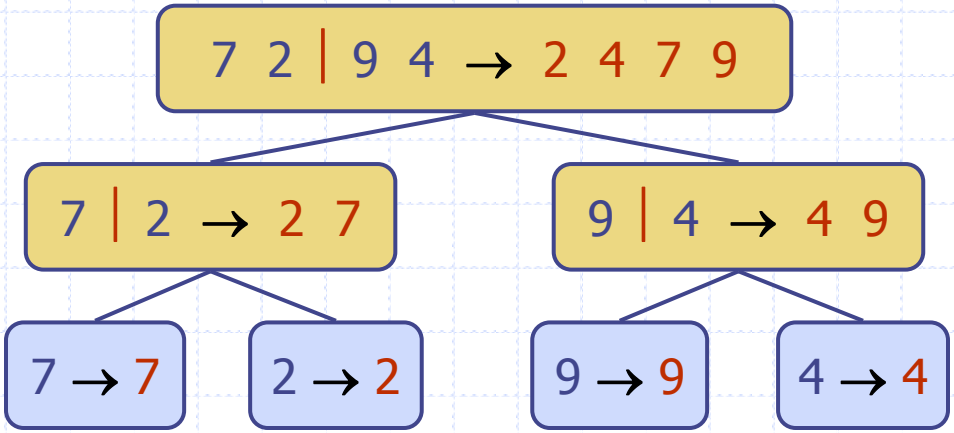
Merge Sort



Divisão e conquista

- Divisão e conquista paradigma geral de projeto de algoritmos:
  - Divisão: divide os dados de entrada S em dois subconjuntos disjuntos S_1 e S_2
  - Recursão: resolve os subproblemas associados a S_1 e S_2
  - Conquista: combina as soluções para S_1 e S_2 em uma solução para S
- Caso base: recursão para subproblemas de tamanho 0 ou 1.

- Merge-sort baseado no paradigma de divisão e conquista
  - Tempo de execução proporcional a $O(n \log n)$
  - Acessa dados de forma sequencial

Merge-Sort

- Merge-sort de uma sequência de entrada v[p..r-1] consiste de três passos:
  - Divisão: divide os dados de entrada ***v[p..r-1]*** em dois subconjuntos disjuntos ***v[p..q-1]*** e ***v[q..r-1]***
  - Recursão: resolve os subproblemas associados aos dois subconjuntos
  - Conquista: combina as soluções para ***as partes*** em uma solução para ***v***
- Caso base: recursão para subproblemas de tamanho 0 ou 1.

```
void mergeSort(int p, int
r, int *v){
    if (p<r-1){
        int q = (p+r)/2;
        mergeSort(p,q,v);
        mergeSort(q,r,v);
        intercala(p,q,r,v);
    }
}
```

Intercalando duas sequências ordenadas (Merging)

- Conquista (junta as duas soluções)
- Intercalação das duas seq. ordenadas, cada uma com n/2 elementos O(n)

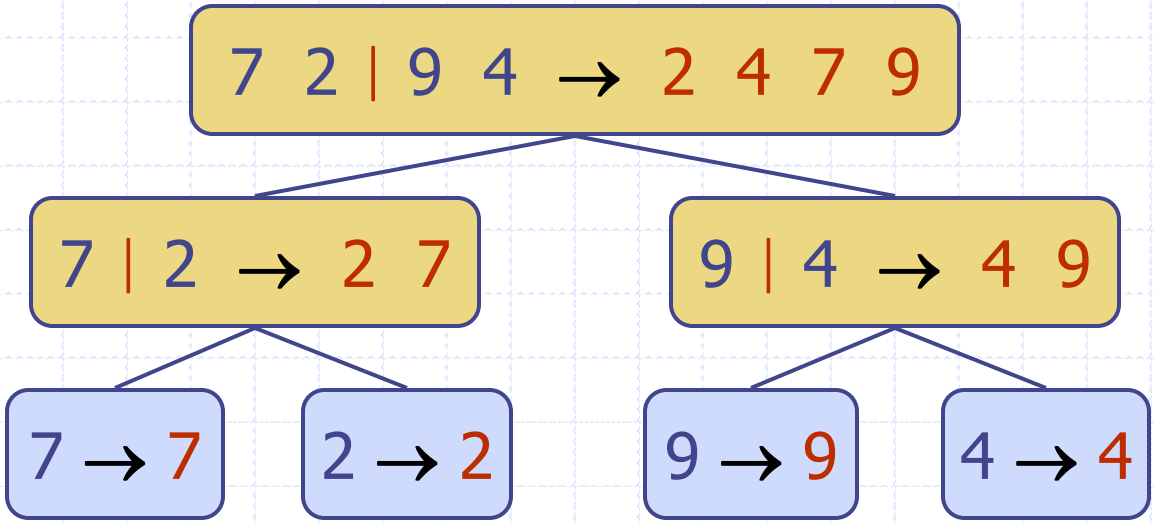
```
// recebe vetores crescentes v[p..q-1] e v[q..r-1]
// interacala também em ordem crescente v[p..r-1]
void intercala(int p, int q, int r, int *v){
   int i,j,k,*aux;
   aux = (int *)malloc((r-p)*sizeof(int)); i = p; j =
q; k = 0;
   while(i<q && j<r)
      if(v[i] <= v[j])
         aux[k++] = v[i++];
      else
         aux[k++] = v[j++];
   while (i<q) aux[k++] = v[i++];
   while (j<r) aux[k++] = v[j++];
   for (i=p; i<r; i++) v[i] = aux[i-p];
   free(aux);
}
```

Intercalando duas sequências ordenadas (Merging)

- **Desempenho da intercalação**
- O tempo que a função consome para fazer o serviço é proporcional ao número de comparações entre elementos do vetor. Esse número é no máximo $r - p - 1$. O consumo de tempo também é proporcional ao número de movimentações, ou seja, cópias de elementos do vetor de um lugar para outro. Esse número é igual a $2(r-p)$. Resumindo, o consumo de tempo da função é *proporcional ao número de elementos do vetor*, ou seja,
  - proporcional a $r - p$.

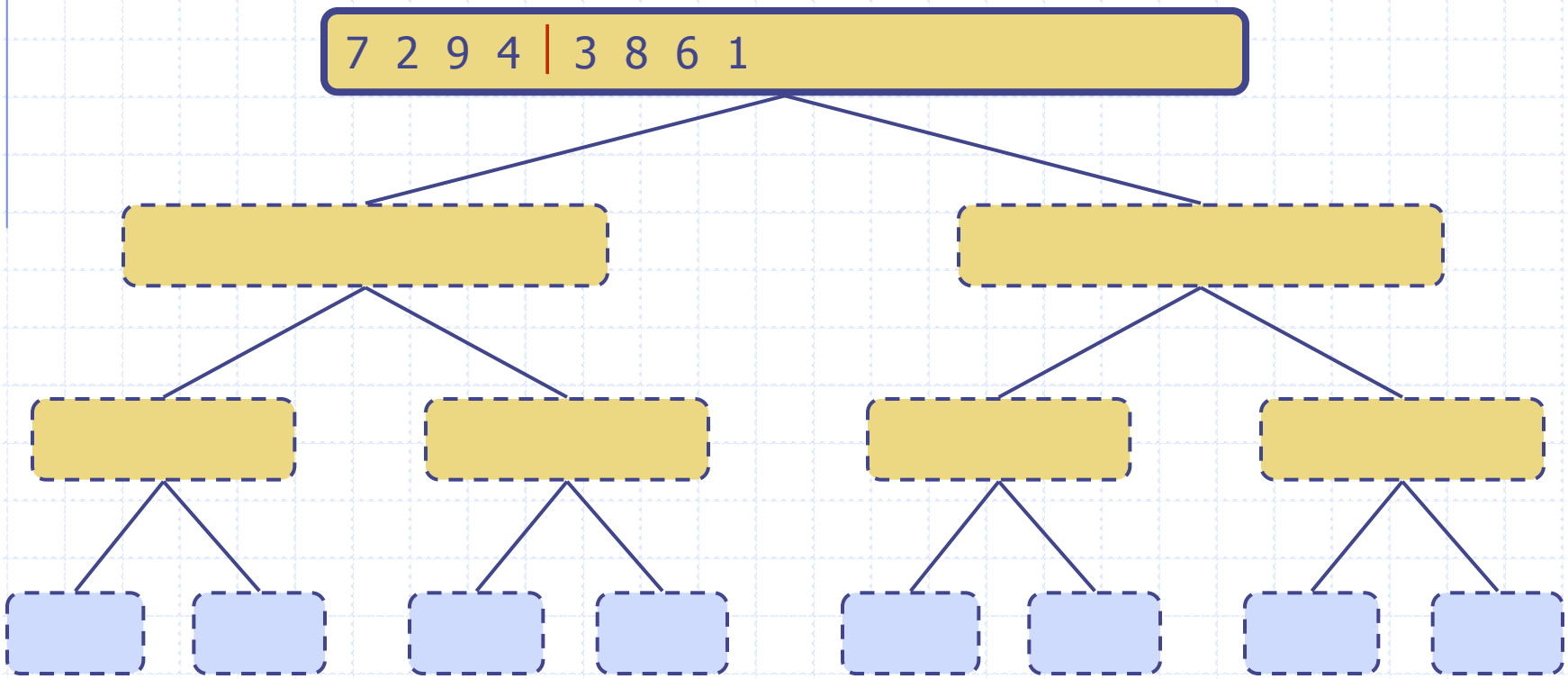
Merge-Sort Tree

- Uma execução do merge-sort é representada por uma árvore binária
  - Cada nó representa uma chamada recursiva do merge-sort e contém a sequência não ordenada antes da execução e suas partições, bem como a sequência ordenada no final da execução
  - A raiz é a chamada inicial
  - As folha são chamadas de subsequências de tamanho 0 ou 1.



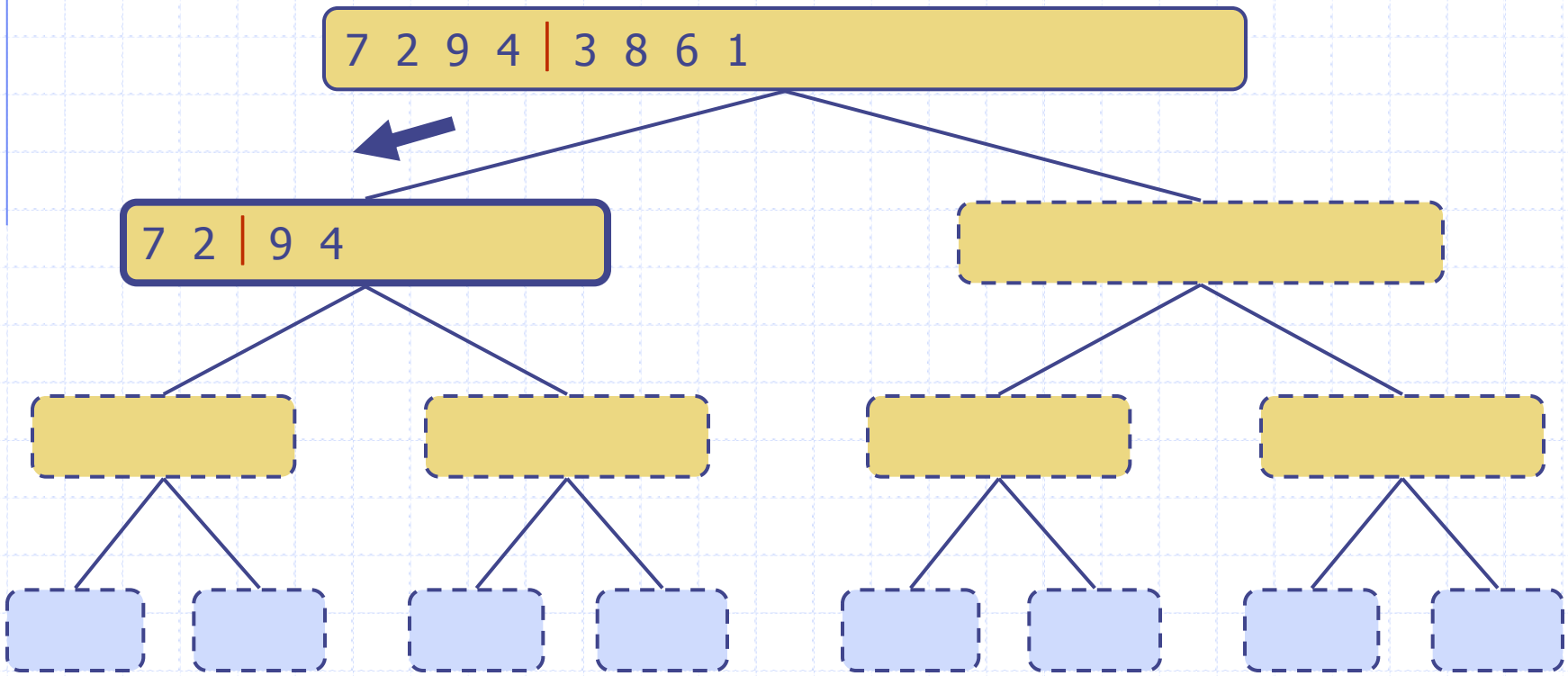
Exemplo de execução

- Particionamento



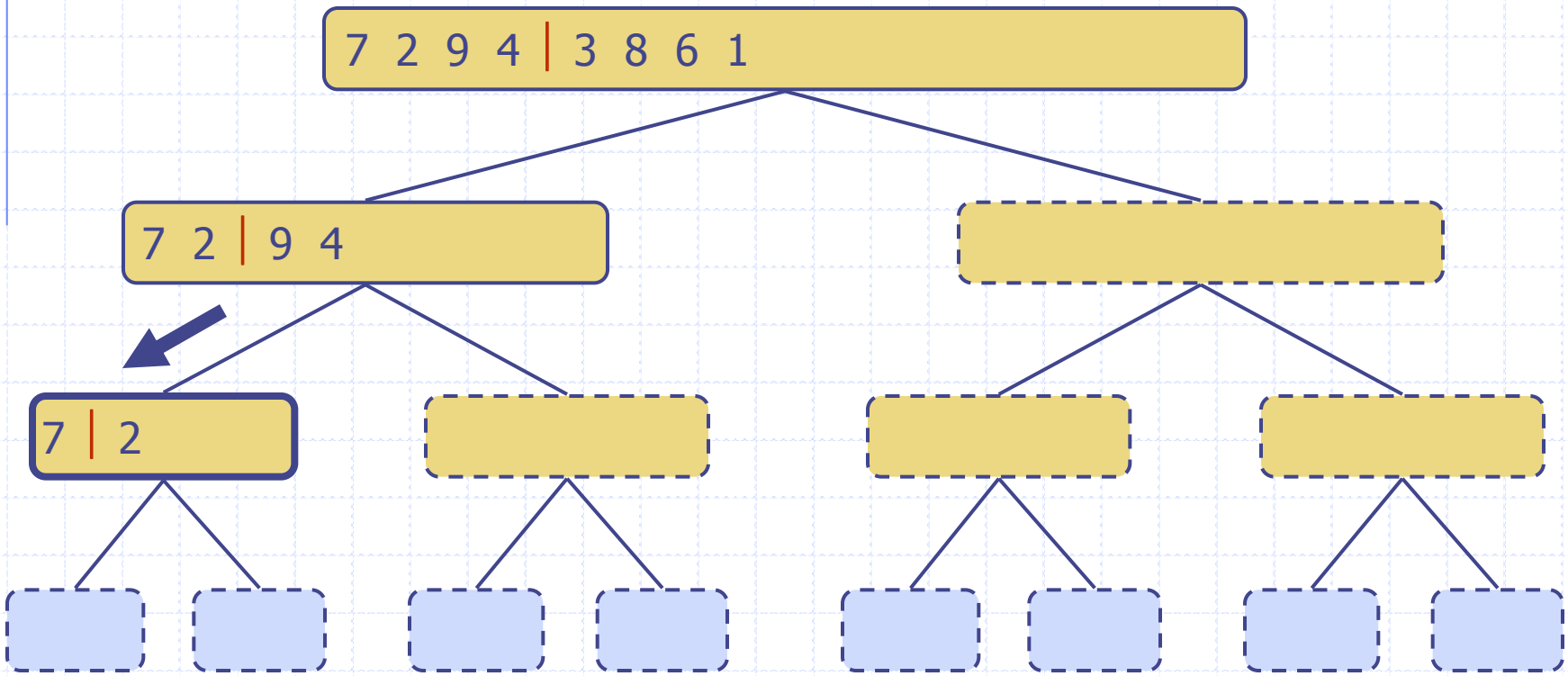
Exemplo de execução (cont.)

- Chamada recursiva, particionamento



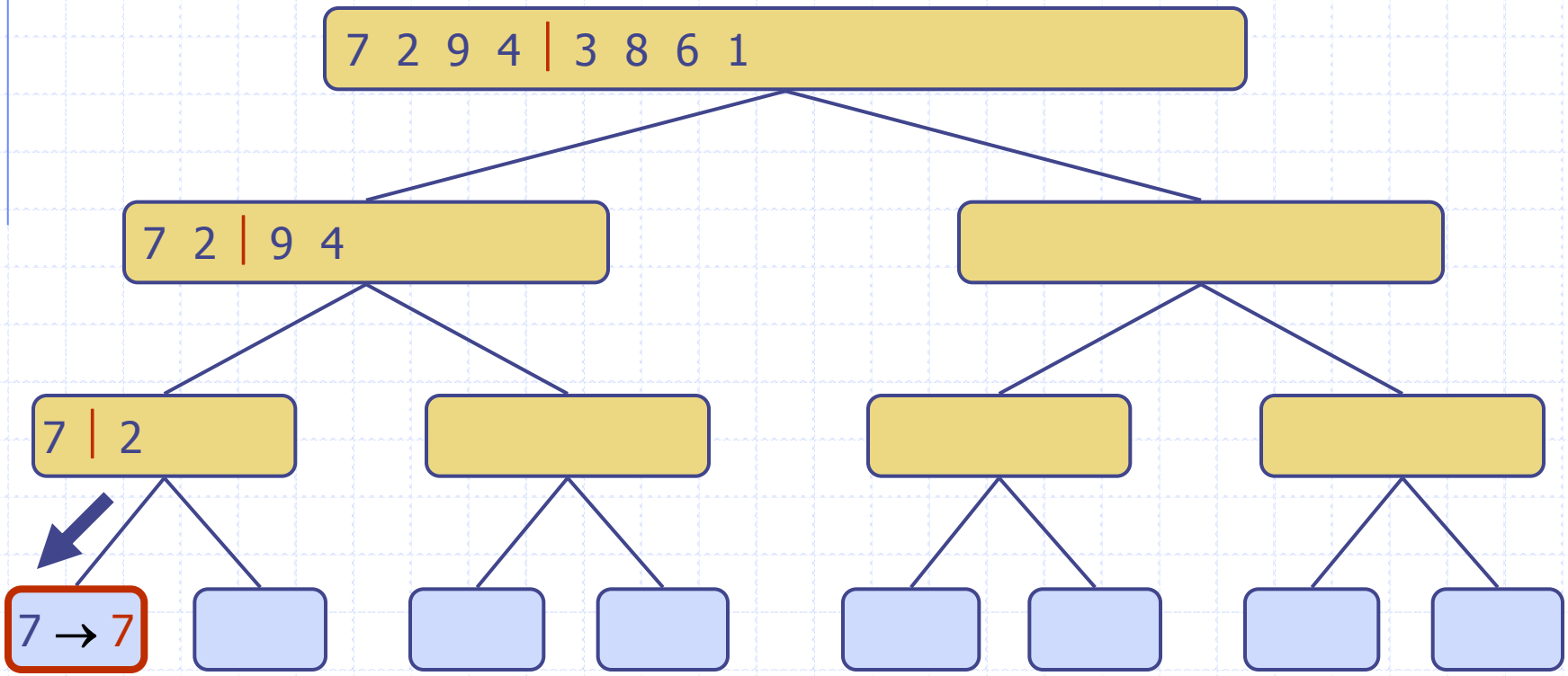
Exemplo de execução (cont.)

- Chamada recursiva, particionamento



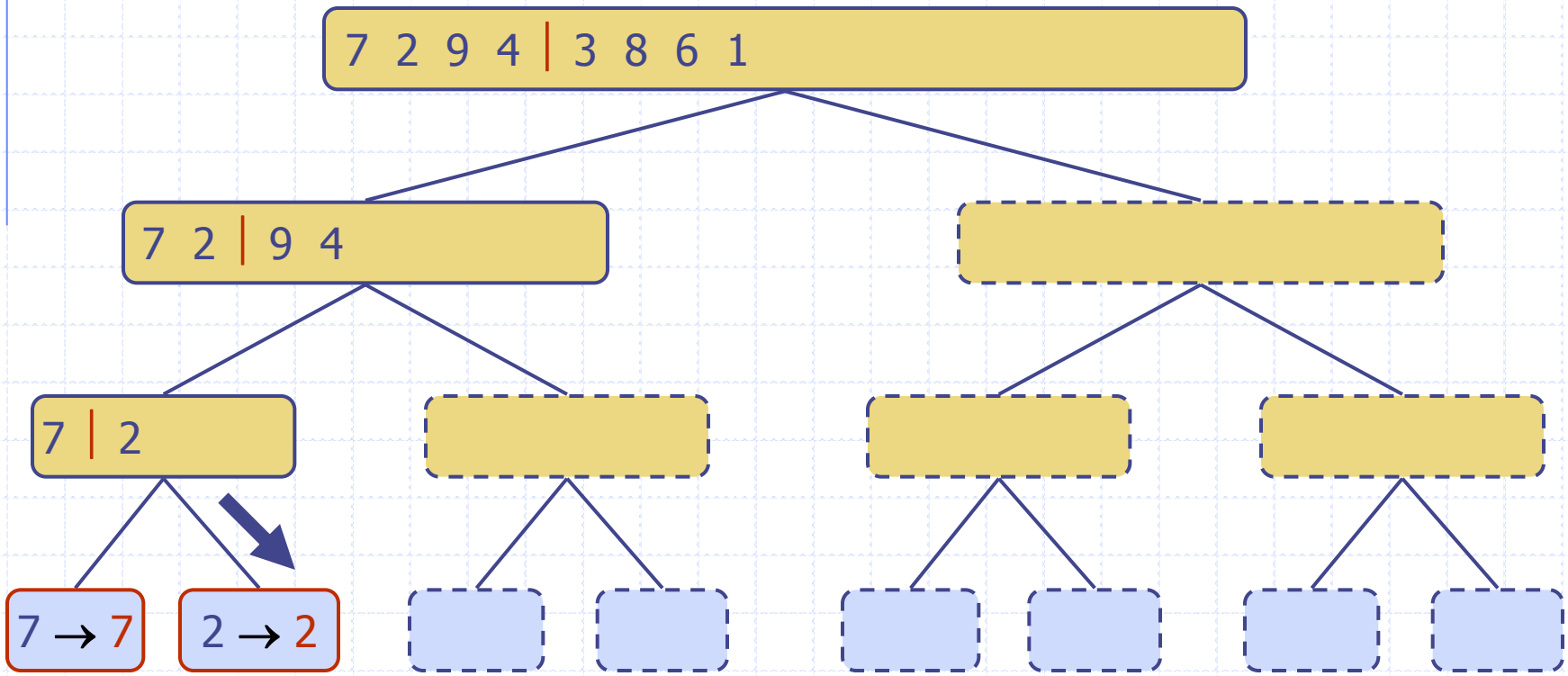
Exemplo de execução (cont.)

- Chamada recursiva, caso base



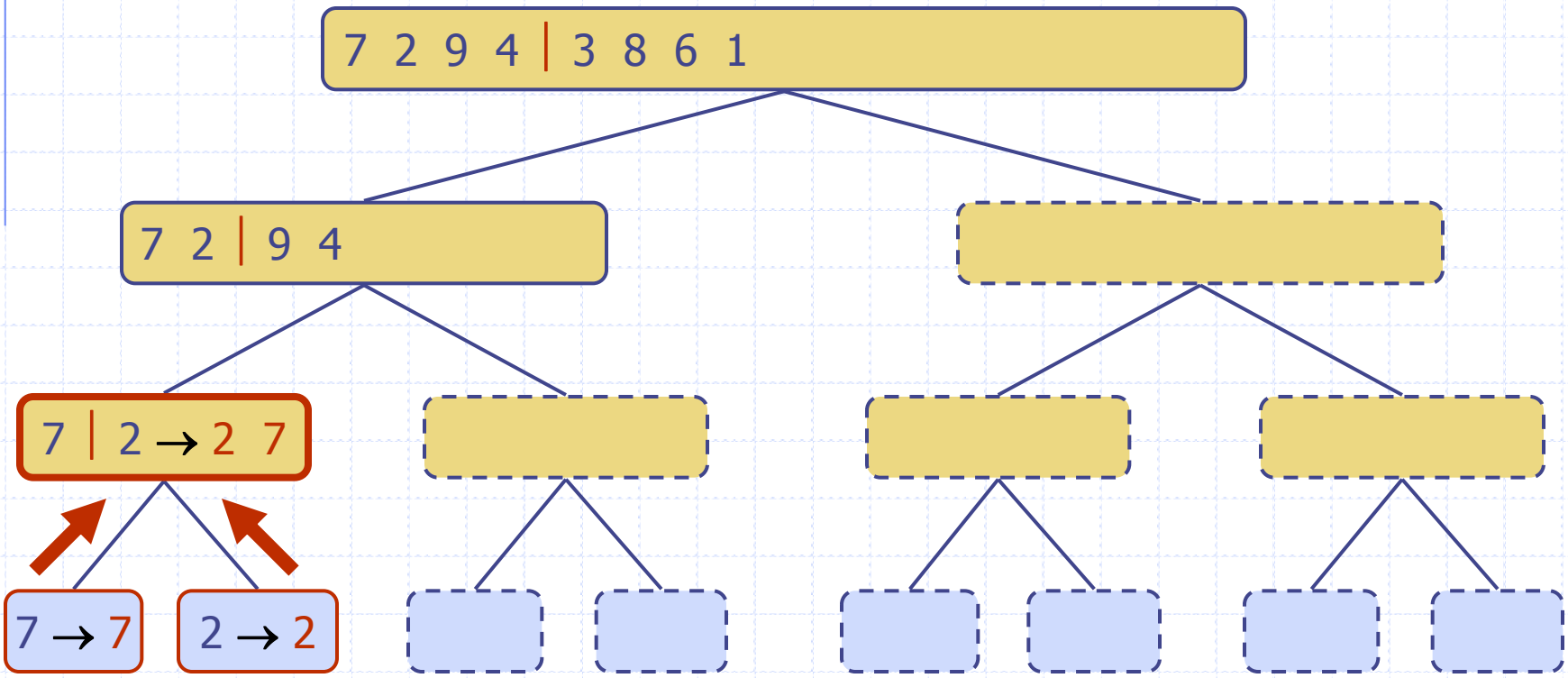
Exemplo de execução (cont.)

- Chamada recursiva, caso base



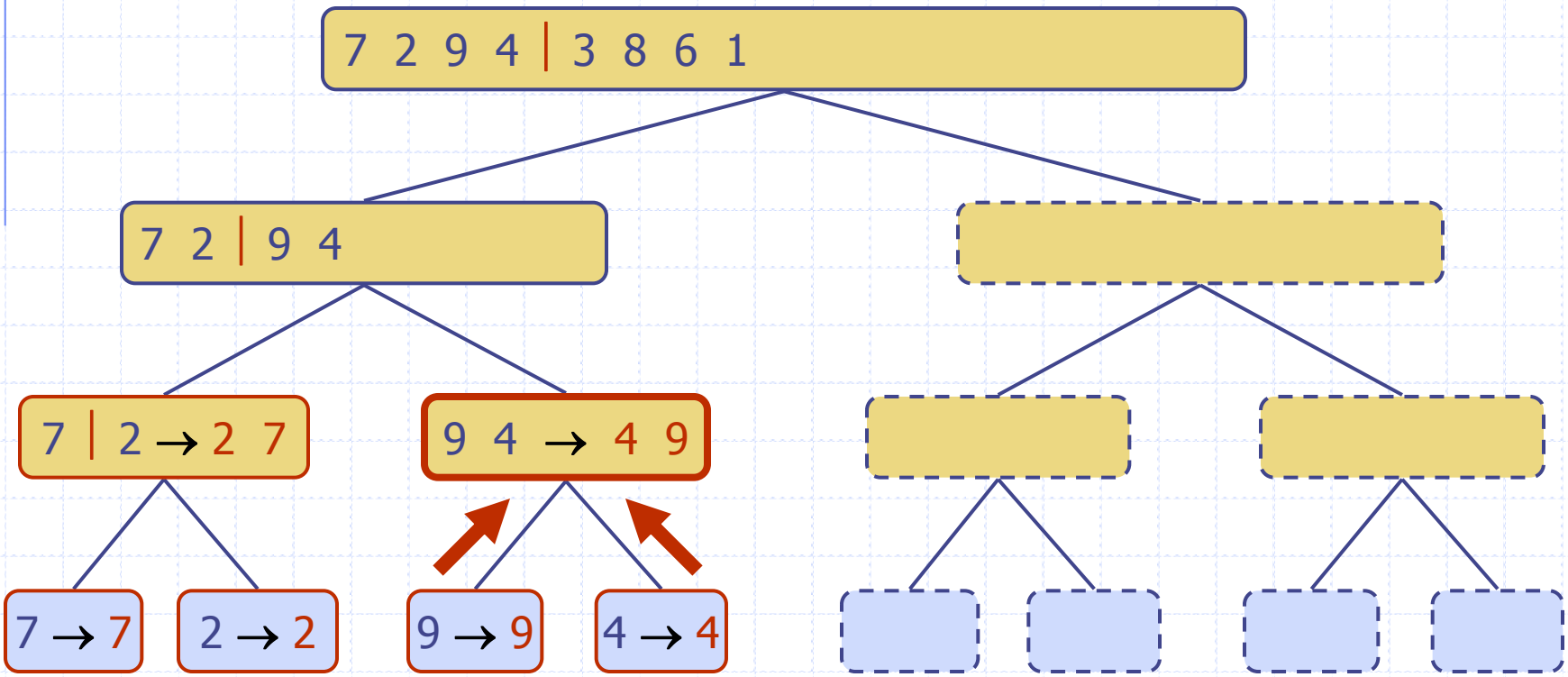
Exemplo de execução (cont.)

- Intercala (merge)



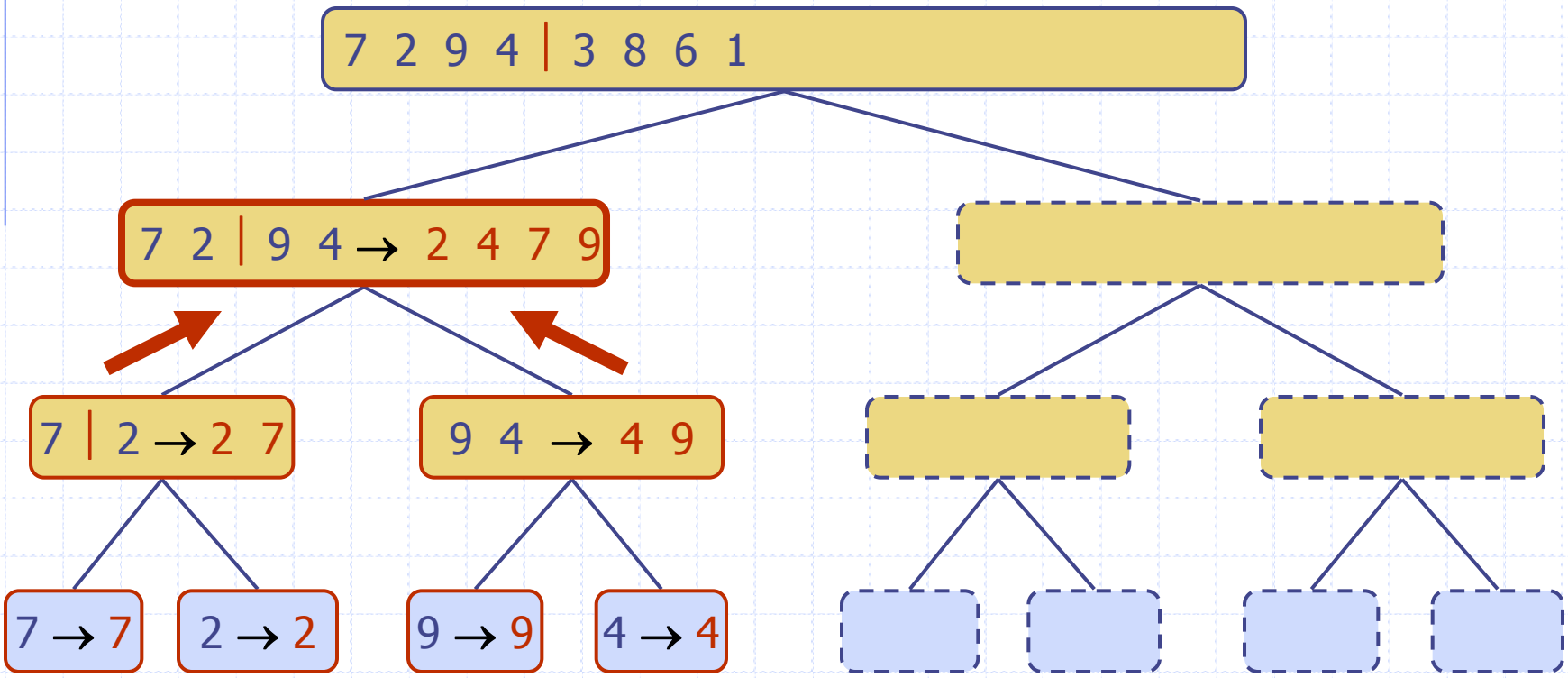
Exemplo de execução (cont.)

- Chamada rec, ..., caso base, merge



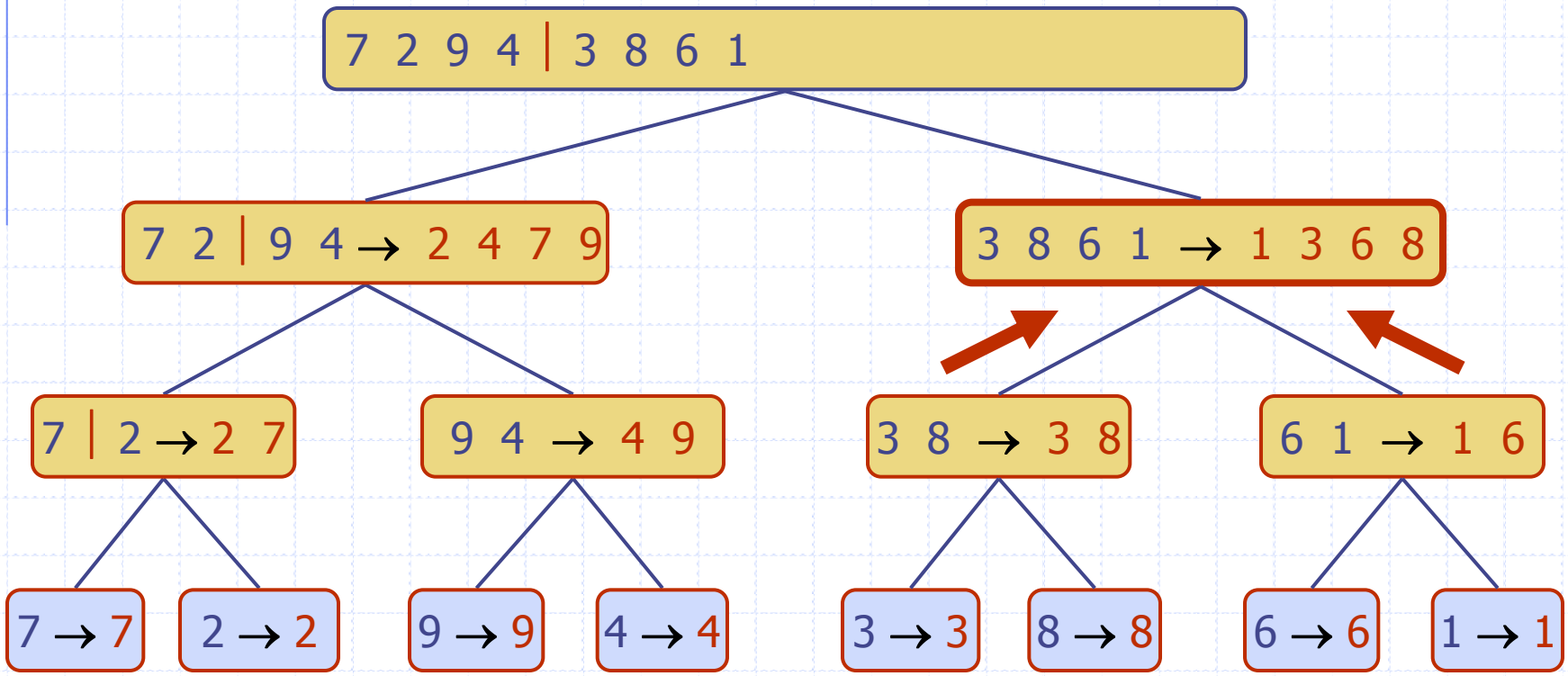
Exemplo de execução (cont.)

- Merge



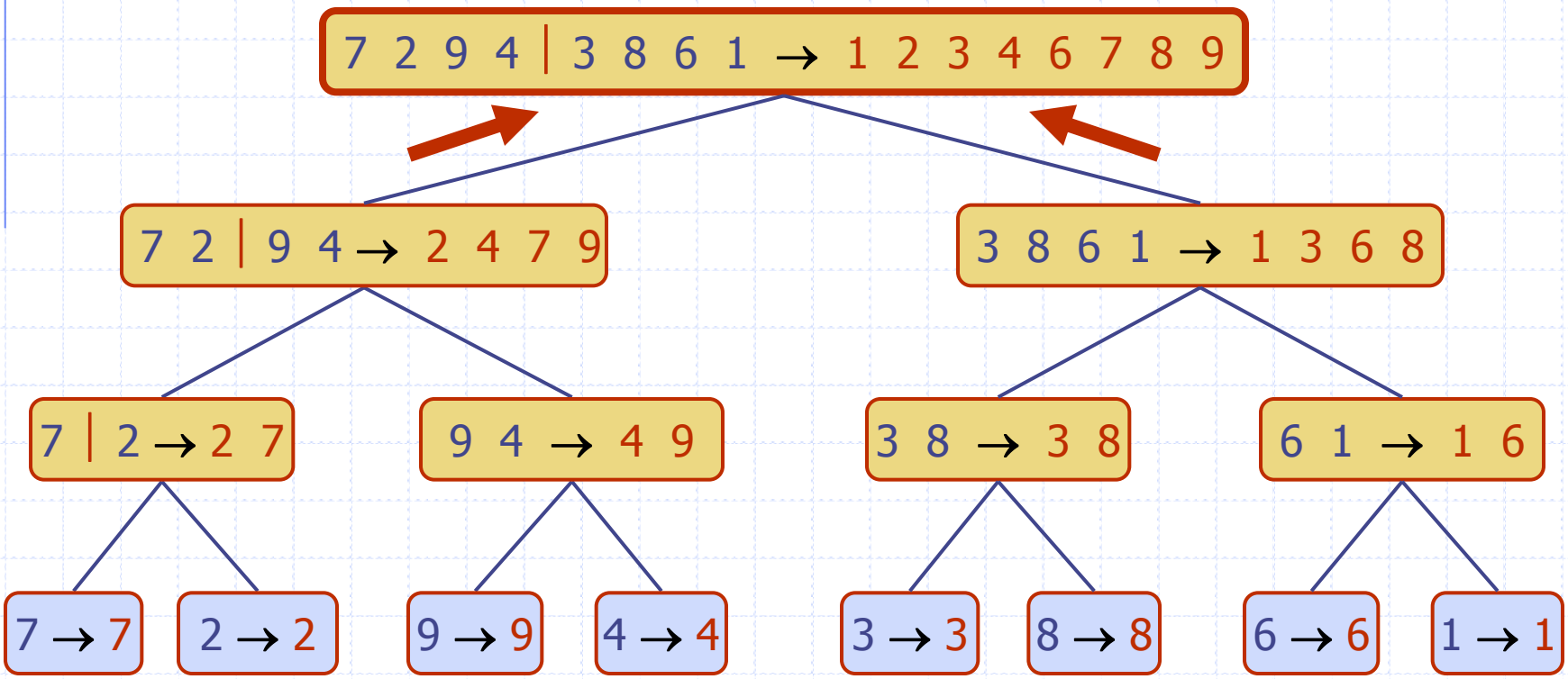
Exemplo de execução (cont.)

- Chamada recursiva, ..., merge, merge



Exemplo de execução (cont.)

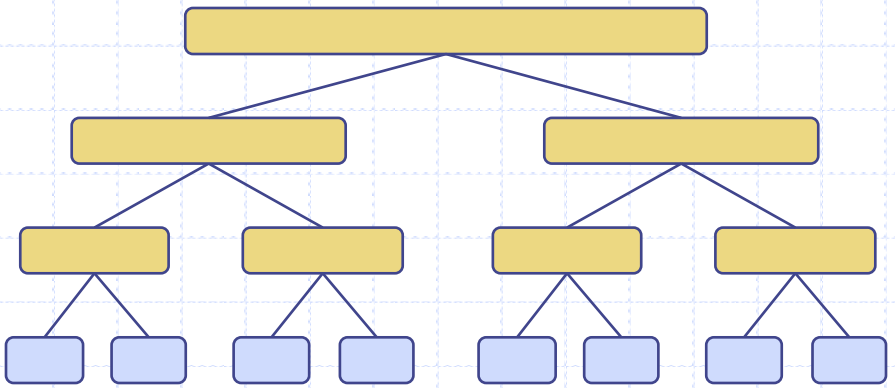
- Merge



Análise do Merge-Sort

- Altura da árvore do merge-sort $O(\log n)$
  - A cada chamada, divide a sequência em duas
- O total de trabalho nos nós de profundidade i é $O(n)$
- The overall amount of work done at the nodes of depth i is $O(n)$
  - we partition and merge 2^i sequences of size $n/2^i$
  - we make 2^{i+1} recursive calls
- Portanto, o tempo total do merge-sort é $O(n \log n)$

depth	#seqs	size
0	1	n
1	2	$n/2$
i	2^i	$n/2^i$
…	…	…



Resumo dos algoritmos de ordenação

Algoritmo	Tempo	Notas
selection-sort insertion-sort buble-sort	$O(n^2)$	◆ lento ◆ in-place ◆ pequenos conjuntos (< 1K)
quick-sort	$O(n \log n)$	◆ rápido ◆ in-place ◆ para grandes cj (1K — 1M)
merge-sort	$O(n \log n)$	◆ rápido ◆ acesso sequências dos dados ◆ para cjs enormes (> 1M)