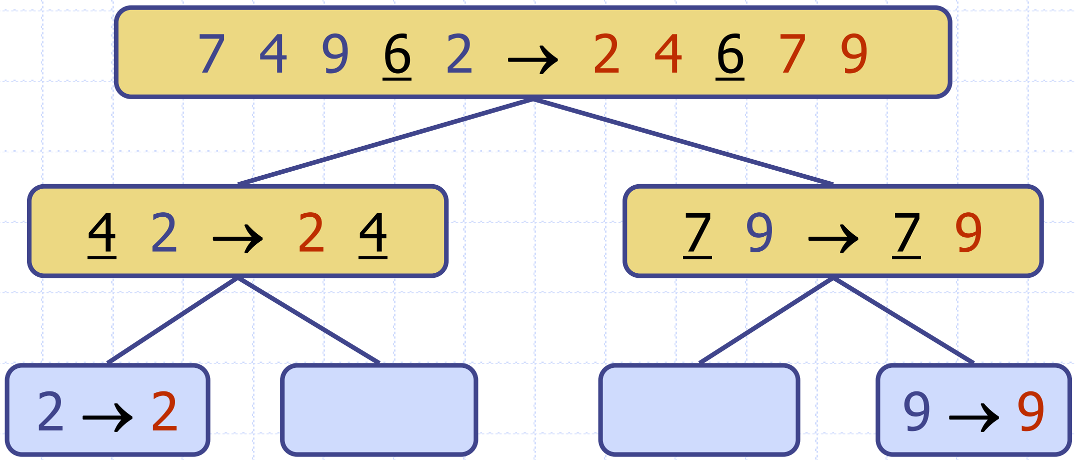
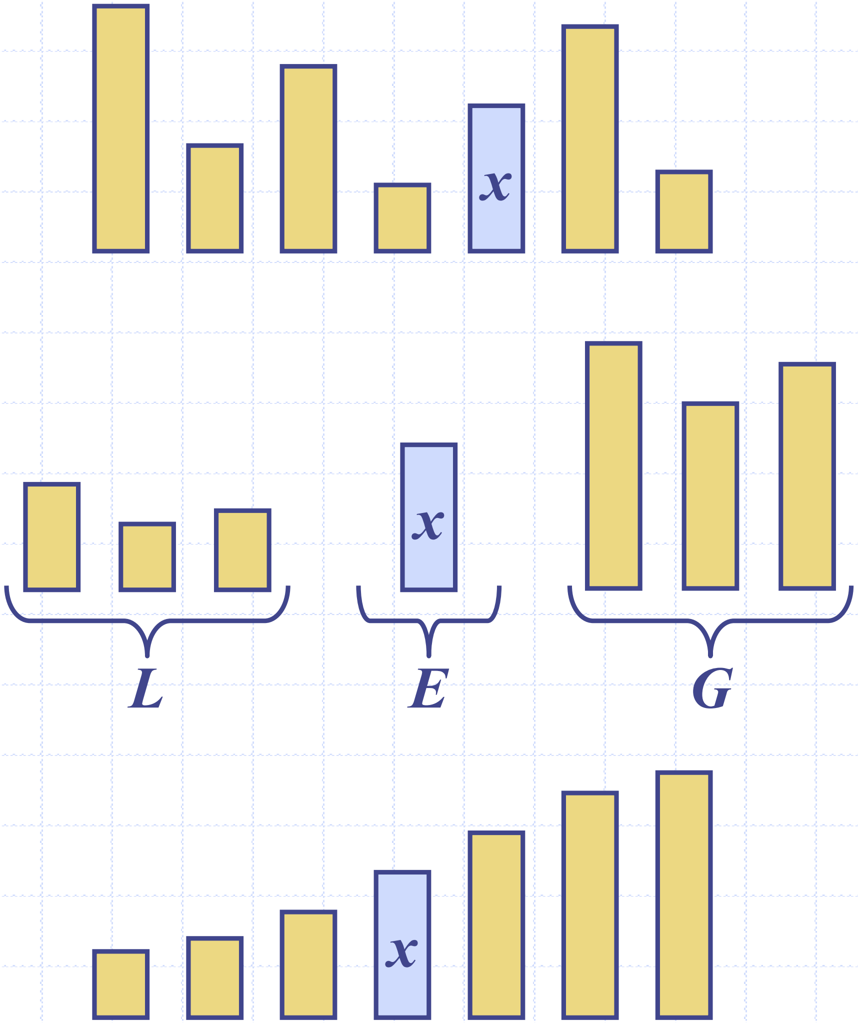
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014 With notes by R. Solis-Oba

Quick-Sort



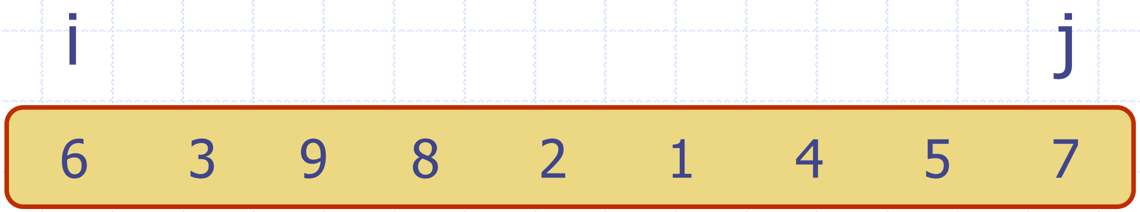
Quick-Sort

- Quick-sort is a sorting algorithm based on the divide-and-conquer paradigm:
  - Divide: pick an element x called pivot and partition the array A into 3 sets
    - L: elements less than x
    - G: elements equal x
    - E: elements greater than x
  - Recur: sort L and G
  - Conquer: join L, E and G



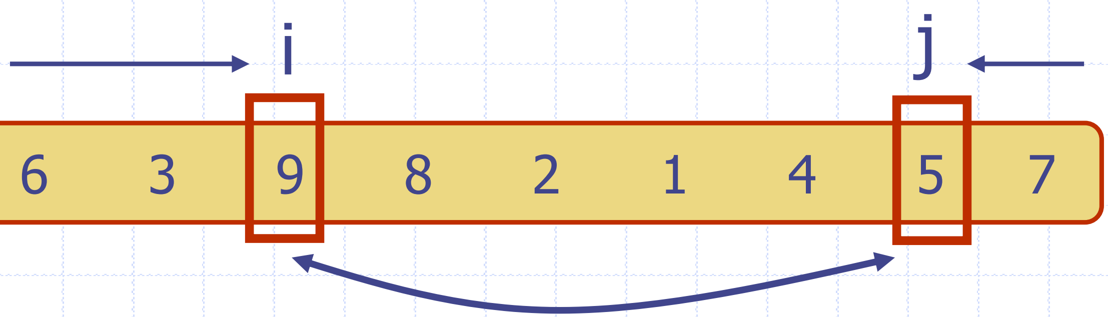
Partitioning

- Perform the partitioning of A using two indices i and j to rearrange the elements in A, so values smaller than the pivot appear on the left, values larger than the pivot appear on the right, and the pivot appears in the middle



(pivot = 6)

- Repeat until i and j cross:
  - Scan i to the right until finding an element > pivot.
  - Scan k to the left until finding an element < pivot.
  - Swap elements at indices i and j



Quick-Sort Algorithm

Algorithm quicksort (A, first, last)
Input: Array A, indices of first and last elements in A
Output: Sorted array A

```
if first < last then { // Array has at least 2 elements
    m ← partition(A,first,last) // returns position of the pivot
    quicksort(A, first, m)
    quicksort(A, m+1, last)
}
```

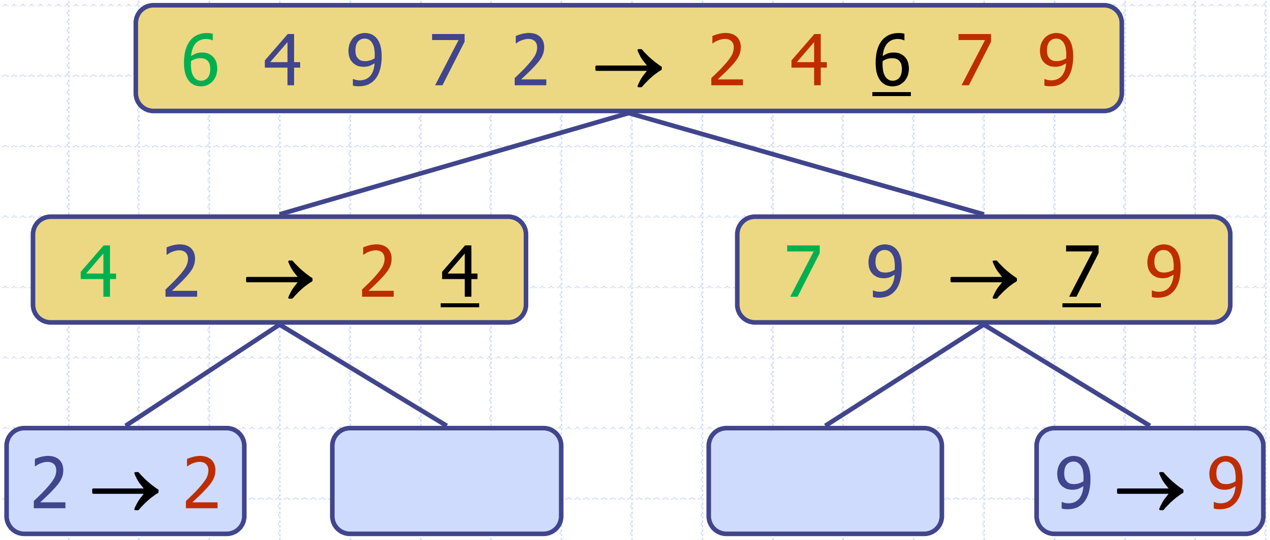
Partition

Algorithm partition (A, first, last)
Input: Array A, indices of first and last elements in A
Output: Position of pivot after partitioning A with respect to A[first]

```
pivot ← A[first]
i ← first + 1
j ← last
while i ≤ j do {
    while A[i] < pivot do i ← i + 1
    while A[j] > pivot do j ← j – 1
    if i < j then { // Swap A[i] and A[j]
        tmp ←A[i]
        A[i] ← A[j]
        A[j] ← tmp
    }
}
A[first] ← A[j]
A[j] ← pivot
return j
```

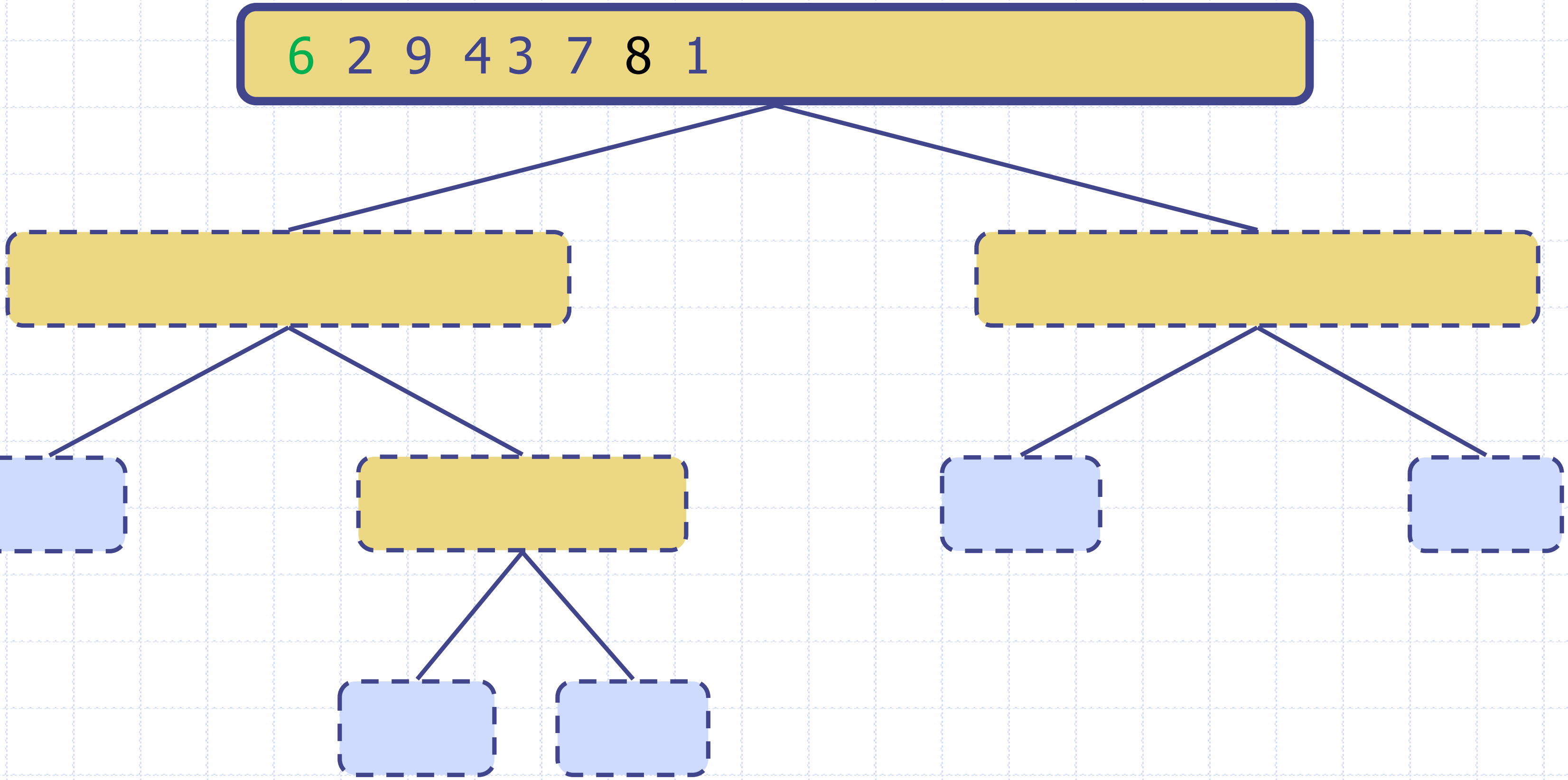
Quick-Sort Tree

- 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 subarrays of size 0 or 1



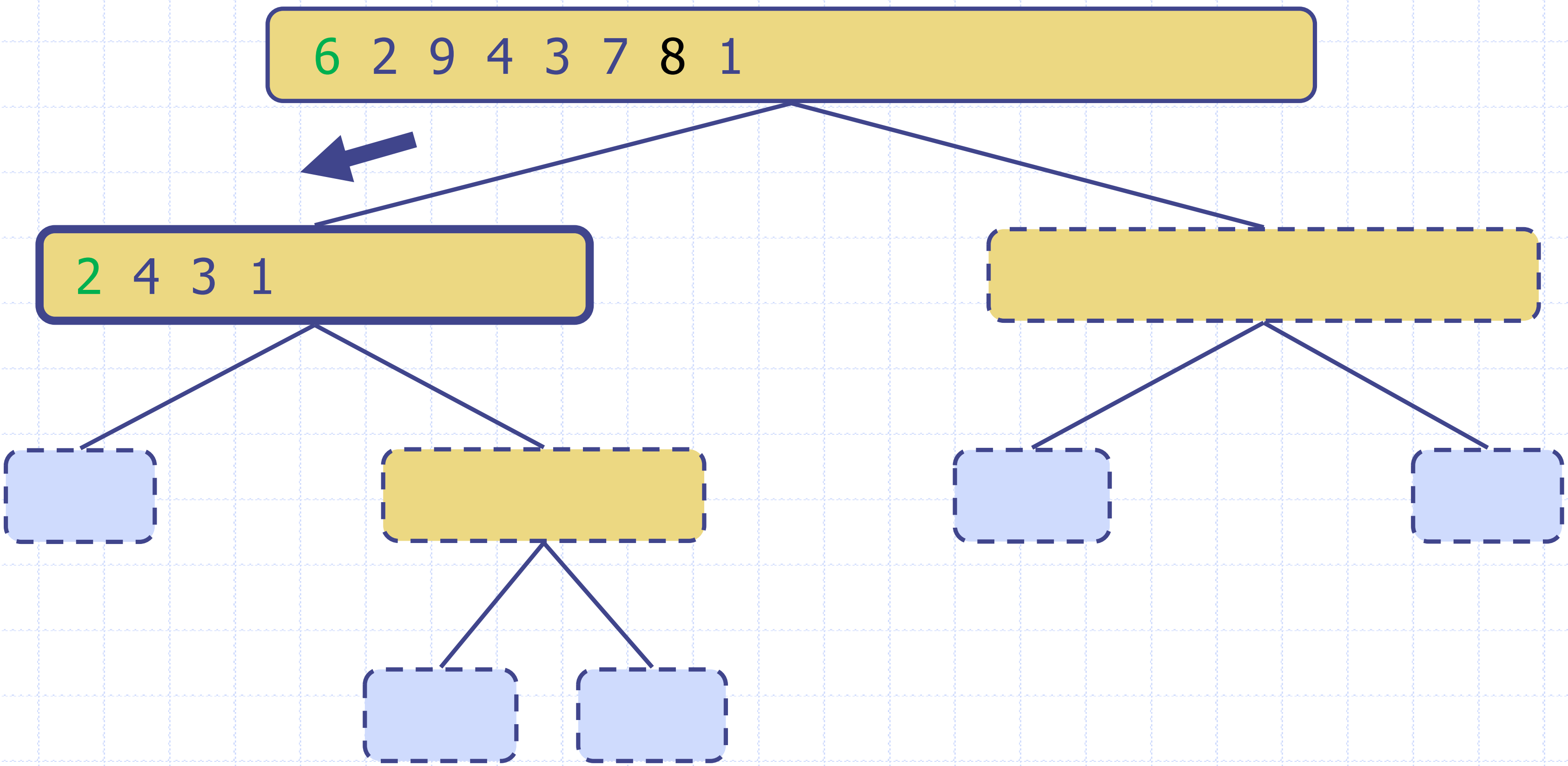
Execution Example

- Pivot selection



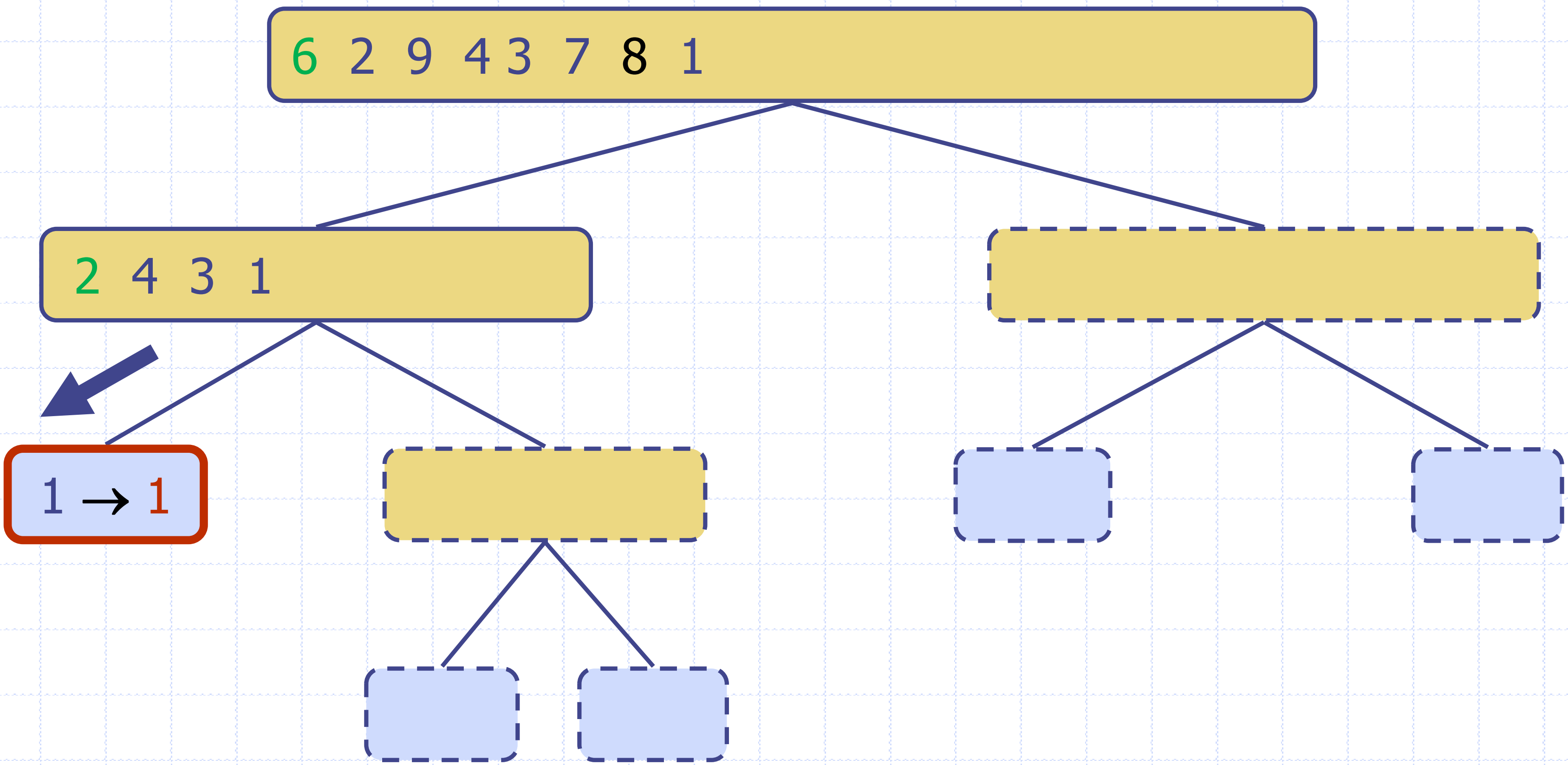
Execution Example (cont.)

- Partition, recursive call, pivot selection



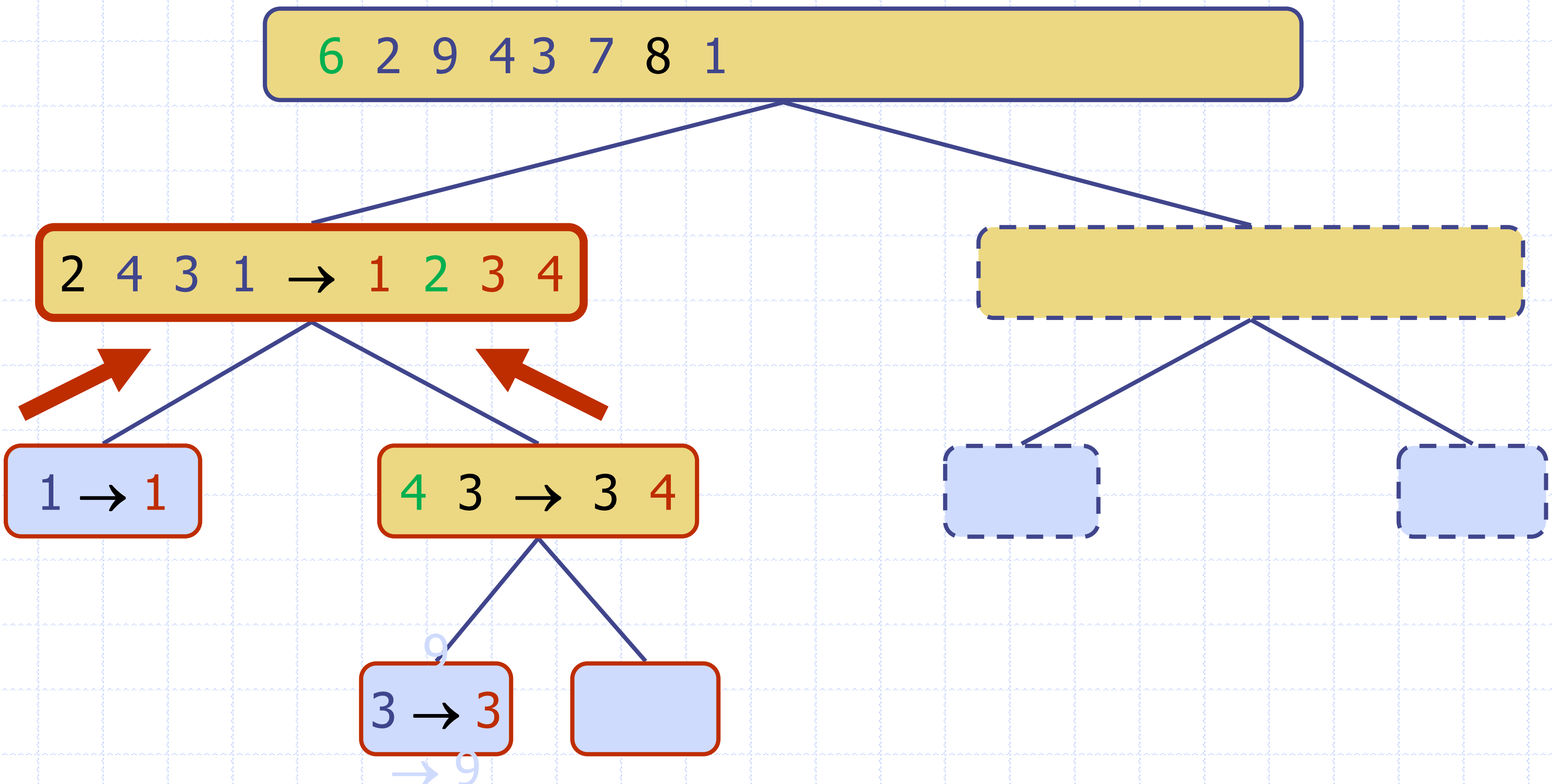
Execution Example (cont.)

- Partition, recursive call, base case



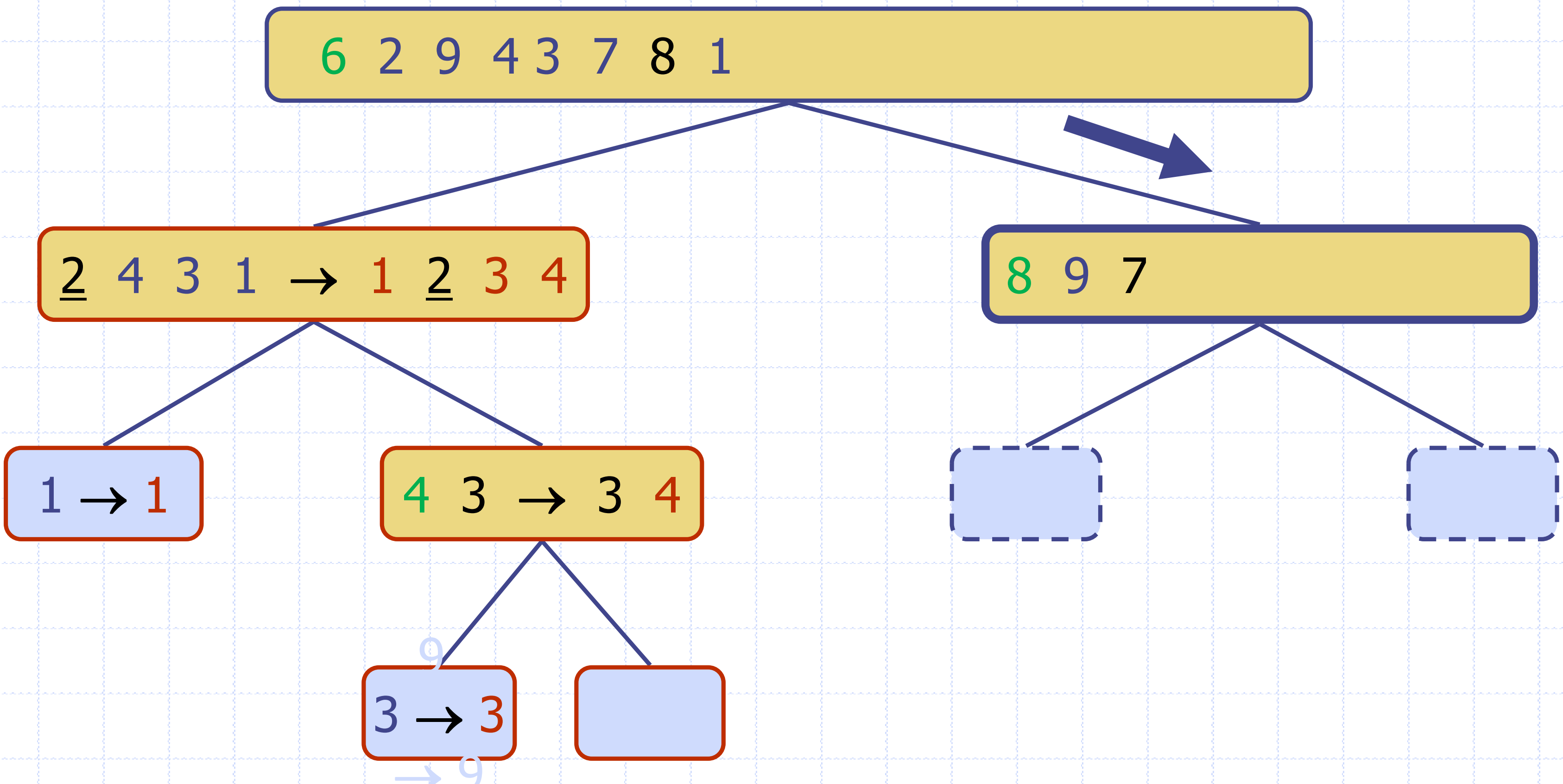
Execution Example (cont.)

- Recursive call, …, base case, join



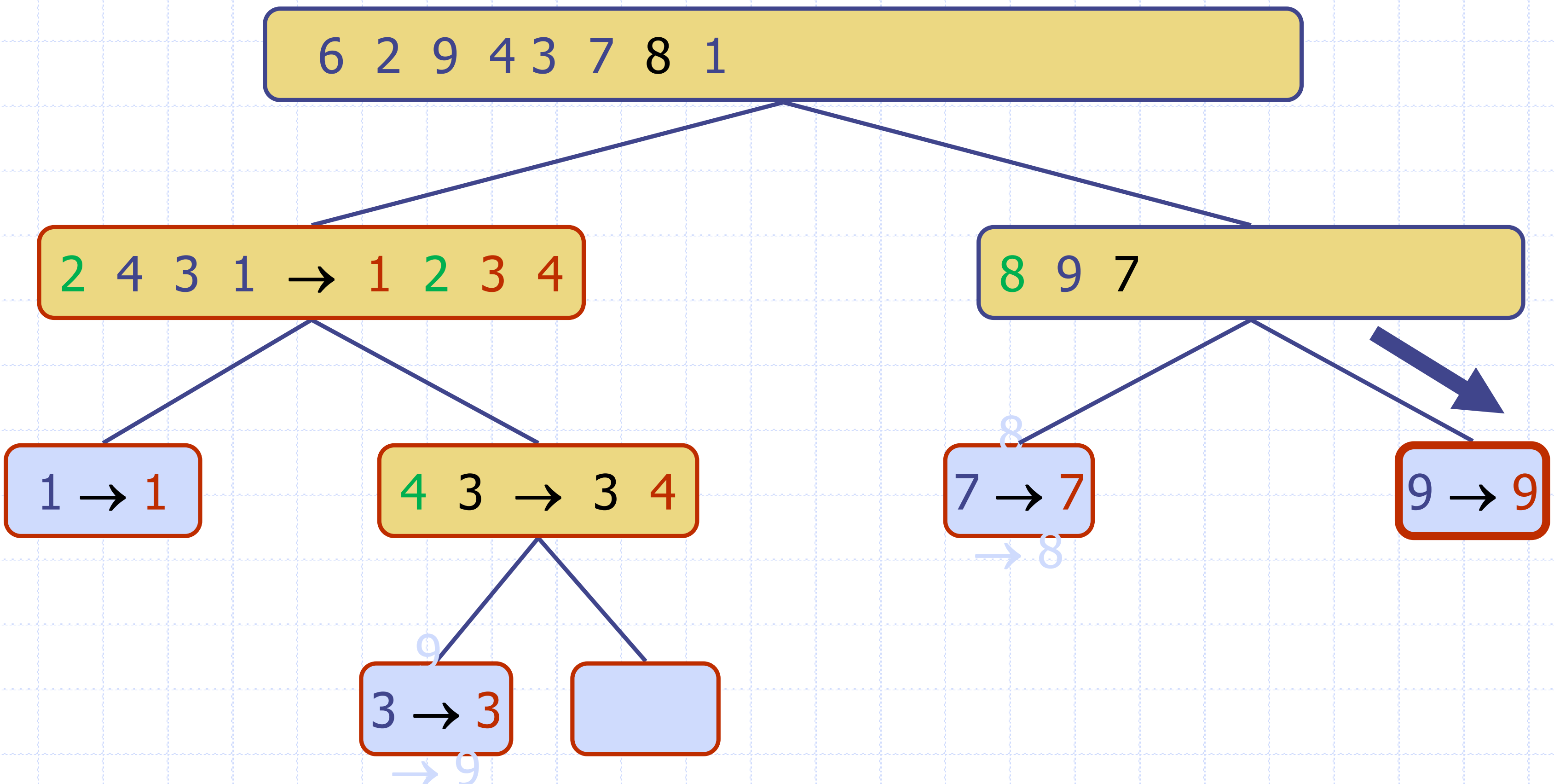
Execution Example (cont.)

- Recursive call, pivot selection



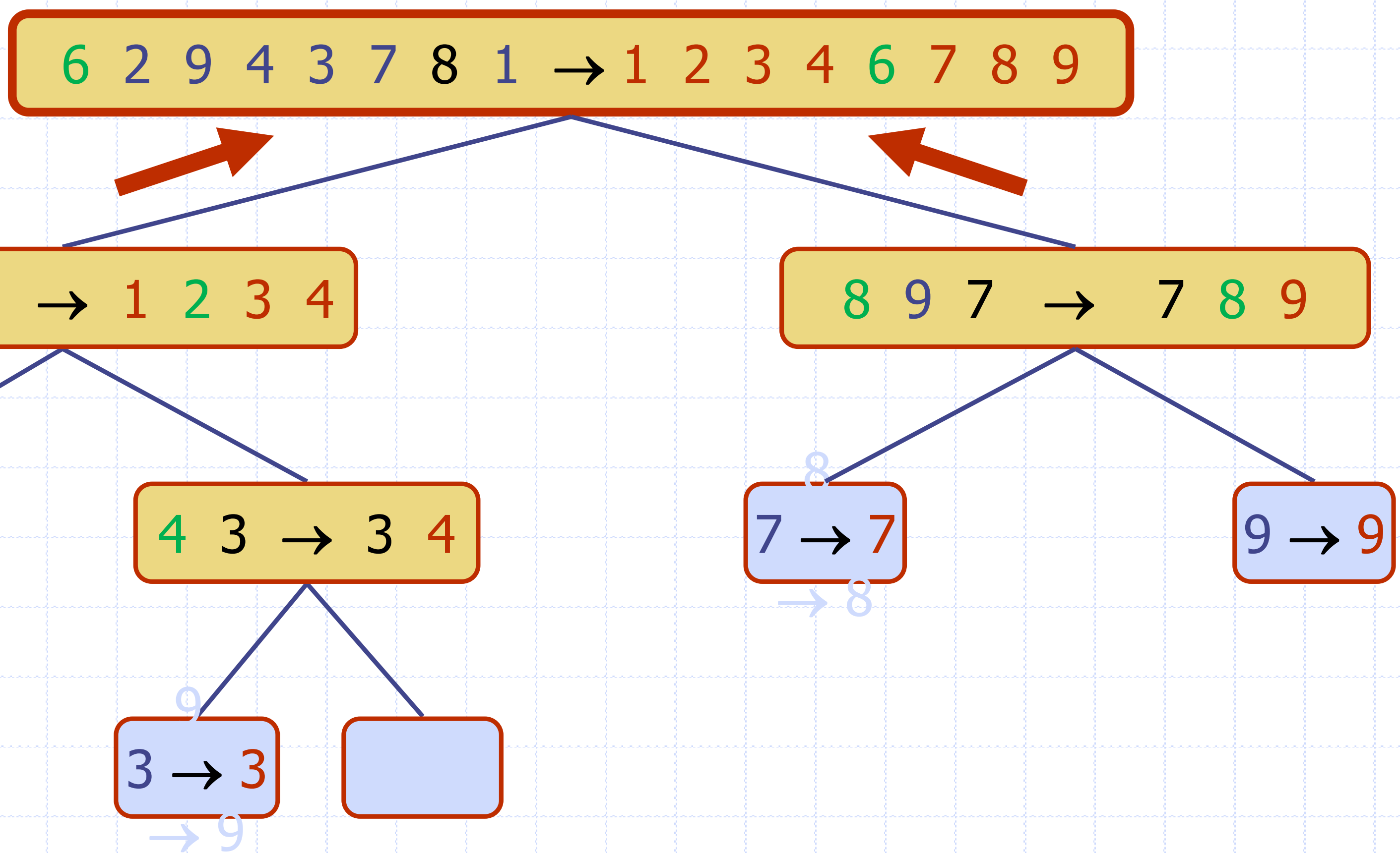
Execution Example (cont.)

- Partition, …, recursive call, base case



Execution Example (cont.)

- Join, join



Running Time

- The worst case for quick-sort occurs when the pivot is the minimum or maximum element
- One of L and G has size $n - 1$ and the other has size 0
- The worst-case running time of quick-sort is $O(n^2)$

- The average-case running time of quicksort is O(n log n).