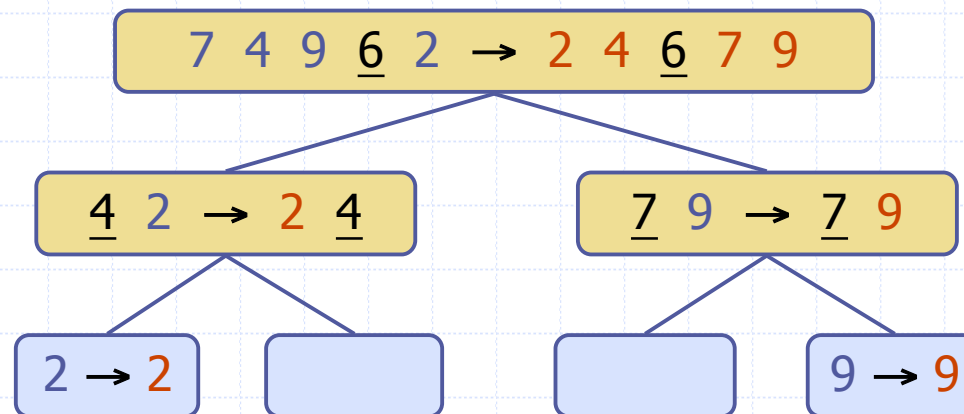
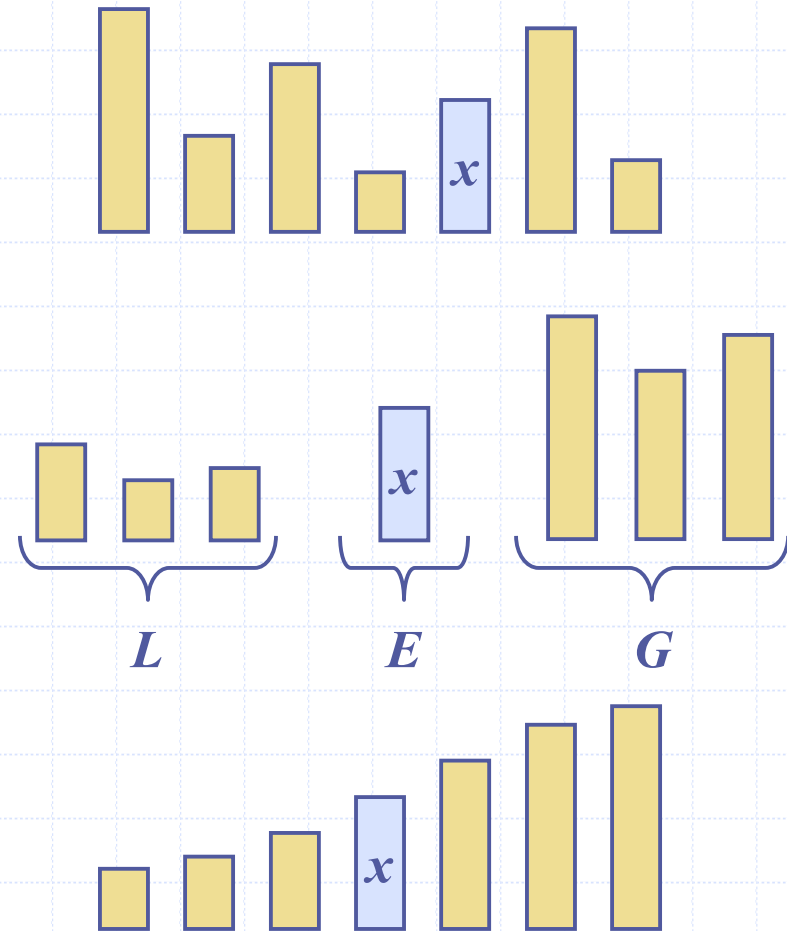


Quick-Sort

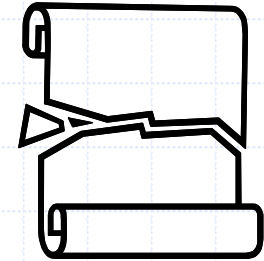


Quick-Sort (§ 10.2)

- ◆ **Quick-sort** is a randomized sorting algorithm based on the divide-and-conquer paradigm:
 - **Divide**: pick a random element x (called **pivot**) and partition S into
 - ◆ L elements less than x
 - ◆ E elements equal to x
 - ◆ G elements greater than x
 - **Recur**: sort L and G
 - **Conquer**: join L , E and G



Partition



- ◆ We partition an input sequence as follows:
 - We remove, in turn, each element y from S and
 - We insert y into L , E or G , depending on the result of the comparison with the pivot x
- ◆ Each insertion and removal is at the beginning or at the end of a sequence, and hence takes $O(1)$ time
- ◆ Thus, the partition step of quick-sort takes $O(n)$ time

Algorithm *partition*(S, p)

Input sequence S , position p of pivot

Output subsequences L, E, G of the elements of S less than, equal to, or greater than the pivot, resp.

$L, E, G \leftarrow$ empty sequences

$x \leftarrow S.remove(p)$

while $\neg S.isEmpty()$

$y \leftarrow S.remove(S.first())$

if $y < x$

$L.insertLast(y)$

else if $y = x$

$E.insertLast(y)$

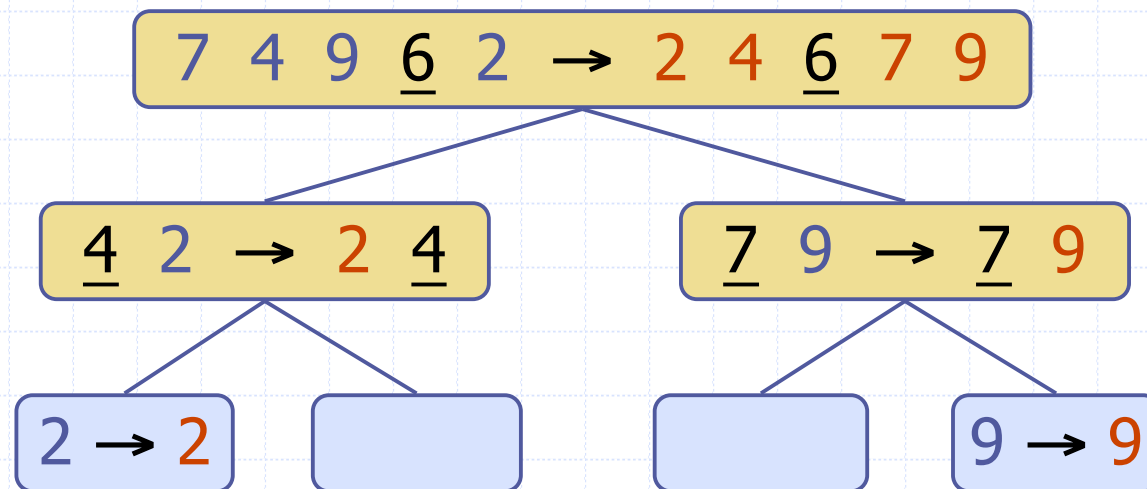
else $\{ y > x \}$

$G.insertLast(y)$

return L, E, G

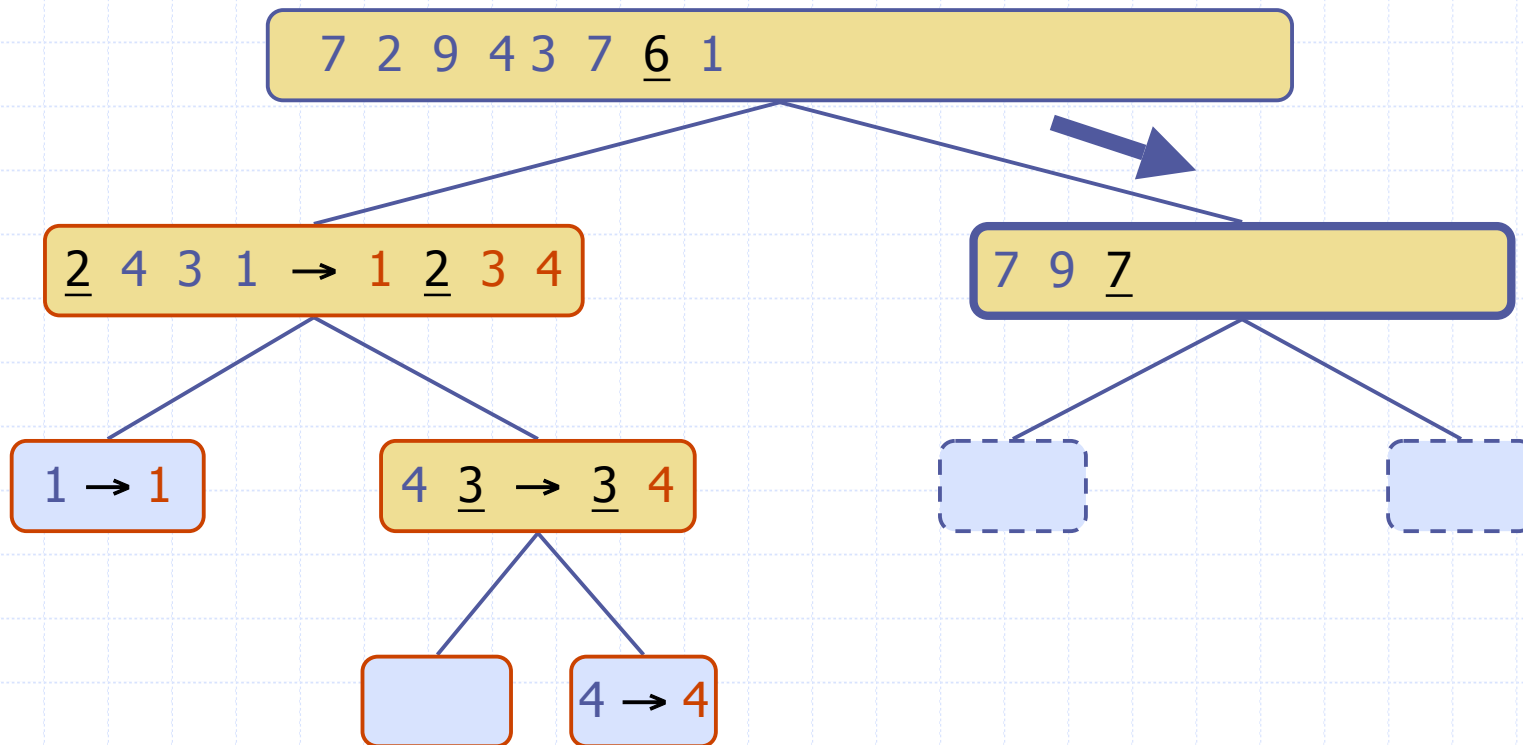
Quick-Sort Tree

- ◆ An execution of quick-sort is depicted by a binary tree
 - Each node represents a recursive call of quick-sort and stores
 - ◆ Unsorted sequence before the execution and its pivot
 - ◆ Sorted sequence at the end of the execution
 - The root is the initial call
 - The leaves are calls on subsequences of size 0 or 1



Execution Example (cont.)

◆ Recursive call, pivot selection



Execution Example (cont.)

◆ Join, join

