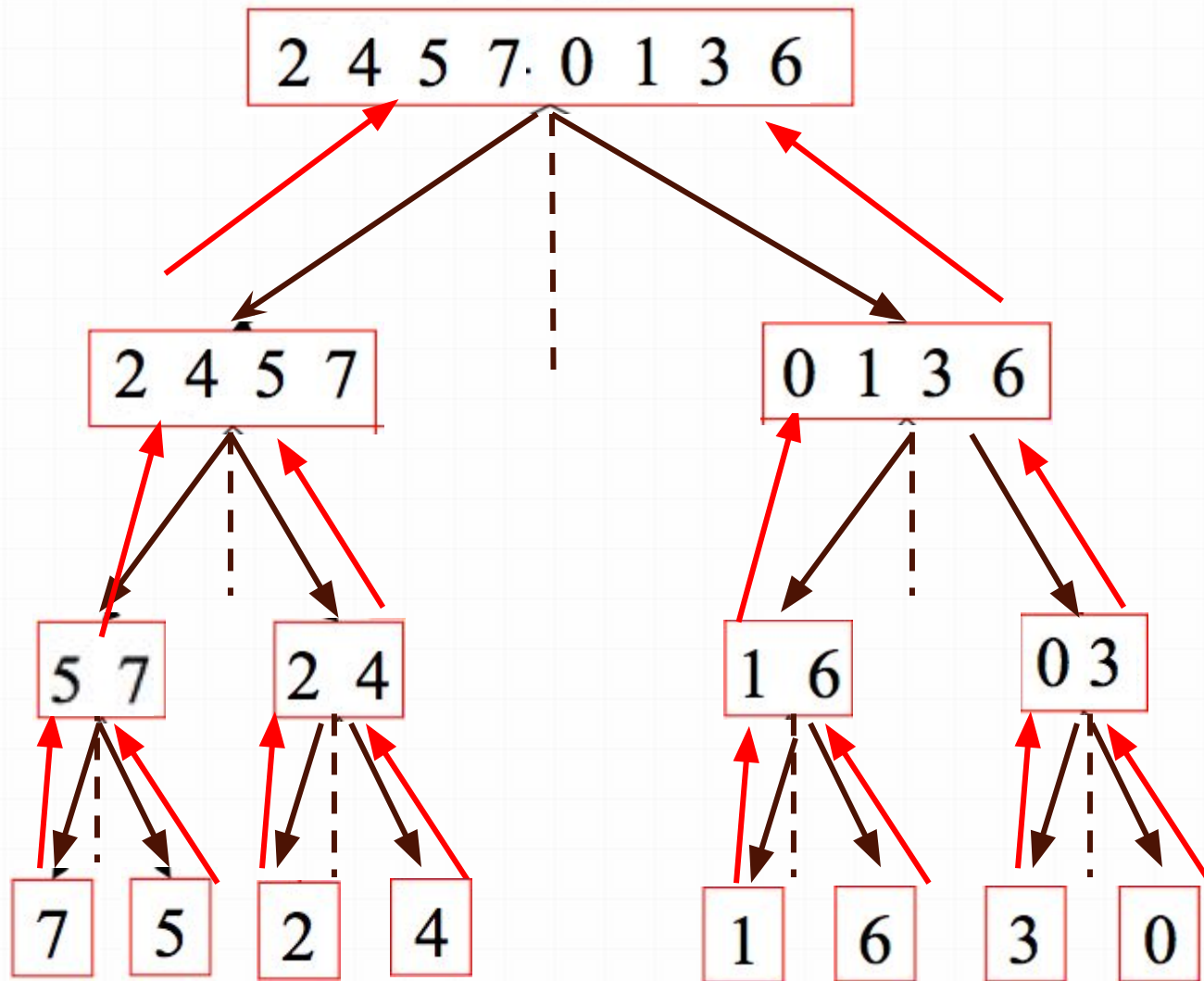
entrada



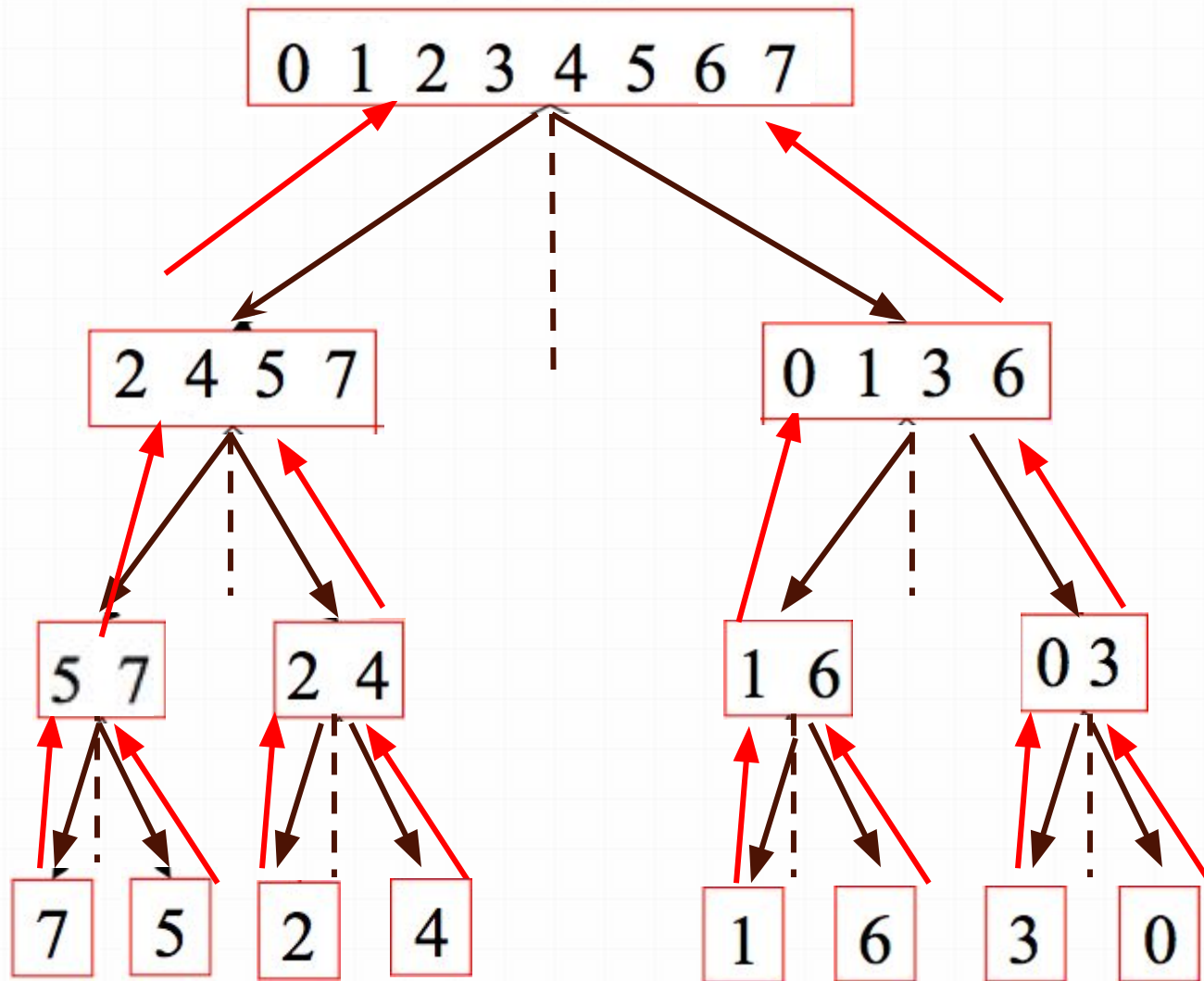
entrada



entrada

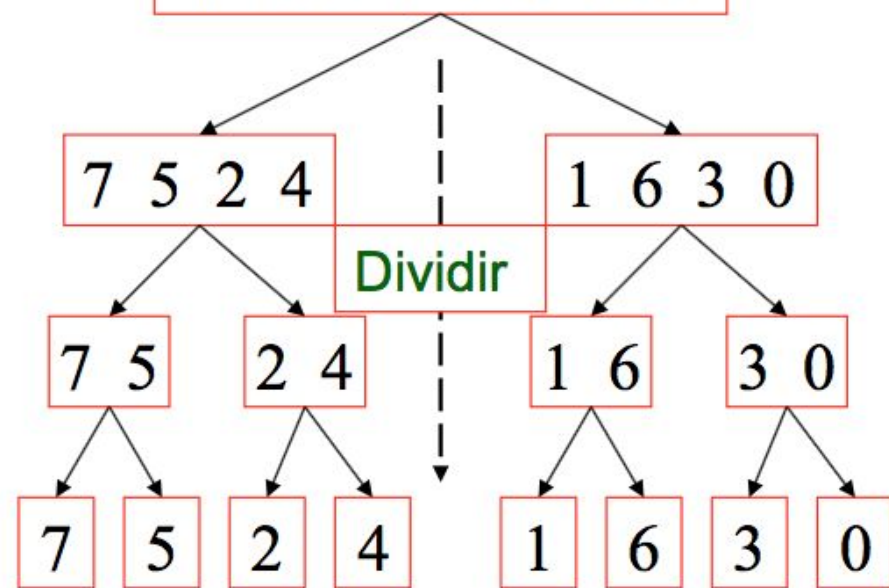


entrada



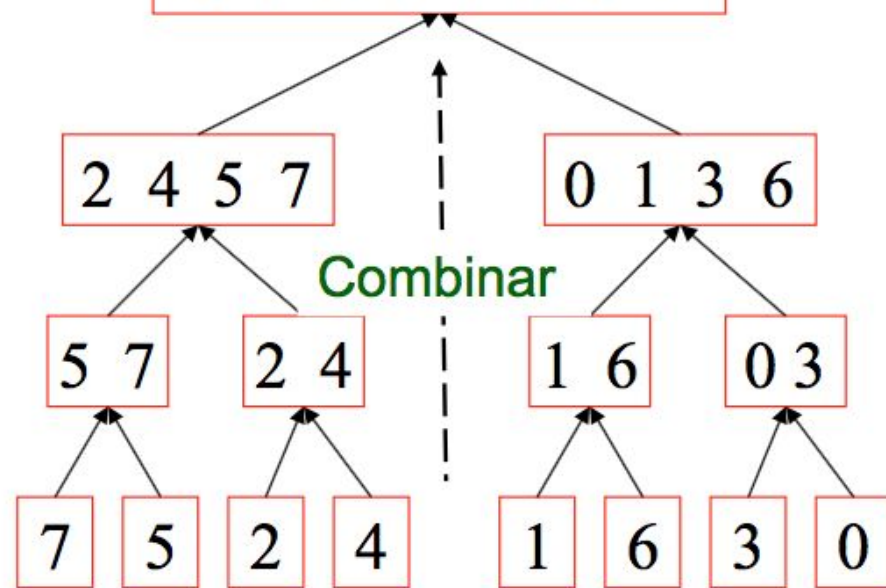
entrada

7 5 2 4 1 6 3 0



saída

0 1 2 3 4 5 6 7



Ordenação por intercalação

- 0 Desempenho da função **mergesort** quando queremos ordenar um vetor $v[0..n-1]$
 - 0 Suponha que n é uma potência de 2
 - 0 O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
 - 0 Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
 - 0 Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..n/2-1]$ e $v[n/2..n-1]$
 - 0 Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..n/4-1]$, $v[n/4..n/2-1]$, $v[n/2..3n/4-1]$ e $v[3n/4..n-1]$
 - 0 E assim por diante
- Além disso, o tempo total que a função **intercala** gasta é proporcional ao número de elementos do vetor v , isto é, $r-p$. Portanto, a função **mergesort** consome tempo proporcional a **$n \log_2 n$** .

Tempo de Execução - Mergesort

```
void mergesort(int p, int r, int v[MAX])
{
    int q;

    if (p < r - 1) {
        q = (p + r) / 2;
        mergesort(p, q, v);      → T(n/2)
        mergesort(q, r, v);      → T(n/2)
        intercala(p, q, r, v);   → n
    }
}
```

RECORRÊNCIA: $T(n) = T(n/2) + T(n/2) + n = 2T(n/2) + n$

$$T(n) = 2T(n/2) + n$$

Tempo de Execução - Mergesort

```
void mergesort(int p, int r, int v[MAX])
{
    int q;

    if (p < r - 1) {
        q = (p + r) / 2;
        mergesort(p, q, v);      → T(n/2)
        mergesort(q, r, v);      → T(n/2)
        intercala(p, q, r, v);   → n
    }
}
```

RECORRÊNCIA:

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2T(n/2) + n, & \text{se } n > 1 \end{cases}$$

Tempo de Execução - Mergesort

```
void mergesort(int p, int r, int v[MAX])
{
    int q;

    if (p < r - 1) {
        q = (p + r) / 2;
        mergesort(p, q, v);      → T(n/2)
        mergesort(q, r, v);      → T(n/2)
        intercala(p, q, r, v);   → n
    }
}
```

RECORRÊNCIA:

$$\begin{cases} T(1) = 1 \\ T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \quad \text{para } n \geq 2. \end{cases}$$

Tempo de Execução - Mergesort

- ÁRVORE
- ÁRVORE DE RECURSÃO
- ALTURA DA ÁRVORE - $\log N$

Tempo de Execução - Mergesort

- $T(n) = 2 T(n/2) + n$