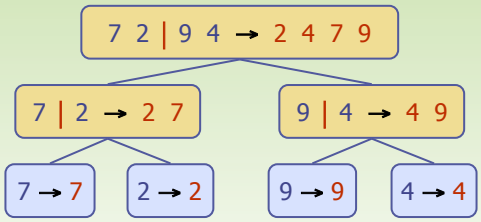
Divide & Conquer Sorting



Divide-and-Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide: divide the input data S in two disjoint subsets S_1 and S_2. Continue division until problem is 'tractable'.
  - Compute: solve the subproblems associated with S_1 and S_2
  - Conquer: combine the solutions for S_1 and S_2 into a solution for S
- Mergesort and Quicksort are two examples.

Merge-Sort

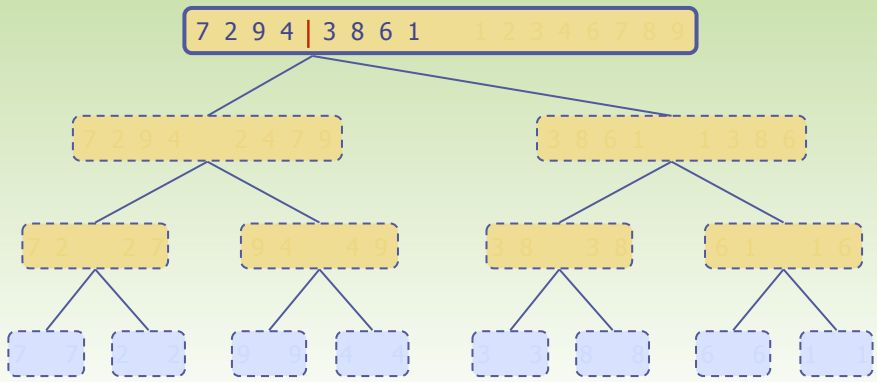
- Merge-sort on an input sequence S with n elements consists of three steps:
  - Divide: partition S into two sequences S_1 and S_2 of about $n/2$ elements each, until only 1 element remains in each S_1 and S_2
  - Compute: sort S_1 and S_2
  - Conquer: merge S_1 and S_2 into a unique sorted sequence

Merging Two Sorted Sequences

- The conquer step of merge-sort consists of merging two sorted sequences A and B into a sorted sequence S containing all of the elements of A and B
- Merging two sorted sequences, each with $n/2$ elements takes $O(n)$ time. Why?
  - Each element must be 'scanned' before being merged!

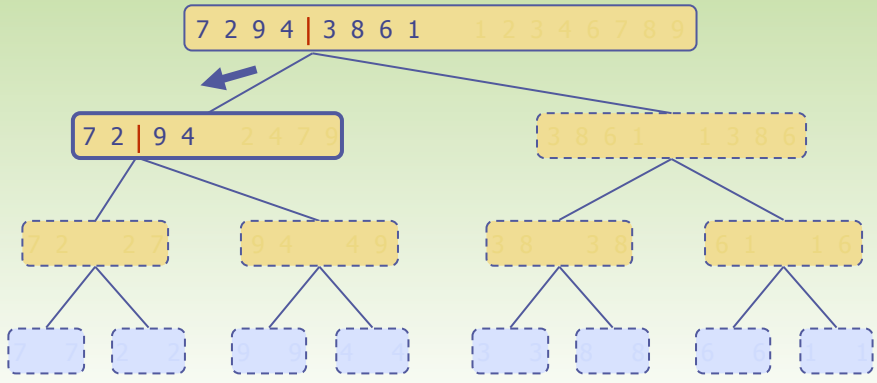
Execution Example

- Partition



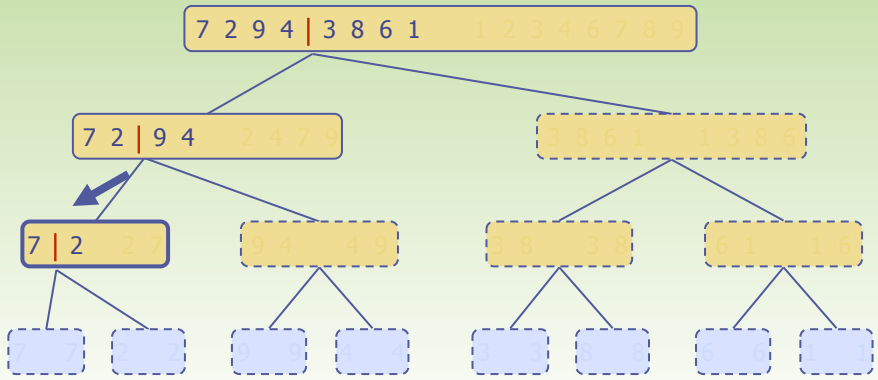
Execution Example (cont.)

- Continue partition until problem size is 1



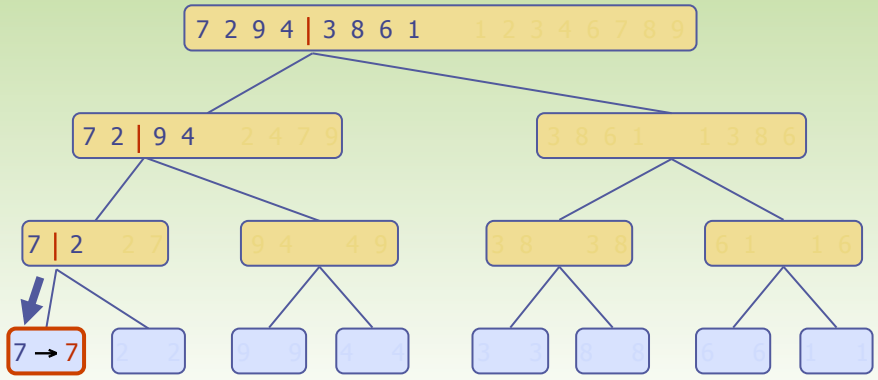
Execution Example (cont.)

- Continue partition until problem size is 1



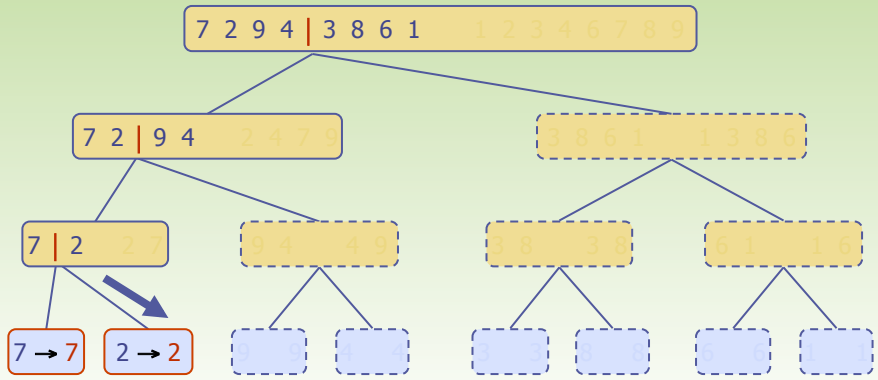
Execution Example (cont.)

- 'Base' case of 1!



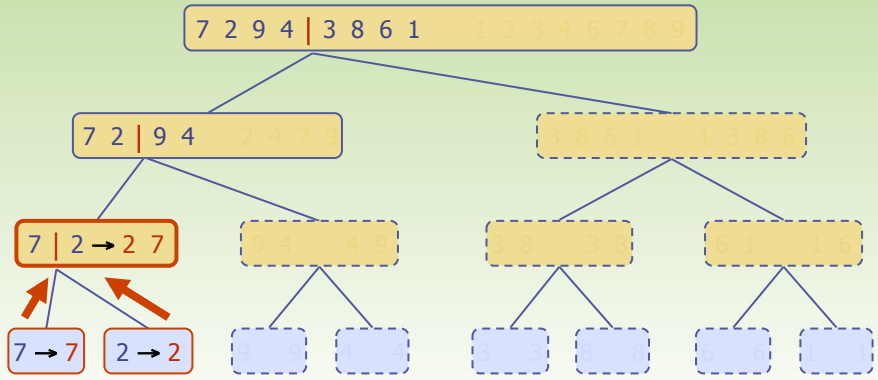
Execution Example (cont.)

- Here, too, base case of 1!



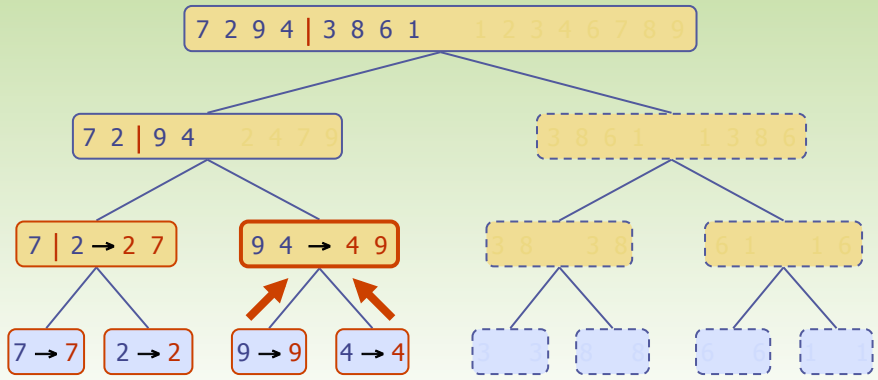
Execution Example (cont.)

- Merge



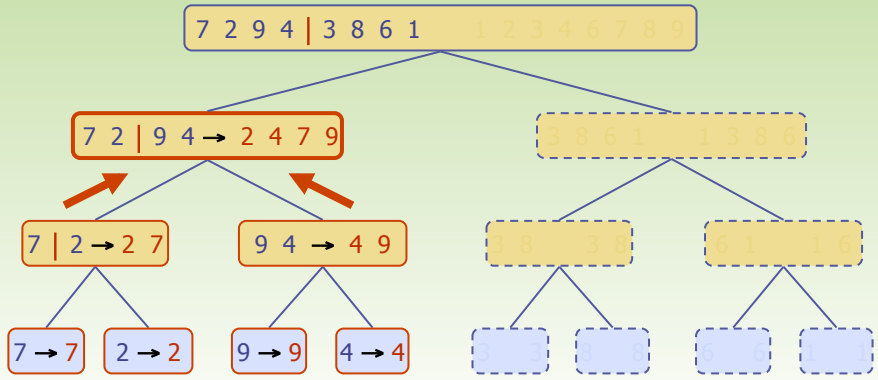
Execution Example (cont.)

- Recursive call, …, base case, merge



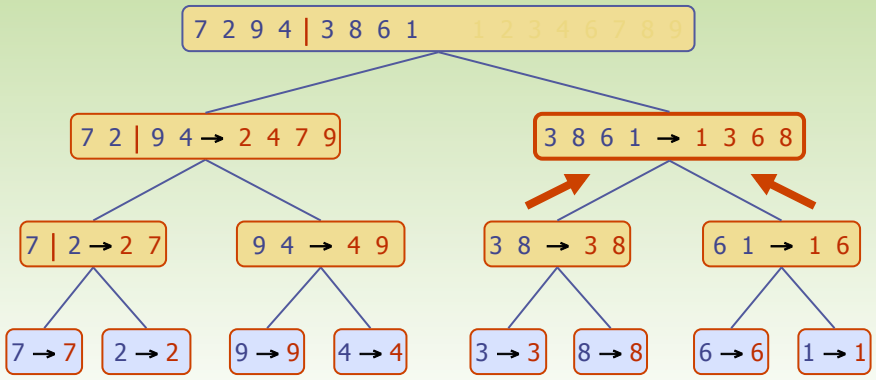
Execution Example (cont.)

- Merge



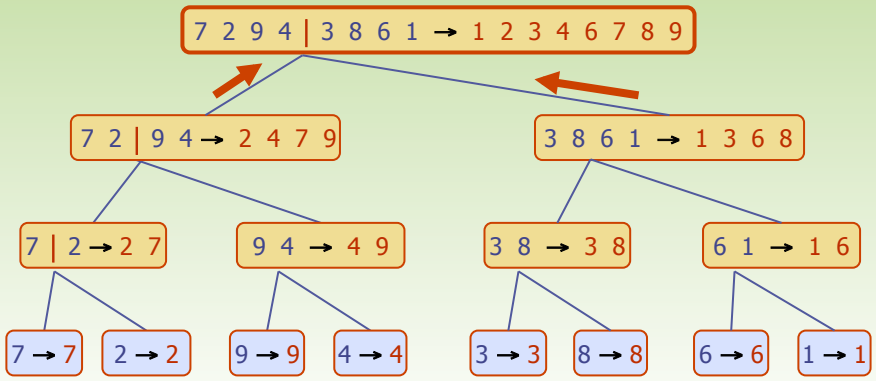
Execution Example (cont.)

- Recursive call, …, merge, merge



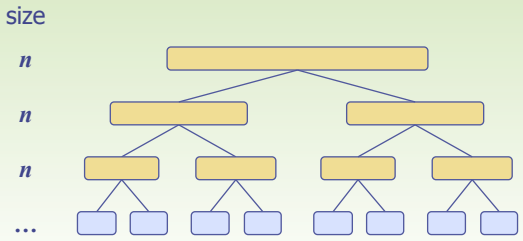
Execution Example (cont.)

- Merge



Analysis of Merge-Sort

- The "height" h of the merge-sort tree is $O(\log_2 n)$
  - This is a consequence of the dividing the problem in 2 at each step,
- The overall amount of work done at each merge i is $O(n)$
- Thus, the total running time of merge-sort is $O(n \log n)$



Running times in practice

- Home PC executes 10^8 comparisons/sec
- Supercomputer does 10^{12} comps/sec.

Insertion Sort (N^2)

Computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 mins
instant	instant	instant

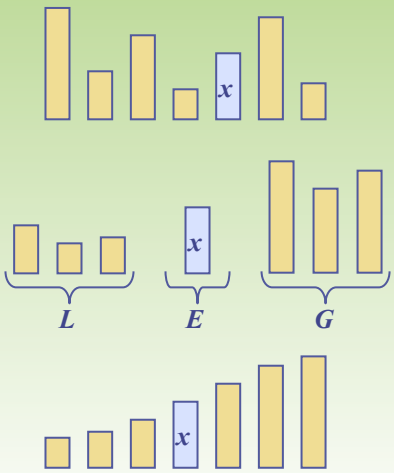
- Lesson 1: Good algorithms are better than a supercomputer!

Introducing Quicksort

- Mergesort
  - Sorts "at the end"
  - Partition is left/right
  - i.e. merge is sorting step
- Quicksort
  - Sorts "at beginning"
  - Partition is < or >
  - i.e. partition is sorting step!

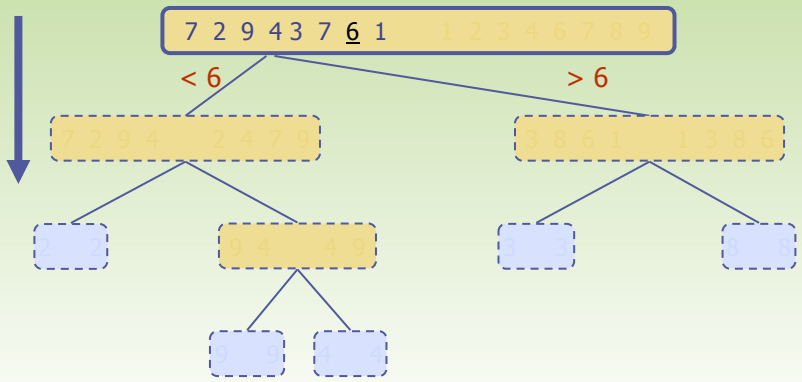
Quick-Sort

- Quick-sort is a 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 x
    - G elements greater than x
  - Re-apply: above on L and G
  - Conquer: join L, E and G



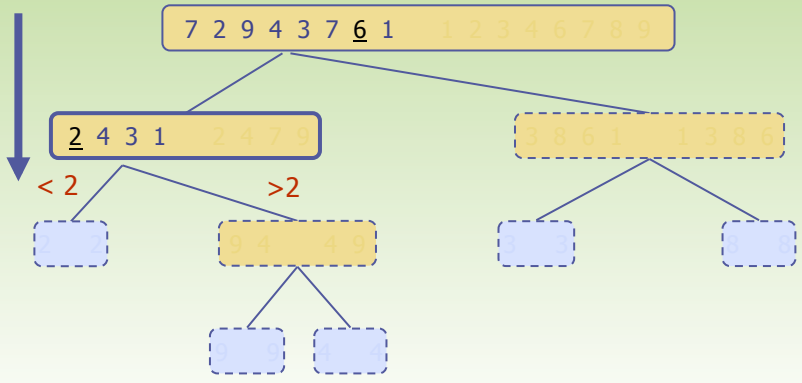
Execution Example

- Pivot selection



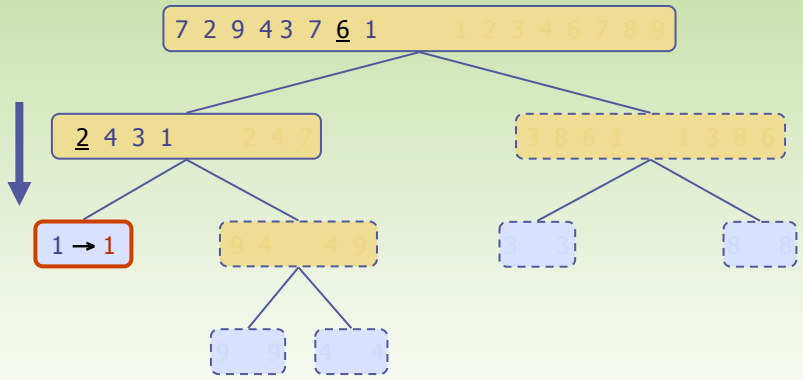
Execution Example (cont.)

- Partition, repeat pivot selection.



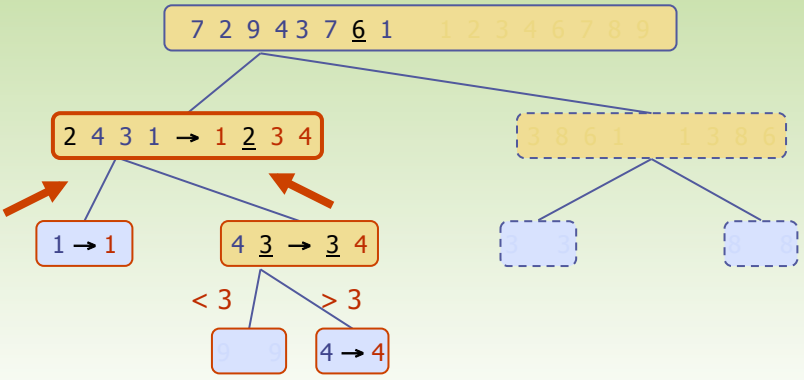
Execution Example (cont.)

- Partition, select pivot, base case



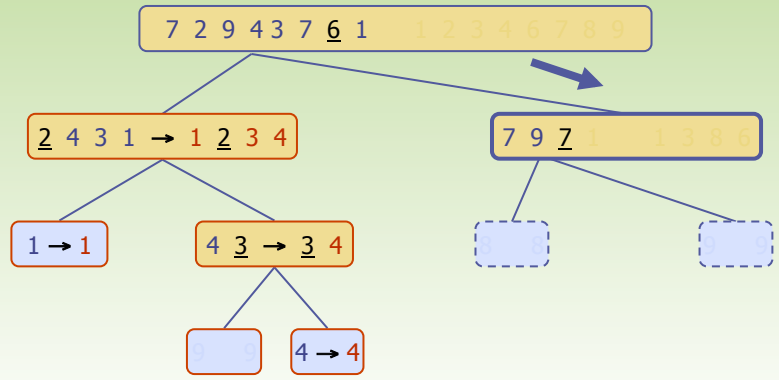
Execution Example (cont.)

- partition, select pivot, smallest case, join



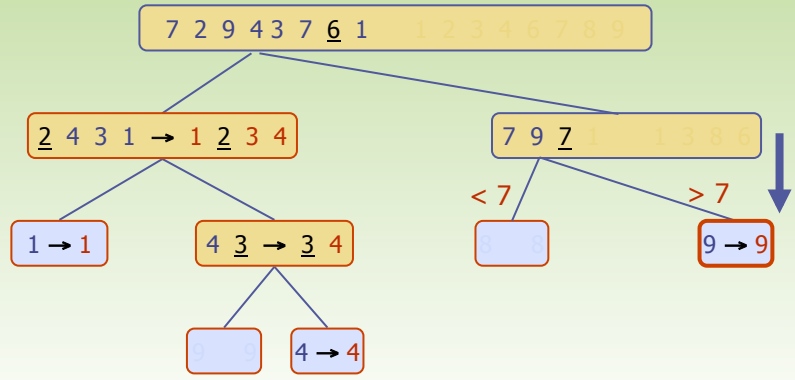
Execution Example (cont.)

- Partition, pivot selection



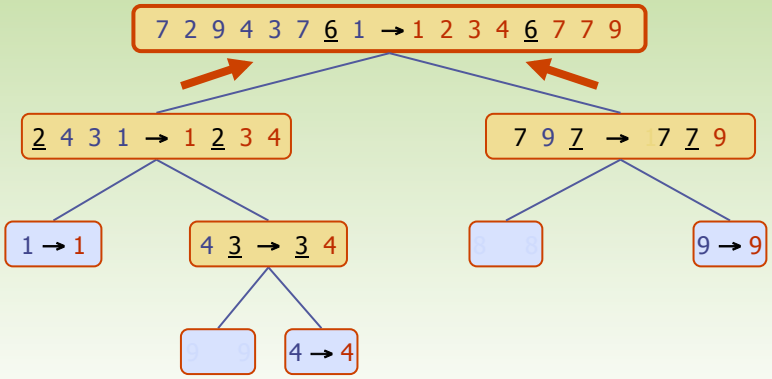
Execution Example (cont.)

- Partition, …, select pivot, smallest case



Execution Example (cont.)

- Join, join

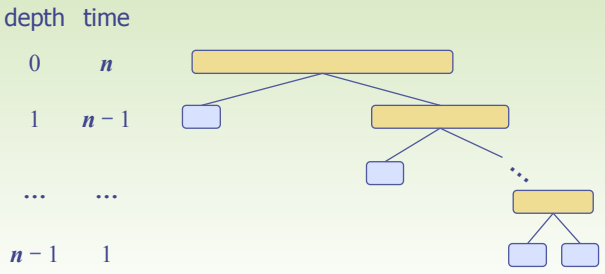


Worst-case Running Time

- The worst case for quick-sort occurs when the pivot is the unique minimum or maximum element
- One of L and G has size $n - 1$ and the other has size 0
- The running time is proportional to the sum

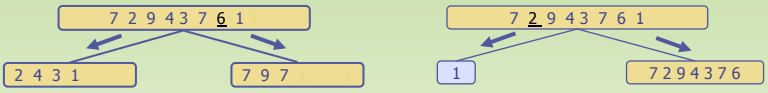
$$n + (n - 1) + \ldots + 2 + 1$$

- Thus, the worst-case running time of quick-sort is $O(n^2)$



Expected Running Time

- Consider a call of quick-sort on a sequence of size s
  - **Good case:** the sizes of L and G are roughly the same
  - **Bad case:** the sizes of L and G differ greatly



Good case **Bad case**

- A case is good with probability $1/2$
  - 1/2 of the possible pivots cause good calls:

Running times in practice

- Home PC executes 10^8 comparisons/sec
- Supercomputer does 10^{12} comps/sec.

Insertion Sort (N^2)

Computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 mins
instant	instant	instant

Quicksort (N log N)

thousand	million	billion
instant	0.3 sec	6 mins
instant	instant	instant

- Lesson 2: Great algorithms better than good ones!