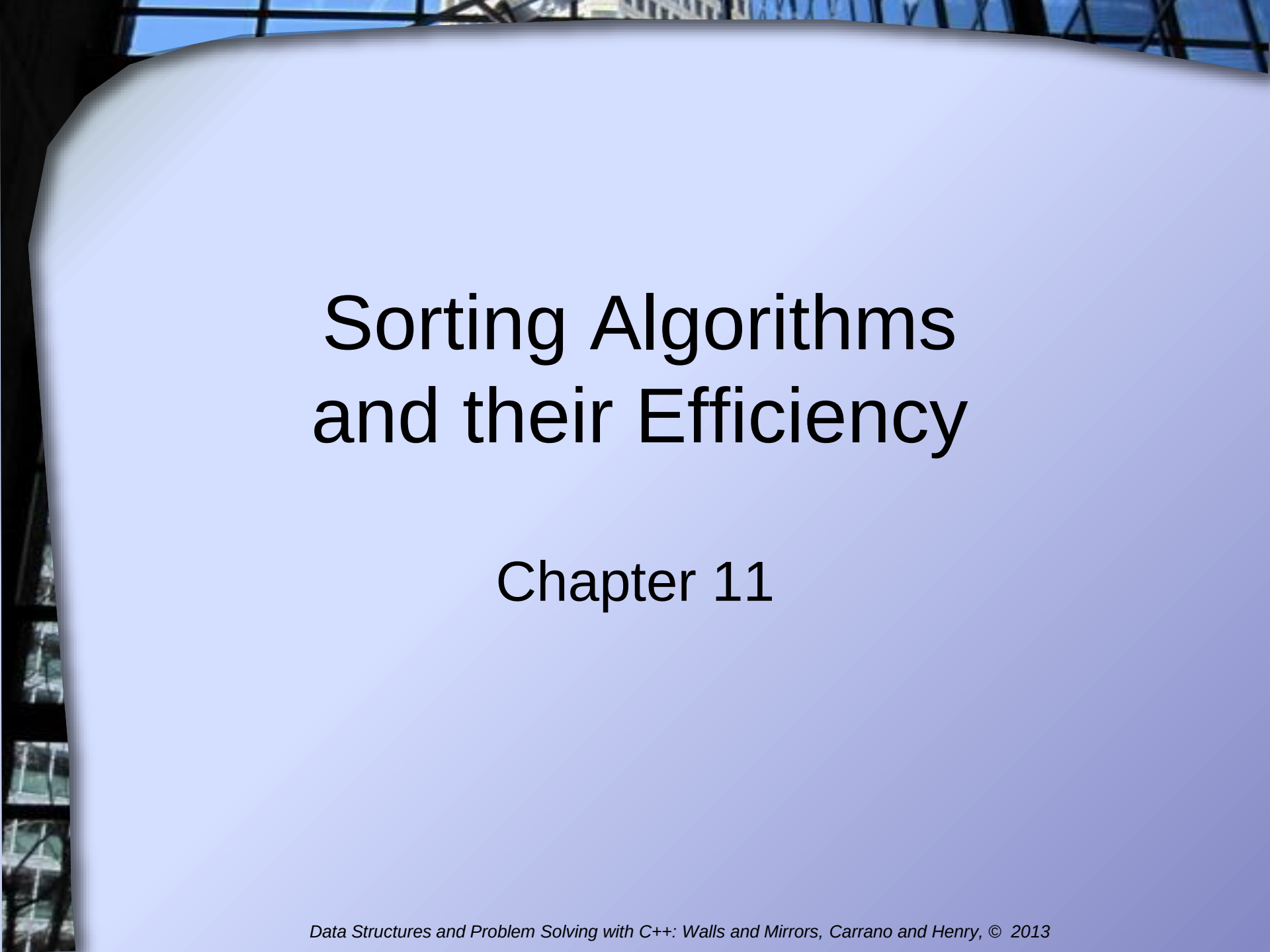


Sorting Algorithms and their Efficiency

Chapter 11

Contents

- Basic Sorting Algorithms
- Faster Sorting Algorithms
- A Comparison of Sorting Algorithms

Basic Sorting Algorithms

- Sorting is:
  - A process
  - It organizes a collection of data
  - Organized into ascending/descending order
- Internal: data fits in memory
- External: data must reside on secondary storage
- Sort key: data item which determines order

The Selection Sort

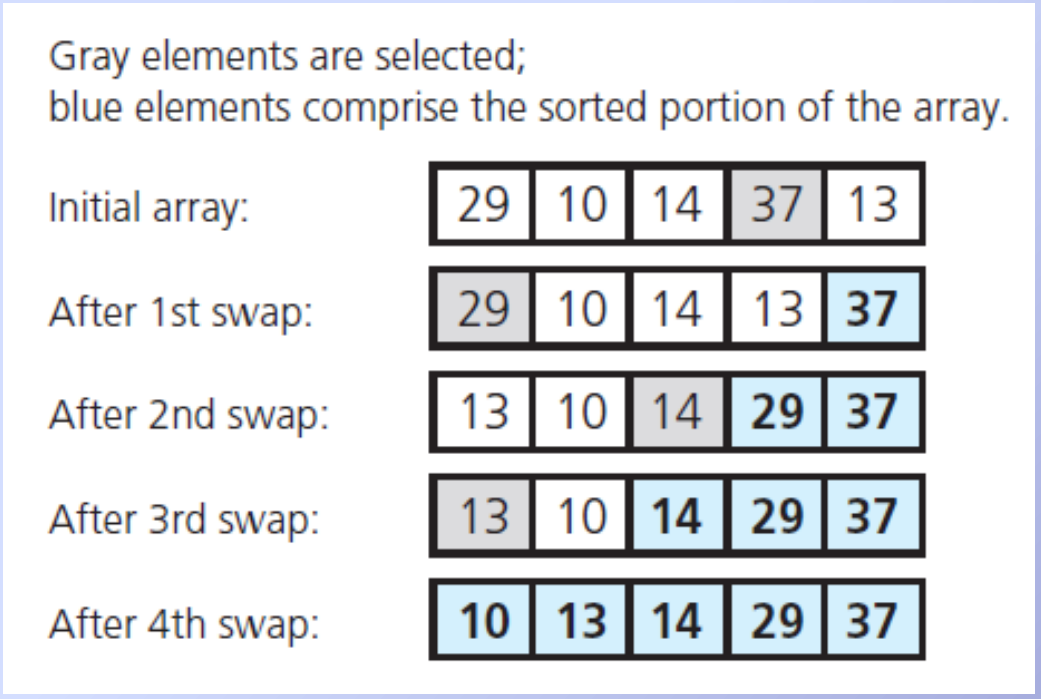


FIGURE 11-1 A selection sort of an array of five integers

The Selection Sort

- View implementation of the selection sort, Listing 11-1
- Analysis
  - This is an O (n^2) algorithm
- If sorting a very large arra
  algorithm probably too ine

.htm code listing files must be in the same folder as the .ppt files for these links to work

The Bubble Sort

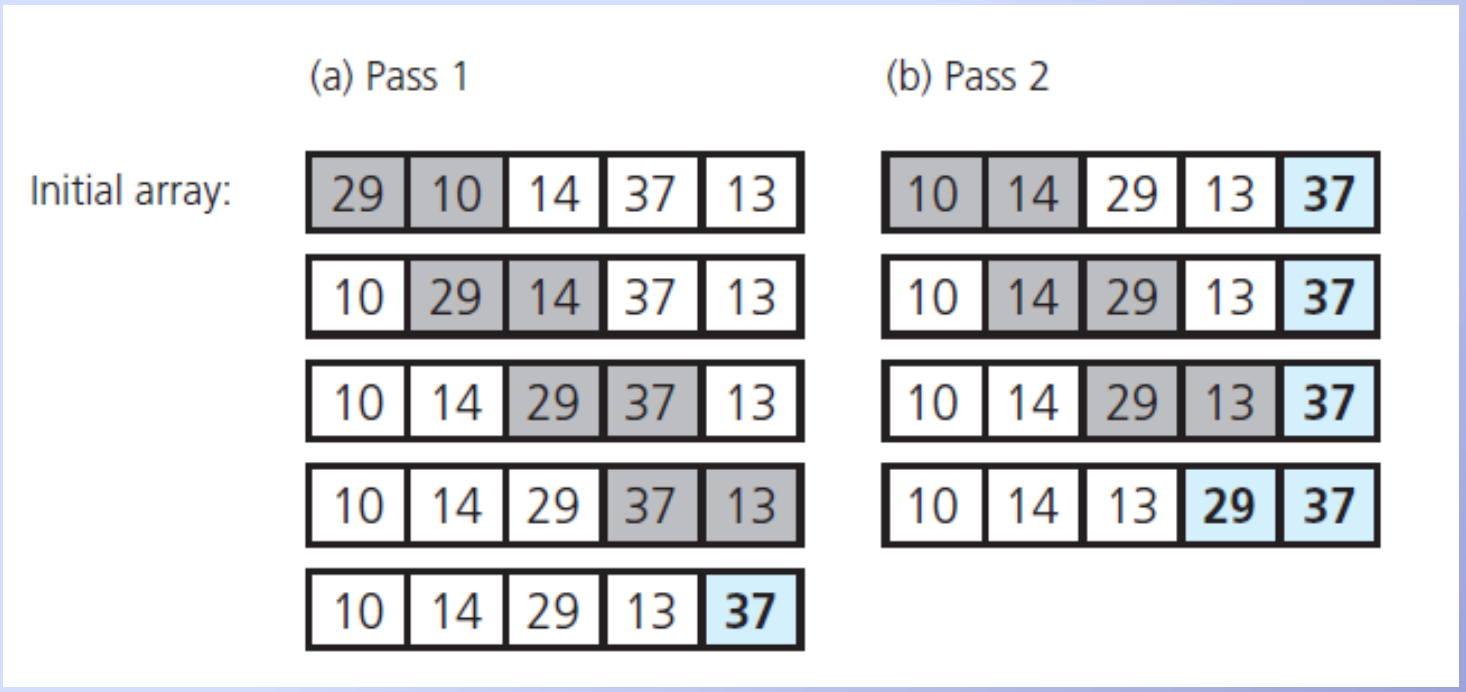


FIGURE 11-2 The first two passes of a bubble sort of an array of five integers

The Bubble Sort

- View an implementation of the bubble sort, Listing 11-2
- Analysis
  - Best case, O(n) algorithm
  - Worst case, $O(n^2)$ algorithm
- Again, a poor choice for large amounts of data

The Insertion Sort

- Take each item from unsorted region, insert into its correct order in sorted region

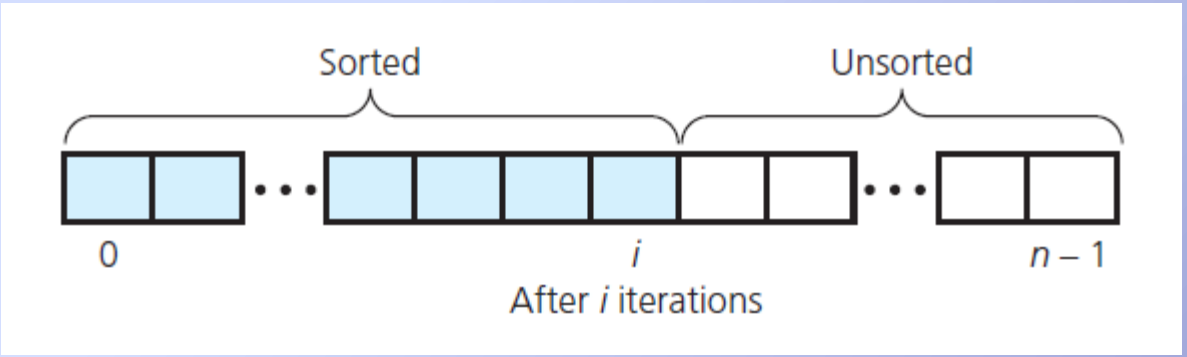


FIGURE 11-3 An insertion sort partitions the array into two regions

The Insertion Sort

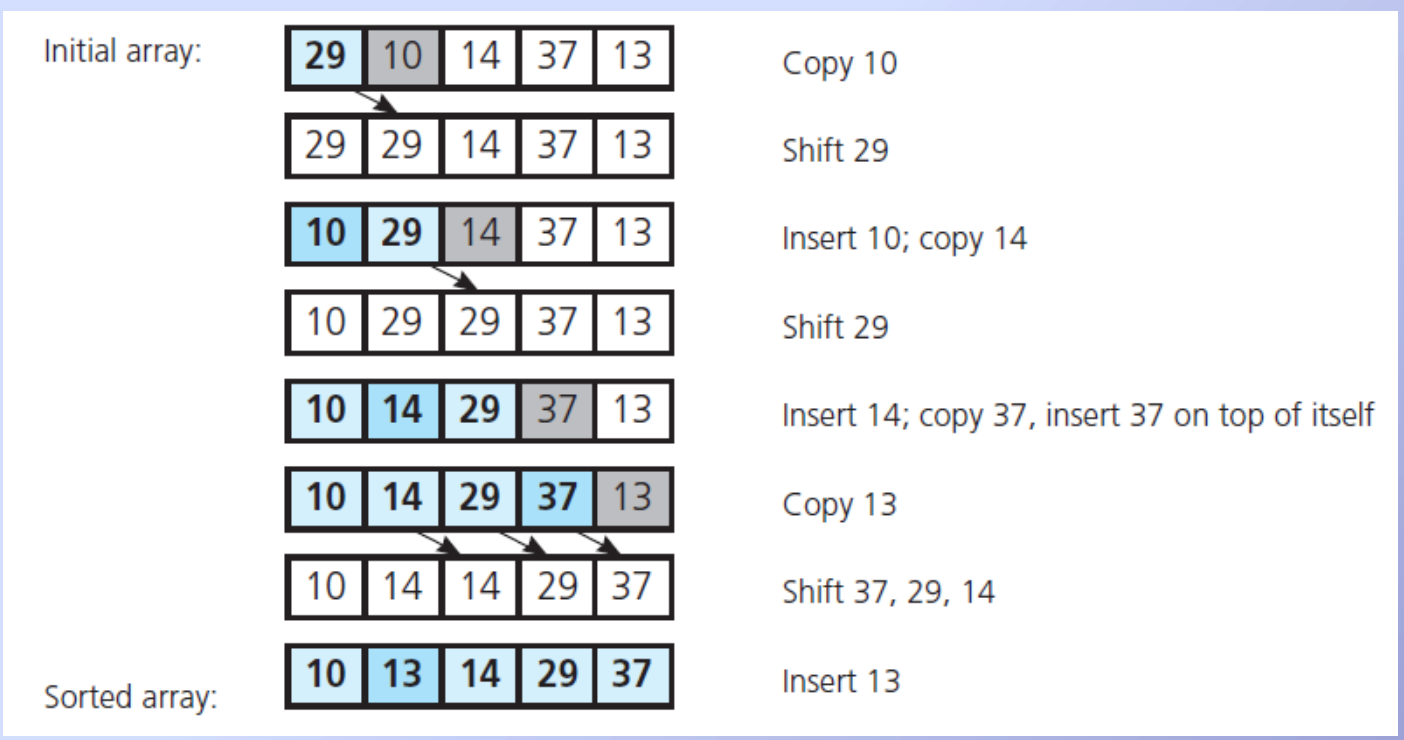


FIGURE 11-4 An insertion sort of an array of five integers

The Insertion Sort

- View implantation of insertion sort, Listing 11-3
- Analysis
  - An algorithm of order $O(n^2)$
  - Best case $O(n)$
- Appropriate for 25 or less data items

The Merge Sort

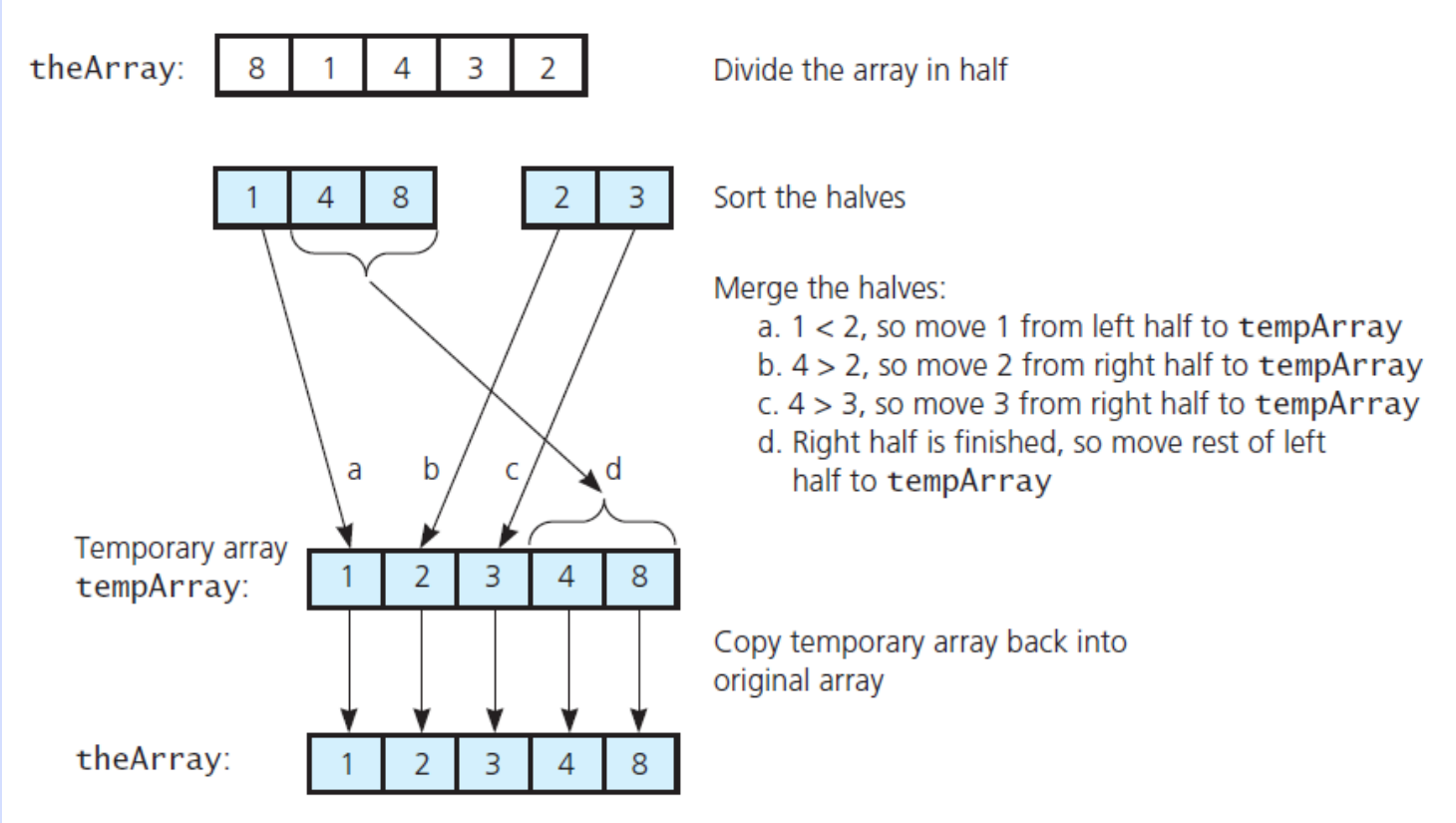


FIGURE 11-5 A merge sort with an auxiliary temporary array

The Merge Sort

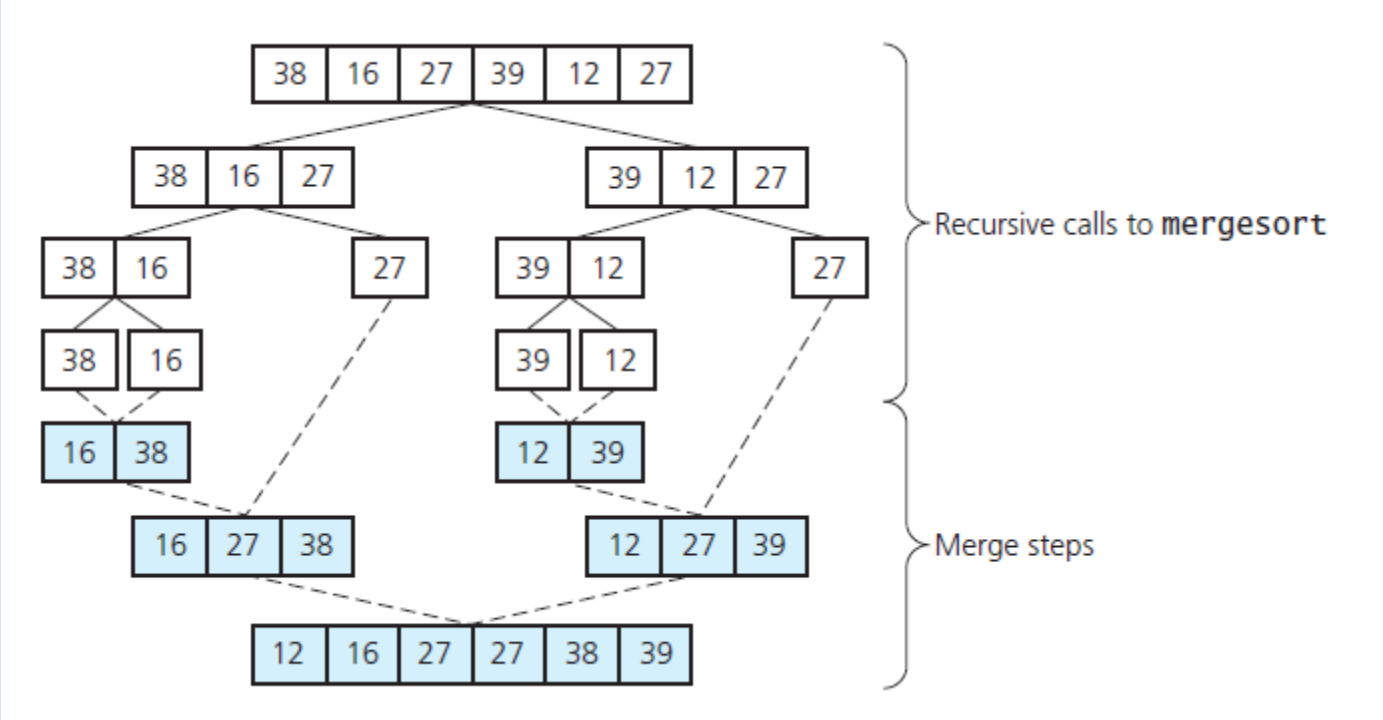
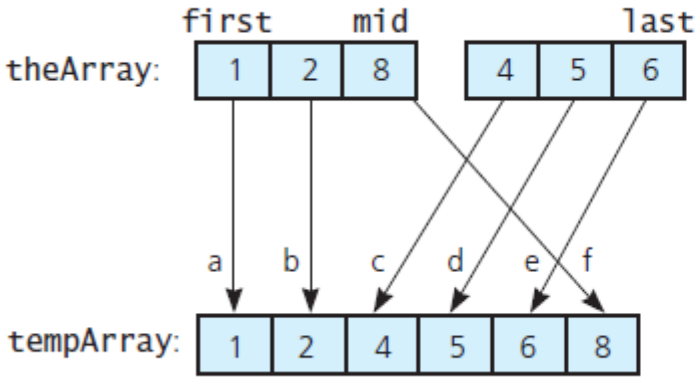


FIGURE 11-6 A merge sort of an array of six integers

The Merge Sort

- View implementation of the merge sort, Listing 11-4
- Analysis
  - Merge sort is of order $O(n \times \log n)$
  - This is significantly faster than $O(n^2)$

The Merge Sort



Merge the halves:

a. 1 < 4, so move 1 from `theArray[first..mid]` to `tempArray`
b. 2 < 4, so move 2 from `theArray[first..mid]` to `tempArray`
c. 8 > 4, so move 4 from `theArray[mid+1..last]` to `tempArray`
d. 8 > 5, so move 5 from `theArray[mid+1..last]` to `tempArray`
e. 8 > 6, so move 6 from `theArray[mid+1..last]` to `tempArray`
f. `theArray[mid+1..last]` is finished, so move 8 to `tempArray`

FIGURE 11-7 A worst-case instance of the merge step in a merge sort

The Merge Sort

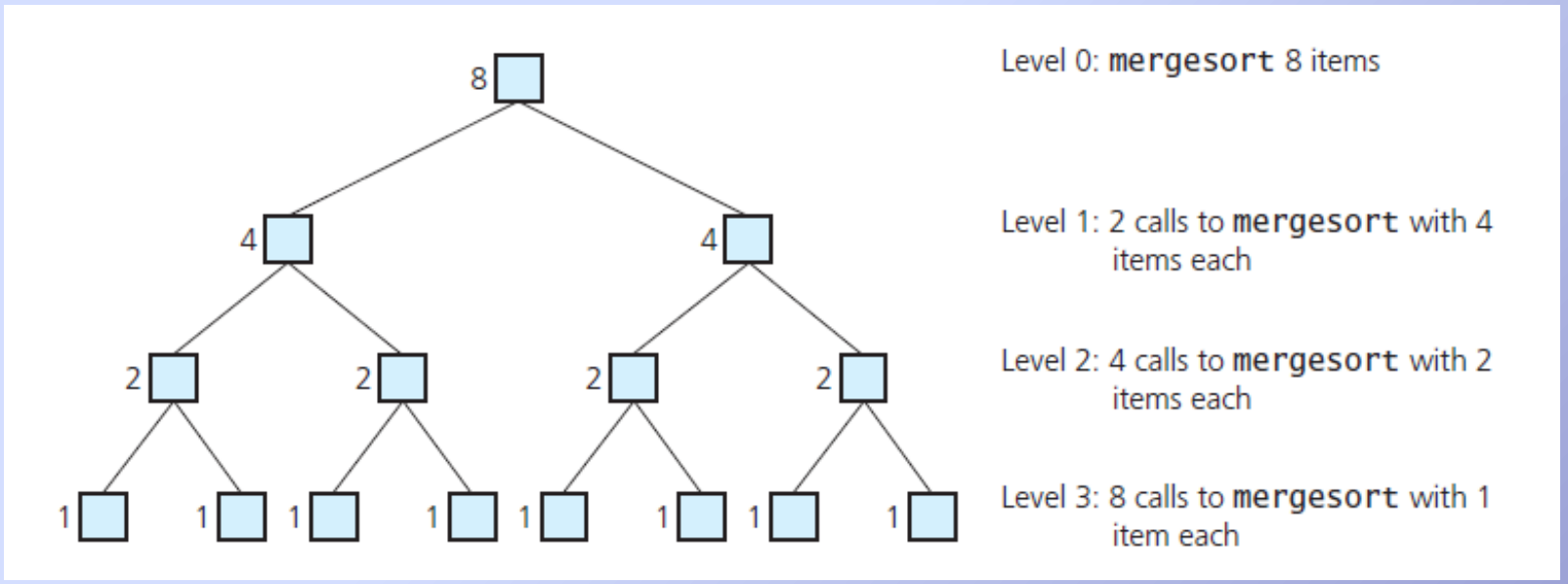


FIGURE 11-8 Levels of recursive calls to **`mergeSort`**, given an array of eight items

The Quick Sort

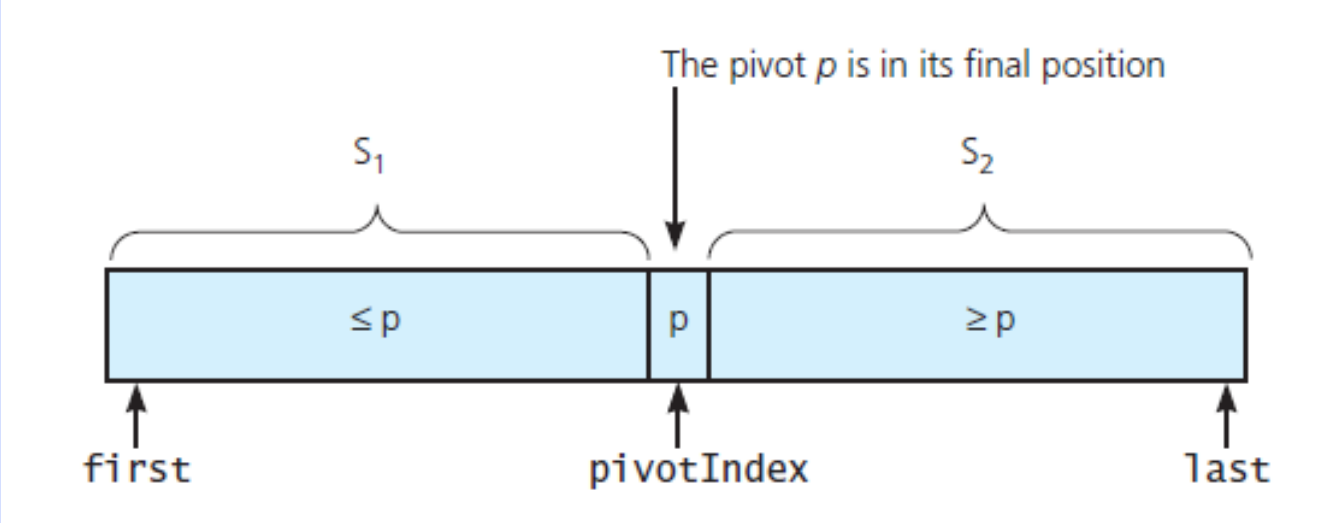


FIGURE 11-9 A partition about a pivot

The Quick Sort

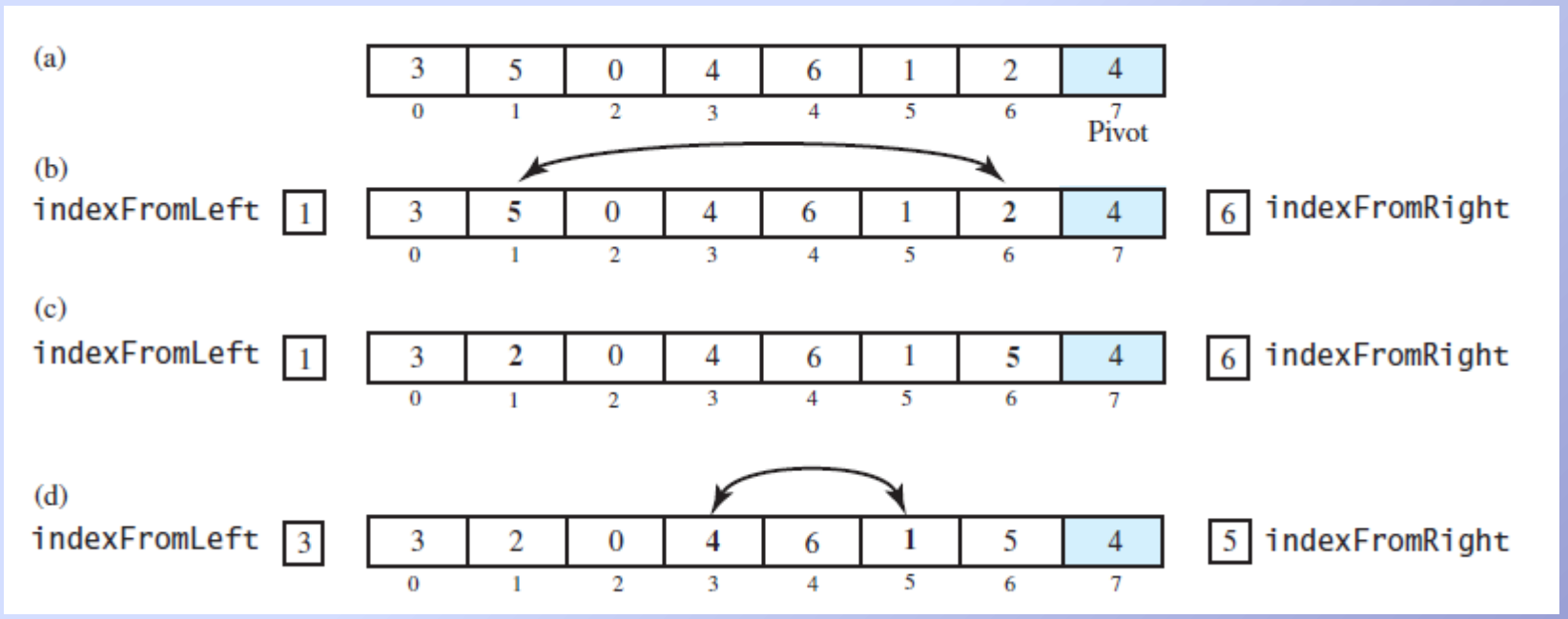


FIGURE 11-10 Partitioning of array during quick sort

The Quick Sort

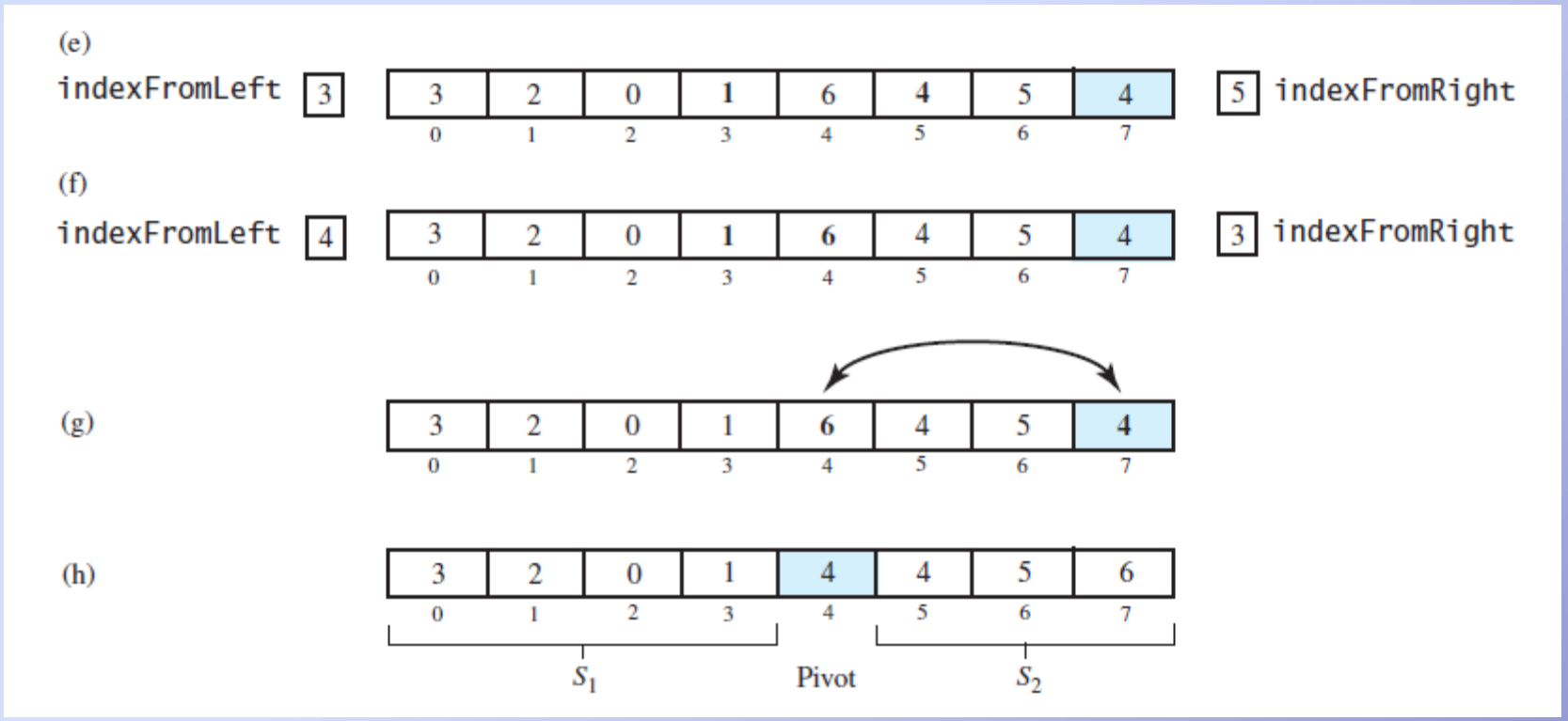


FIGURE 11-10 Partitioning of array during quick sort

The Quick Sort

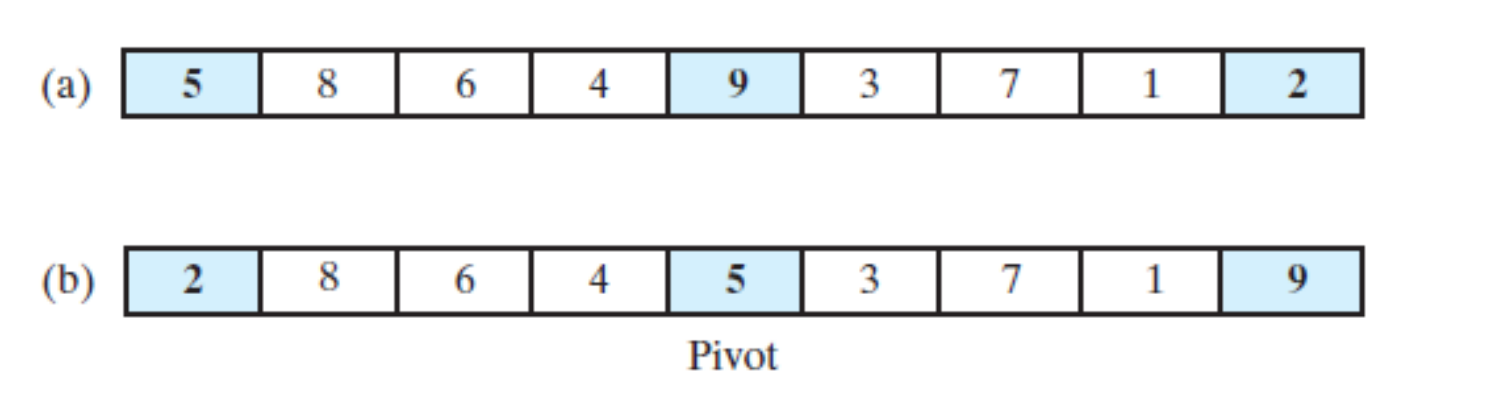


FIGURE 11-11 Median-of-three pivot selection: (a) The original array; (b) the array with its first, middle, and last entries sorted

The Quick Sort

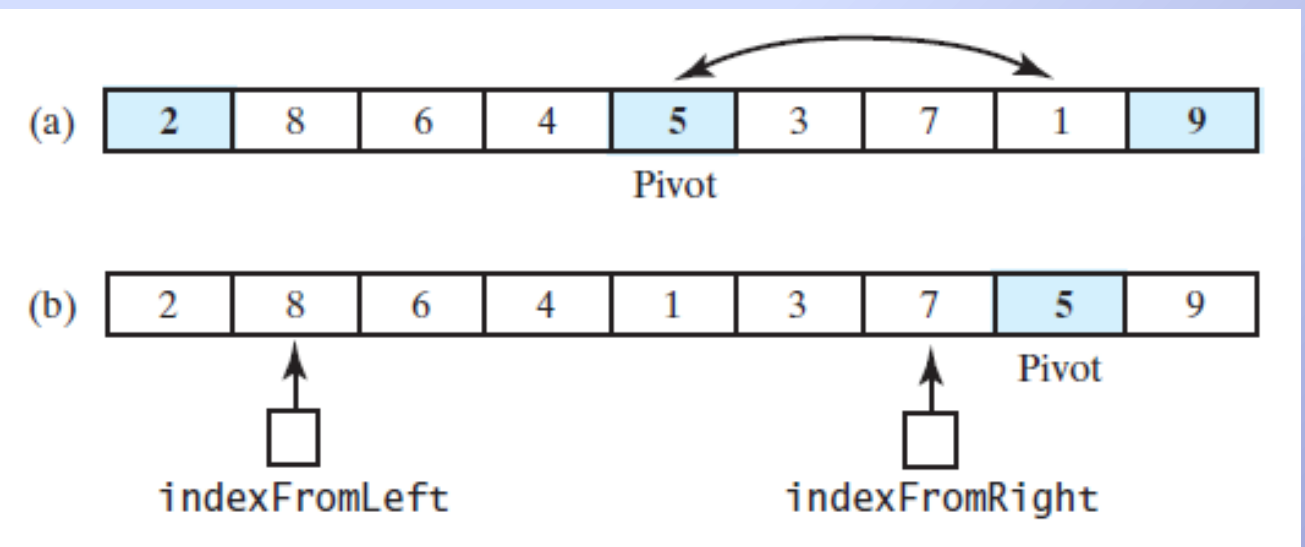


FIGURE 11-12 (a) The array with its first, middle, and last entries sorted; (b) the array after positioning the pivot and just before partitioning

The Quick Sort

- Note function that performs a quick sort, Listing 11-5
- Analysis
  - Worst case $O(n^2)$
  - Average case $O(n \times \log n)$
  - Does not require extra memory like merge sort

The Quick Sort

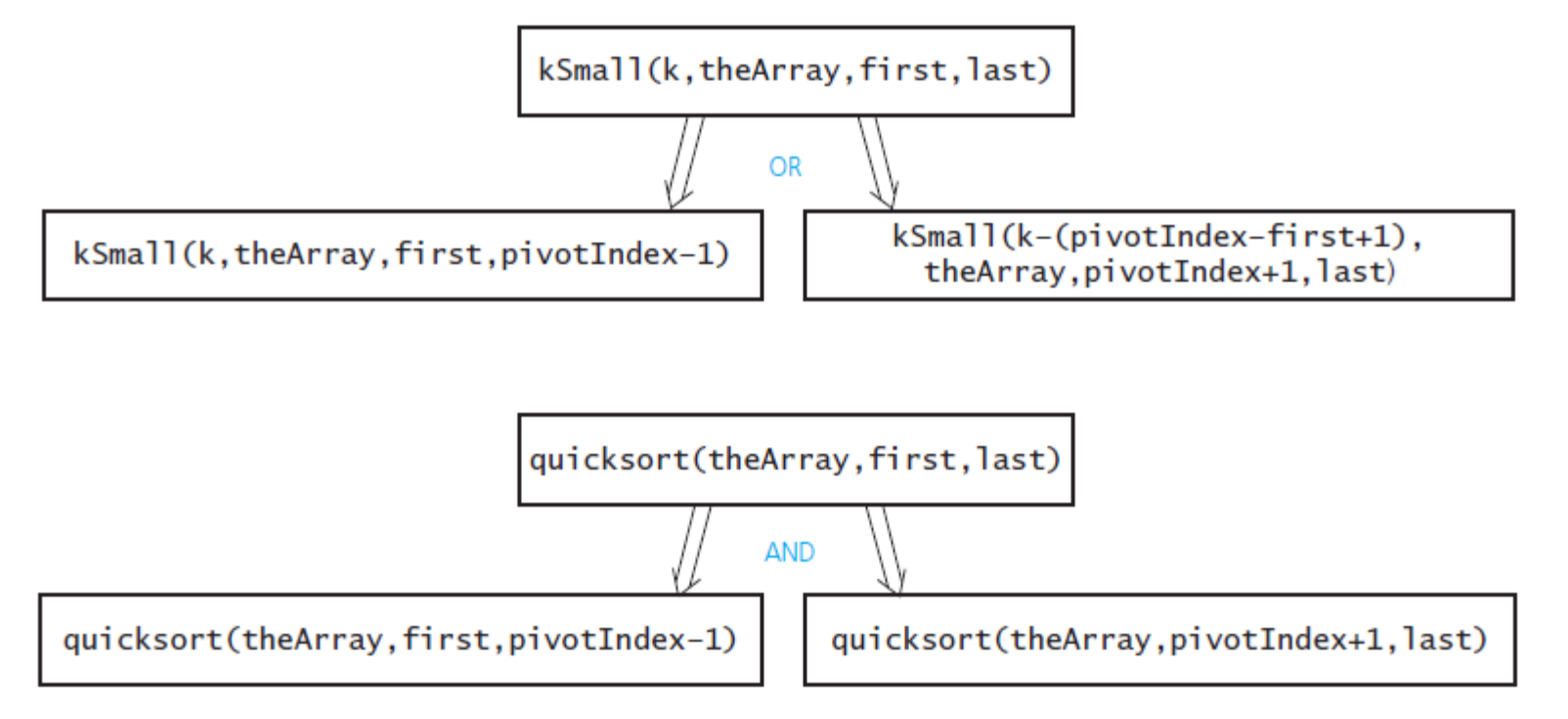


FIGURE 11-13 **`kSmall`** versus **`quickSort`**

The Radix Sort

- Uses the idea of forming groups, then combining them to sort a collection of data.
- Consider collection of three letter groups

  ABC, XYZ, BWZ, AAC, RLT, JBX, RDT, KLT, AEO, TLJ
- Group strings by rightmost letter

  (AB**C**, AA**C**) (TL**J**) (AE**O**) (RL**T**, RD**T**, KL**T**) (JB**X**) (XY**Z**, BW**Z**)
- Combine groups

  ABC, AAC, TLJ, AEO, RLT, RDT, KLT, JBX, XYZ, BWZ

The Radix Sort

- Group strings by middle letter

  (A**A**C) (A **B** C, J **B** X) (R **D** T) (A **E** O) (T **L** J, R **L** T, K **L** T) (B **W** Z) (X **Y** Z)

- Combine groups

  AAC, ABC, JBX, RDT, AEO, TLJ, RLT, KLT, BWZ, XYZ

- Group by first letter, combine again

  (**A** AC, **A** BC, **A** EO) (**B** WZ) (**J** BX) (**K** LT) (**R** DT, **R** LT) (**T** LJ) (**X** YZ)

- Sorted strings

  AAC, ABC, AEO, BWZ, JBX, KLT, RDT, RLT, TLJ, XYZ

The Radix Sort

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150	Original integers
(1560, 2150) (1061) (0222) (0123, 0283) (2154, 0004)	Grouped by fourth digit
1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004	Combined
(0004) (0222, 0123) (2150, 2154) (1560, 1061) (0283)	Grouped by third digit
0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283	Combined
(0004, 1061) (0123, 2150, 2154) (0222, 0283) (1560)	Grouped by second digit
0004, 1061, 0123, 2150, 2154, 0222, 0283, 1560	Combined
(0004, 0123, 0222, 0283) (1061, 1560) (2150, 2154)	Grouped by first digit
0004, 0123, 0222, 0283, 1061, 1560, 2150, 2154	Combined (sorted)

FIGURE 11-14 A radix sort of eight integers

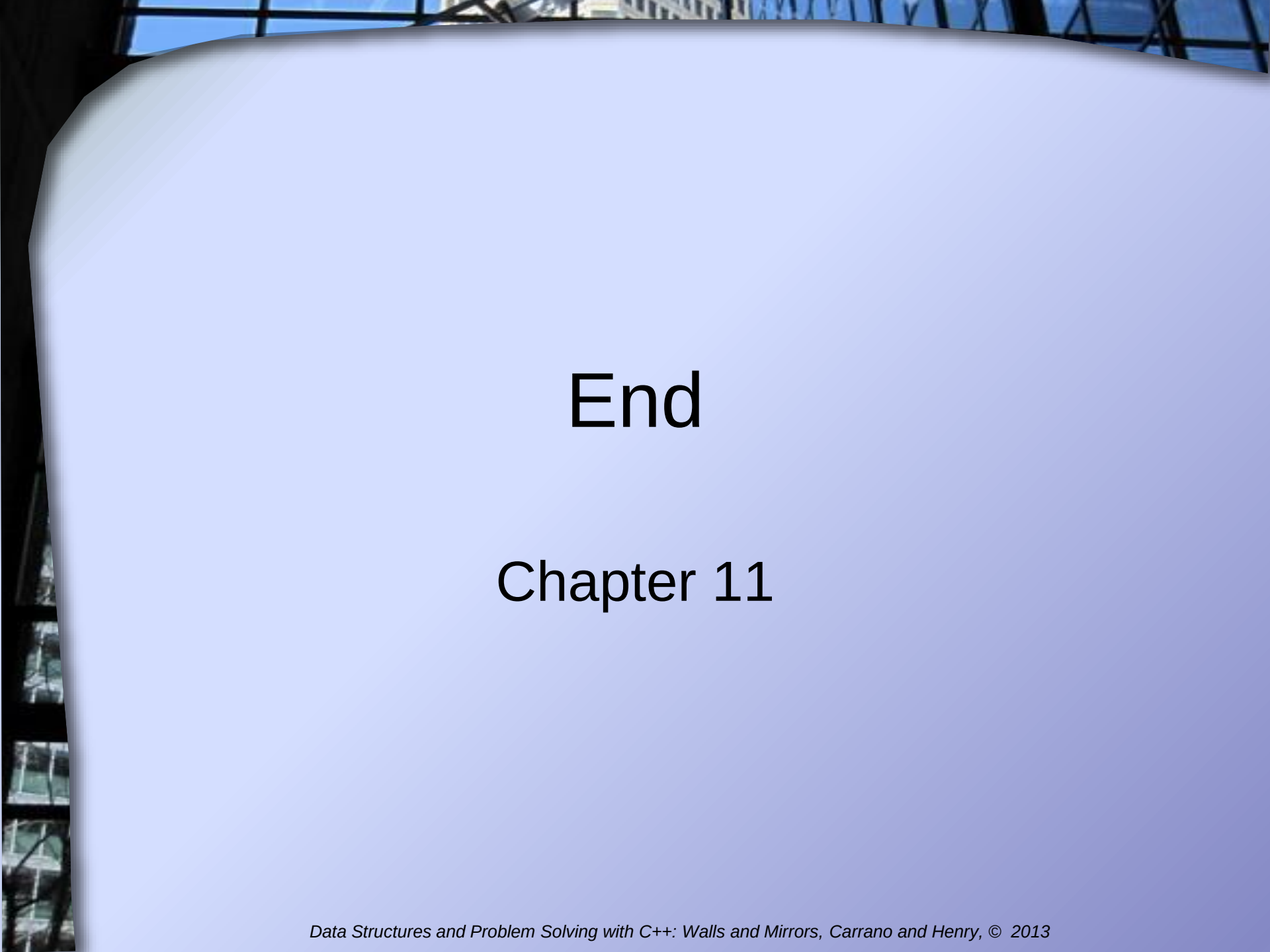
The Radix Sort

- Analysis
  - Has order O(n)
  - Large n requires significant amount of memory, especially if arrays are used
  - Memory can be saved by using chain of linked nodes
- Radix sort more appropriate for chain than for array

Comparison of Sorting Algorithms

	Worst case	Average case
Selection sort	n^2	n^2
Bubble sort	n^2	n^2
Insertion sort	n^2	n^2
Merge sort	$n \times \log n$	$n \times \log n$
Quick sort	n^2	$n \times \log n$
Radix sort	n	n
Tree sort	n^2	$n \times \log n$
Heap sort	$n \times \log n$	$n \times \log n$

FIGURE 11-15 Approximate growth rates of time required for eight sorting algorithms



End

Chapter 11