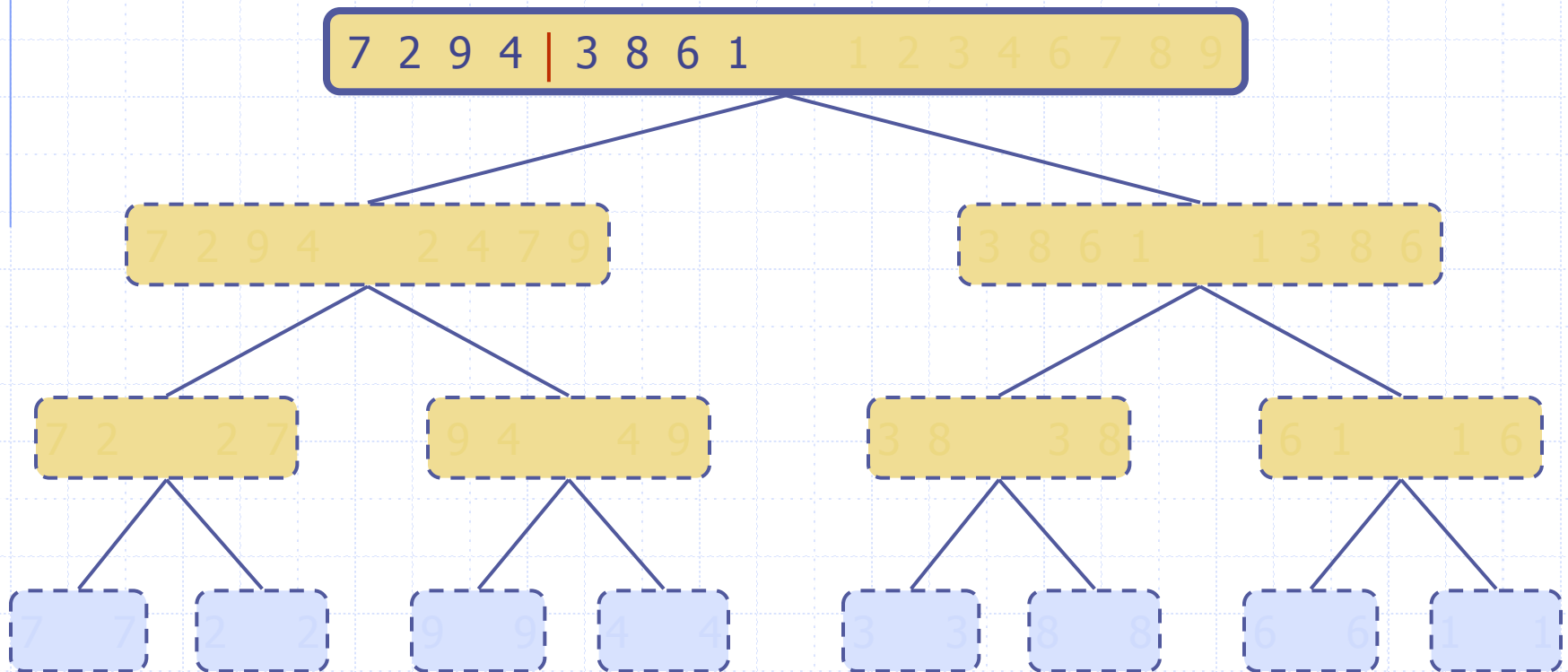


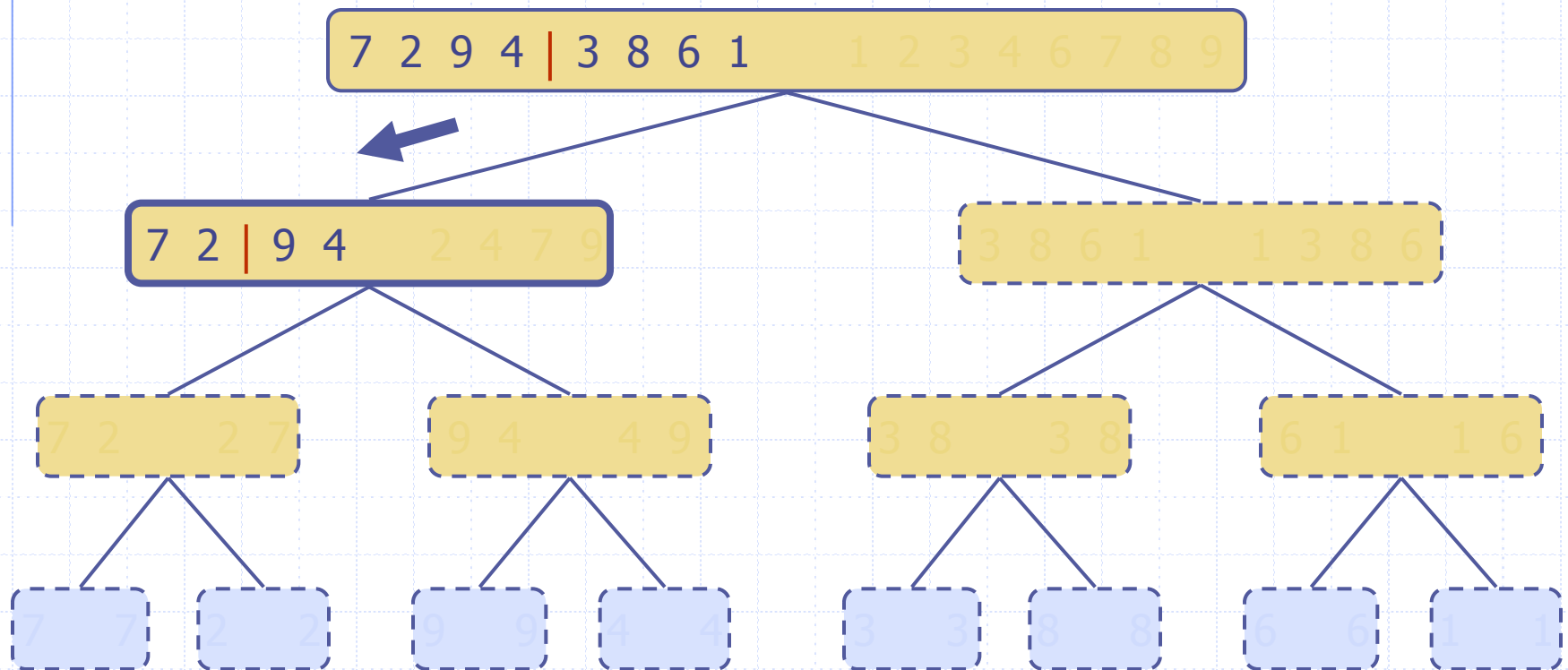
Execution Example

◆ Partition



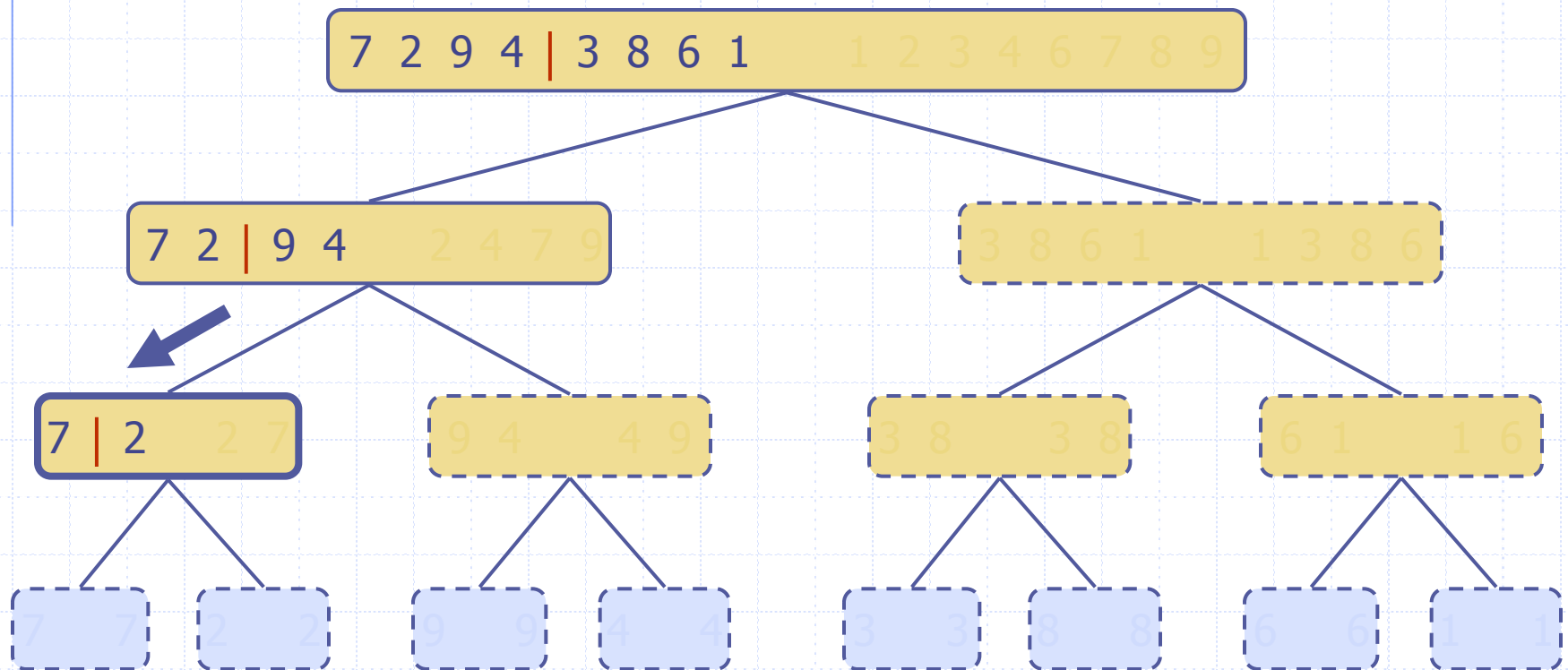
Execution Example (cont.)

◆ Recursive call, partition



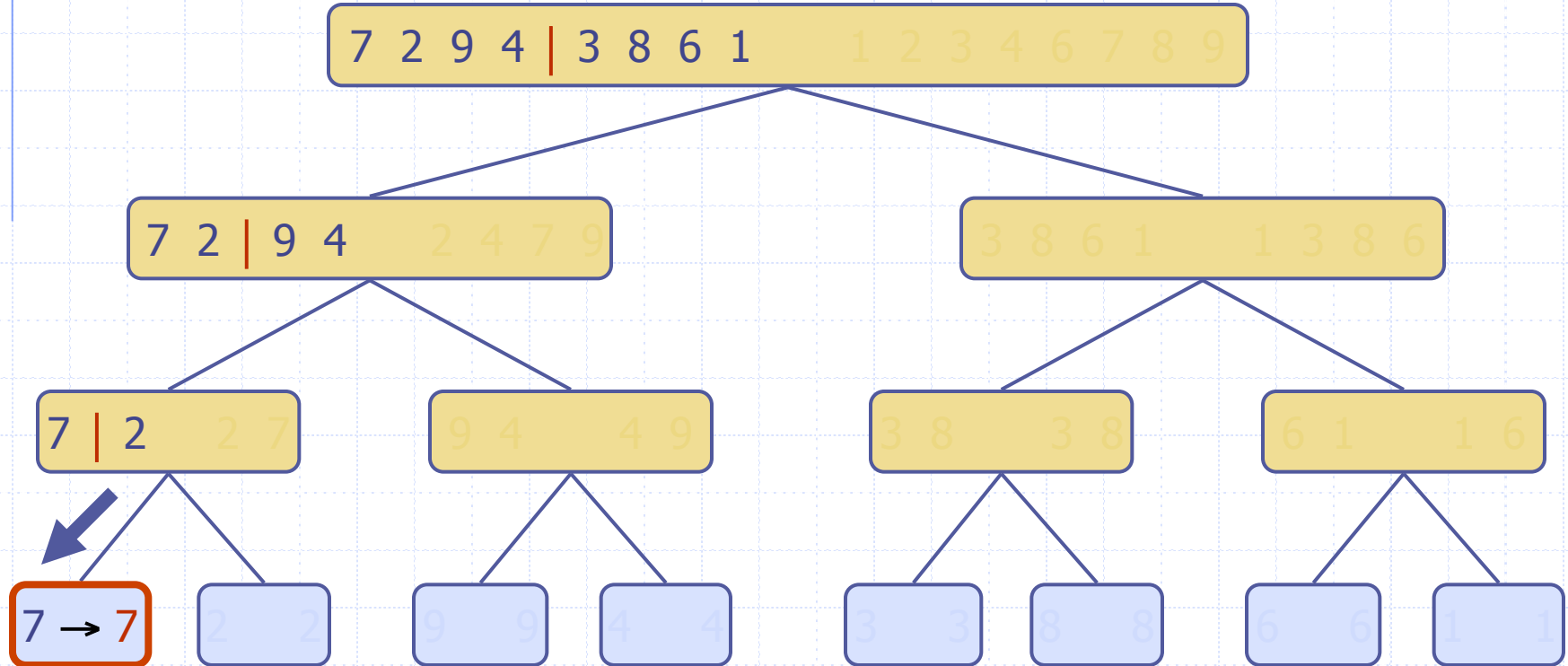
Execution Example (cont.)

◆ Recursive call, partition



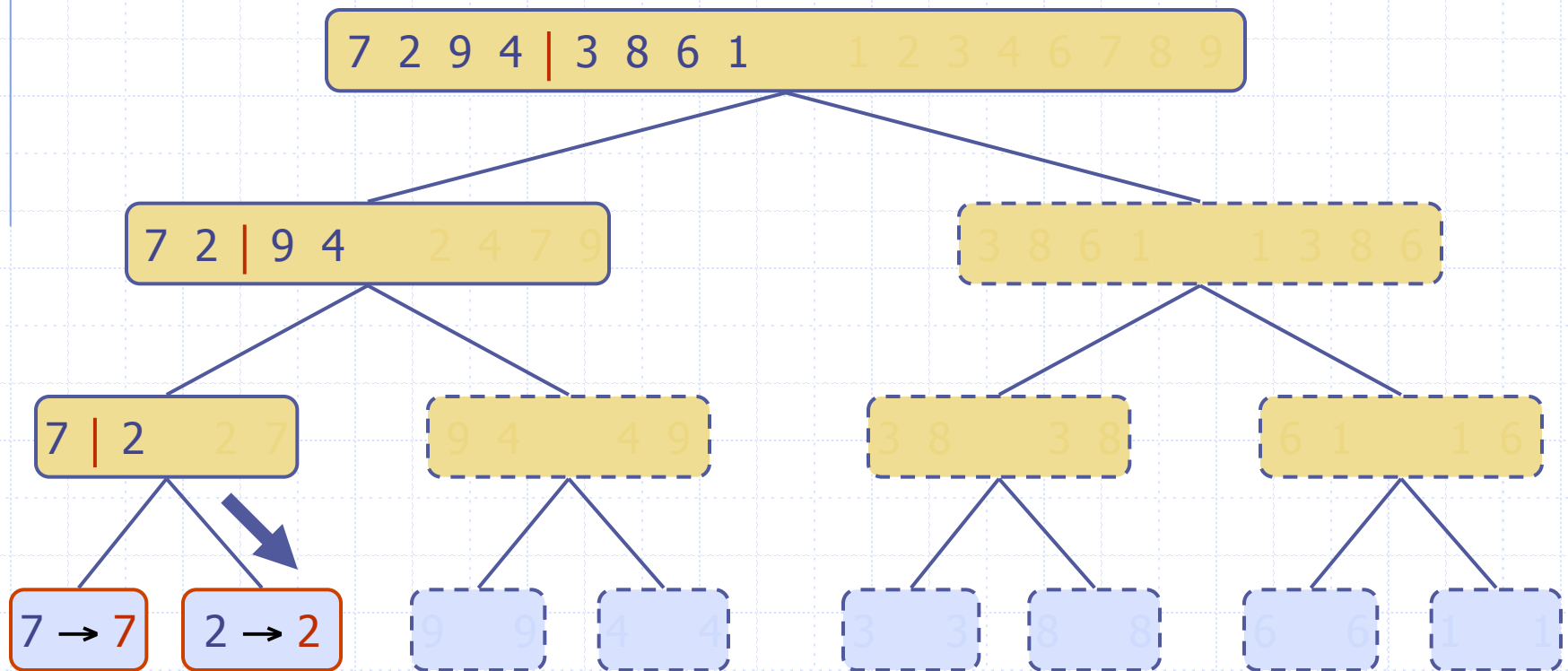
Execution Example (cont.)

◆ Recursive call, base case



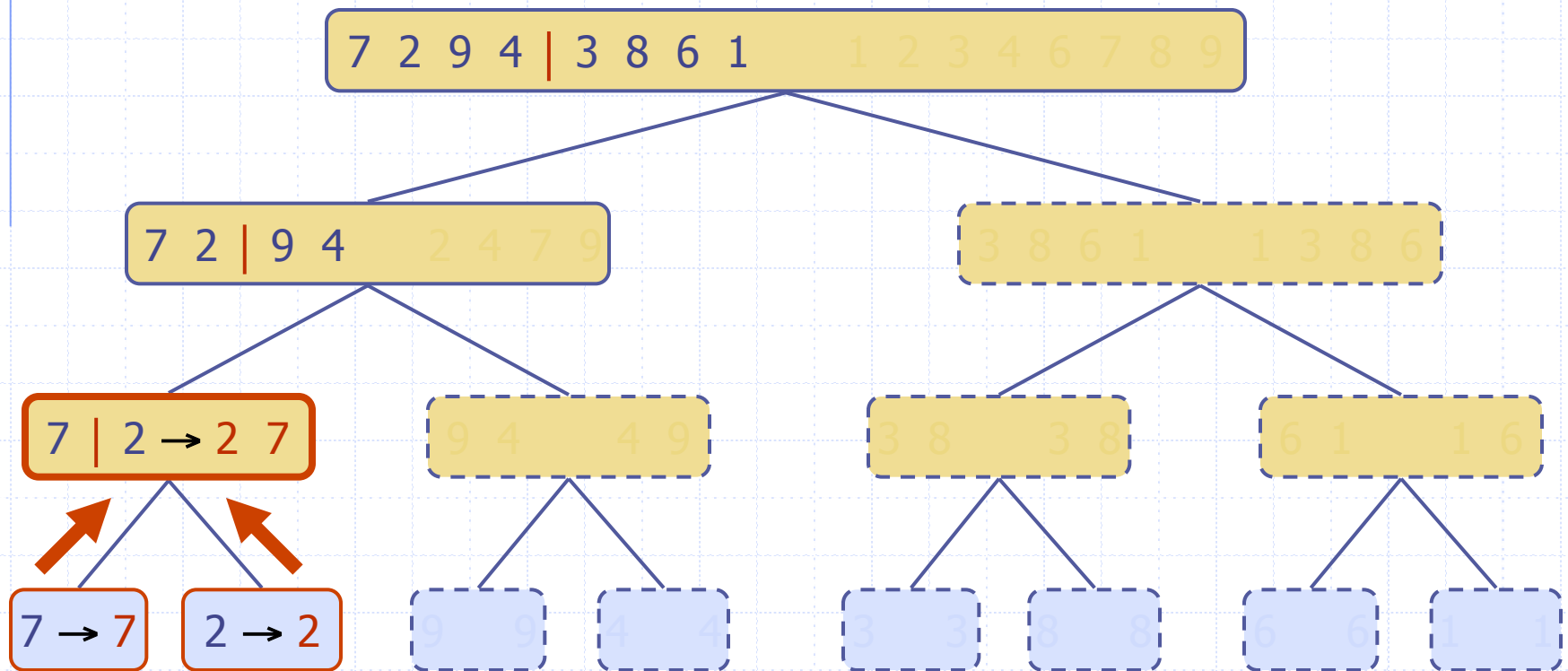
Execution Example (cont.)

◆ Recursive call, base case



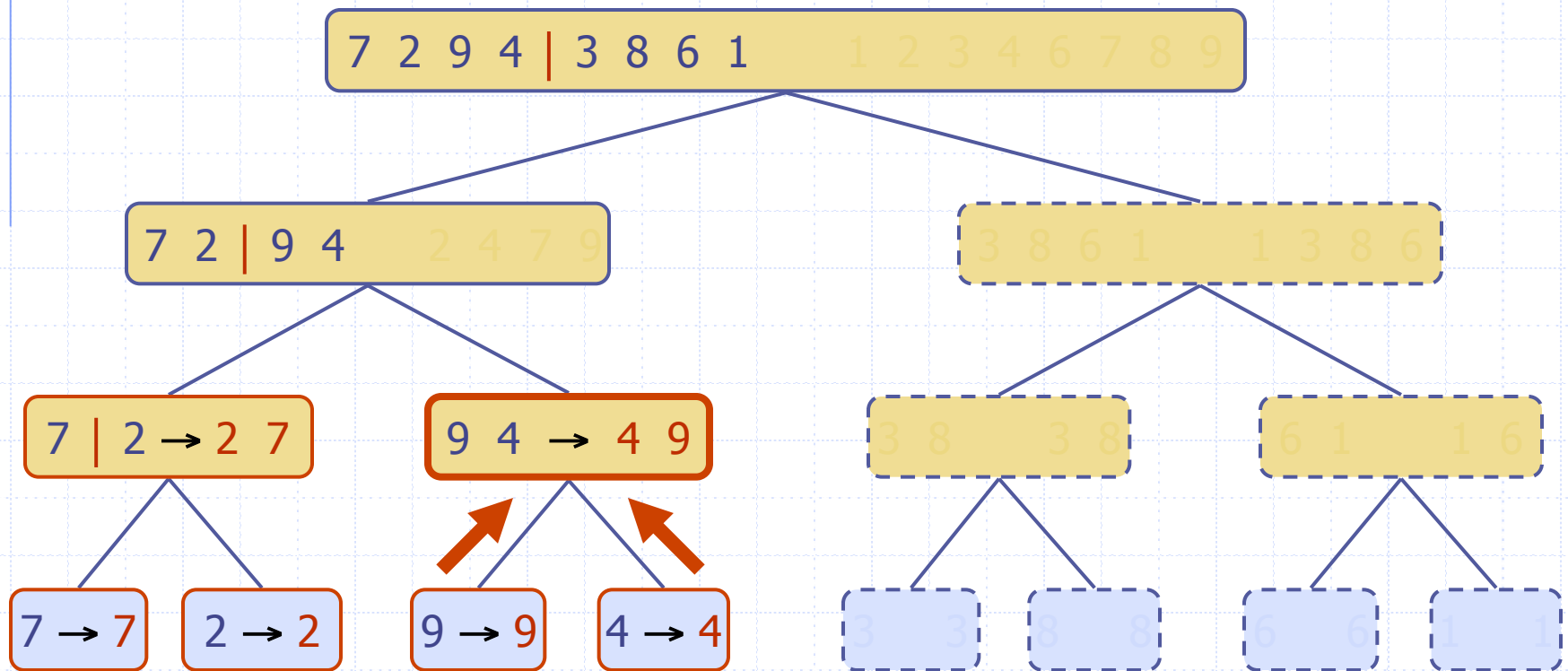
Execution Example (cont.)

◆ Merge



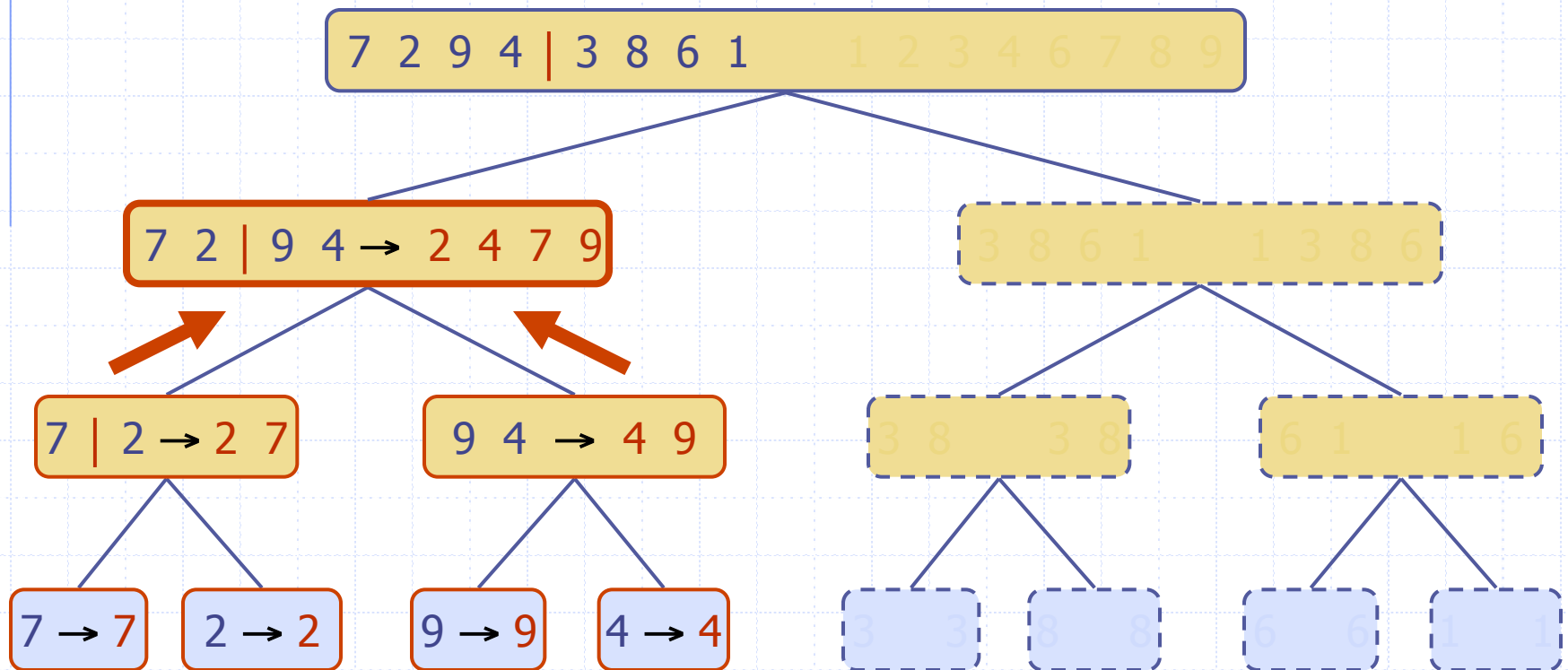
Execution Example (cont.)

◆ Recursive call, ..., base case, merge



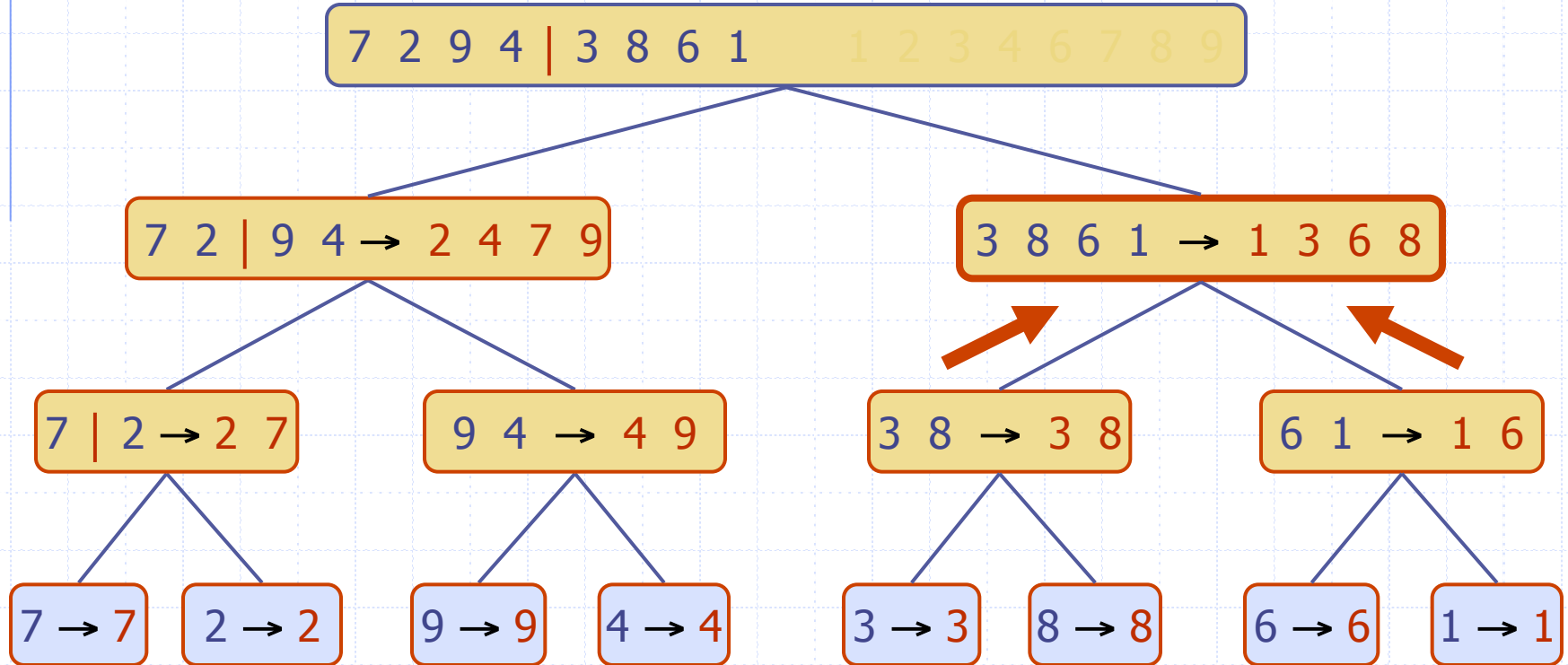
Execution Example (cont.)

◆ Merge



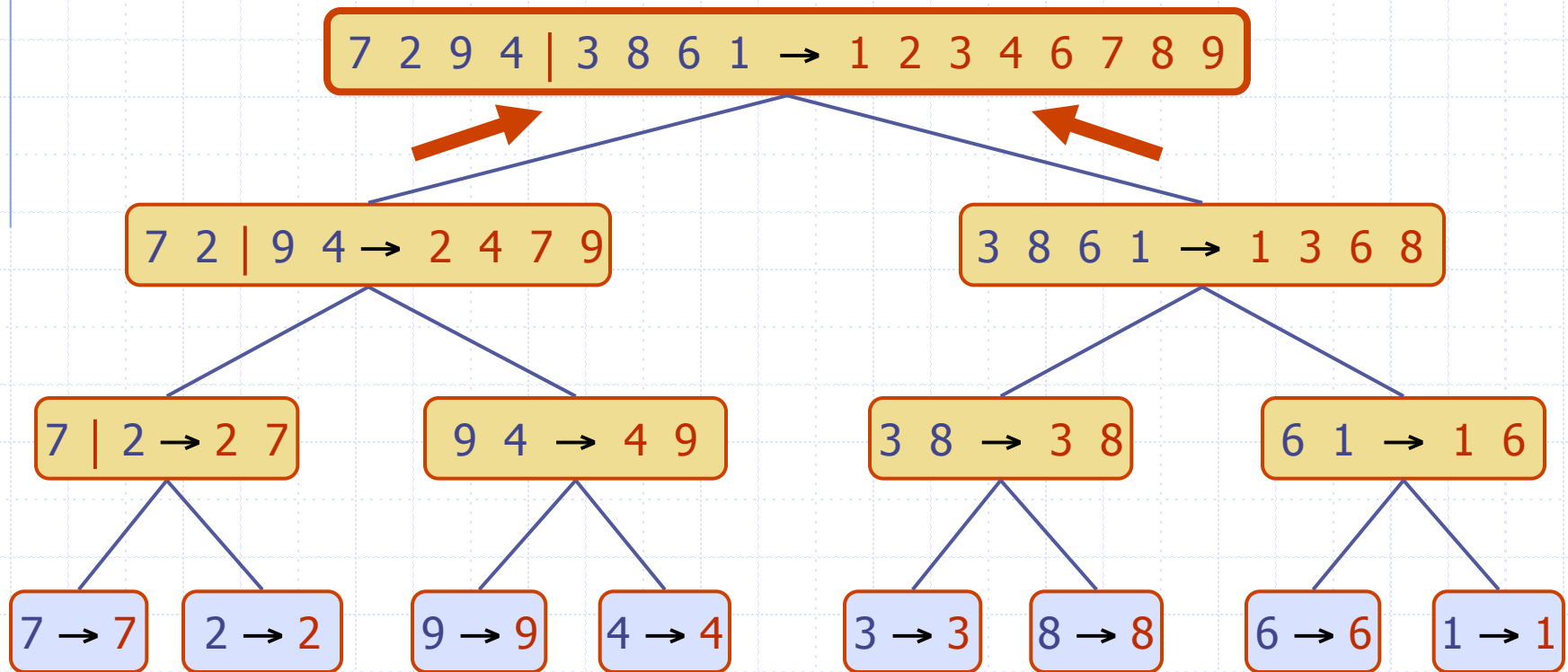
Execution Example (cont.)

◆ Recursive call, ..., merge, merge



Execution Example (cont.)

◆ Merge



Analysis of Merge-Sort

- ◆ The height h of the merge-sort tree is $O(\log n)$
 - at each recursive call we divide in half the sequence,
- ◆ 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
- ◆ Thus, the total running time of merge-sort is $O(n \log n)$

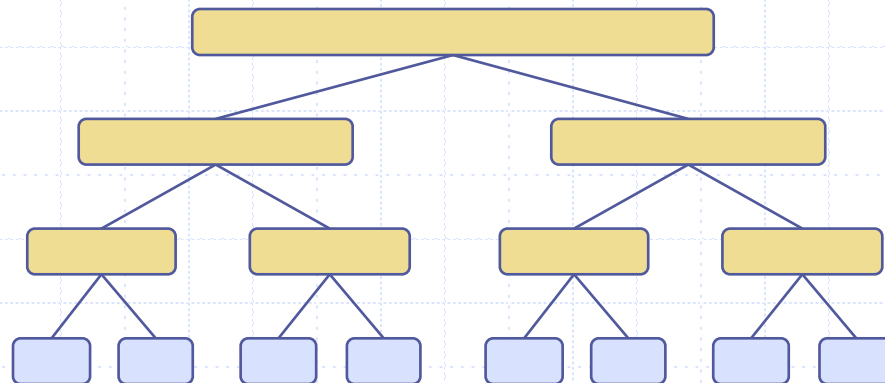
depth	#seqs	size
-------	-------	------

0	1	n
---	---	-----

1	2	$n/2$
---	---	-------

i	2^i	$n/2^i$
-----	-------	---------

...
-----	-----	-----



Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	<ul style="list-style-type: none">slowin-placefor small data sets (< 1K)
insertion-sort	$O(n^2)$	<ul style="list-style-type: none">slowin-placefor small data sets (< 1K)
heap-sort	$O(n \log n)$	<ul style="list-style-type: none">fastin-placefor large data sets (1K — 1M)
merge-sort	$O(n \log n)$	<ul style="list-style-type: none">fastsequential data accessfor huge data sets (> 1M)