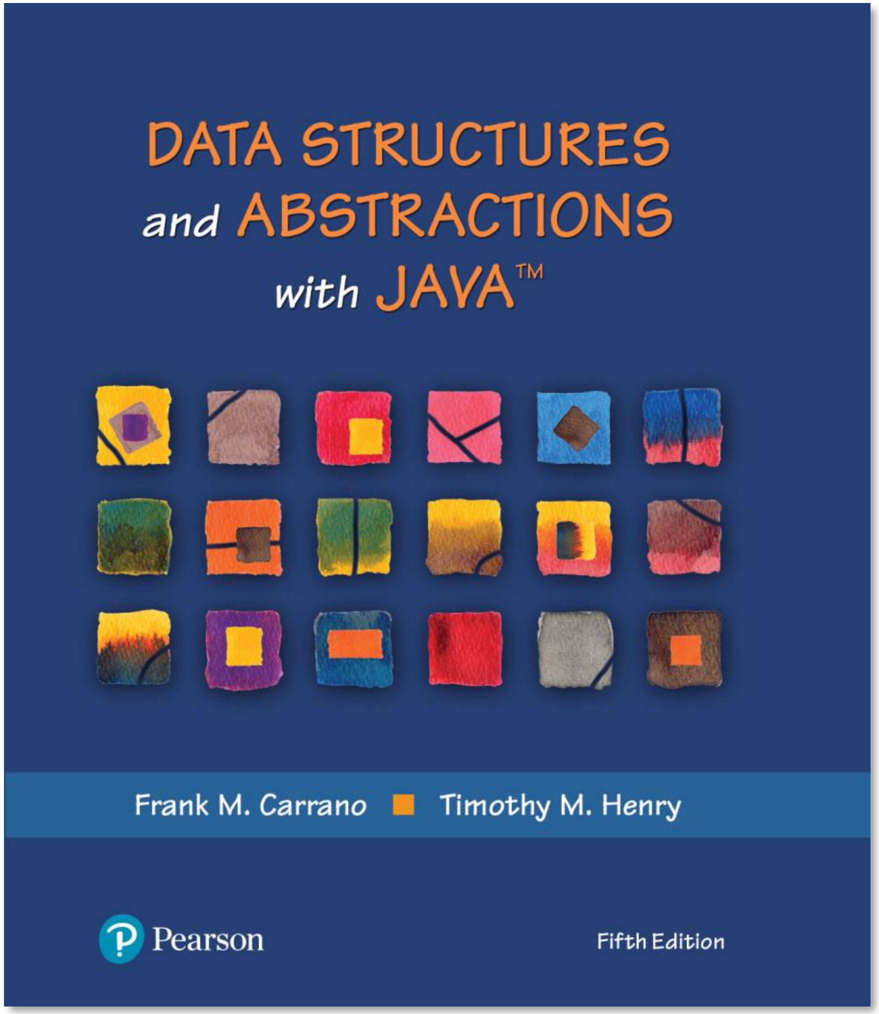
Data Structures and Abstractions with Java™

5th Edition



Chapter 16

Faster Sorting Methods

Merge Sort

- Divides an array into halves
- Sorts the two halves,
  - Then merges them into one sorted array.
- The algorithm for merge sort is usually stated recursively.
- Major programming effort is in the merge process

Merge Sort

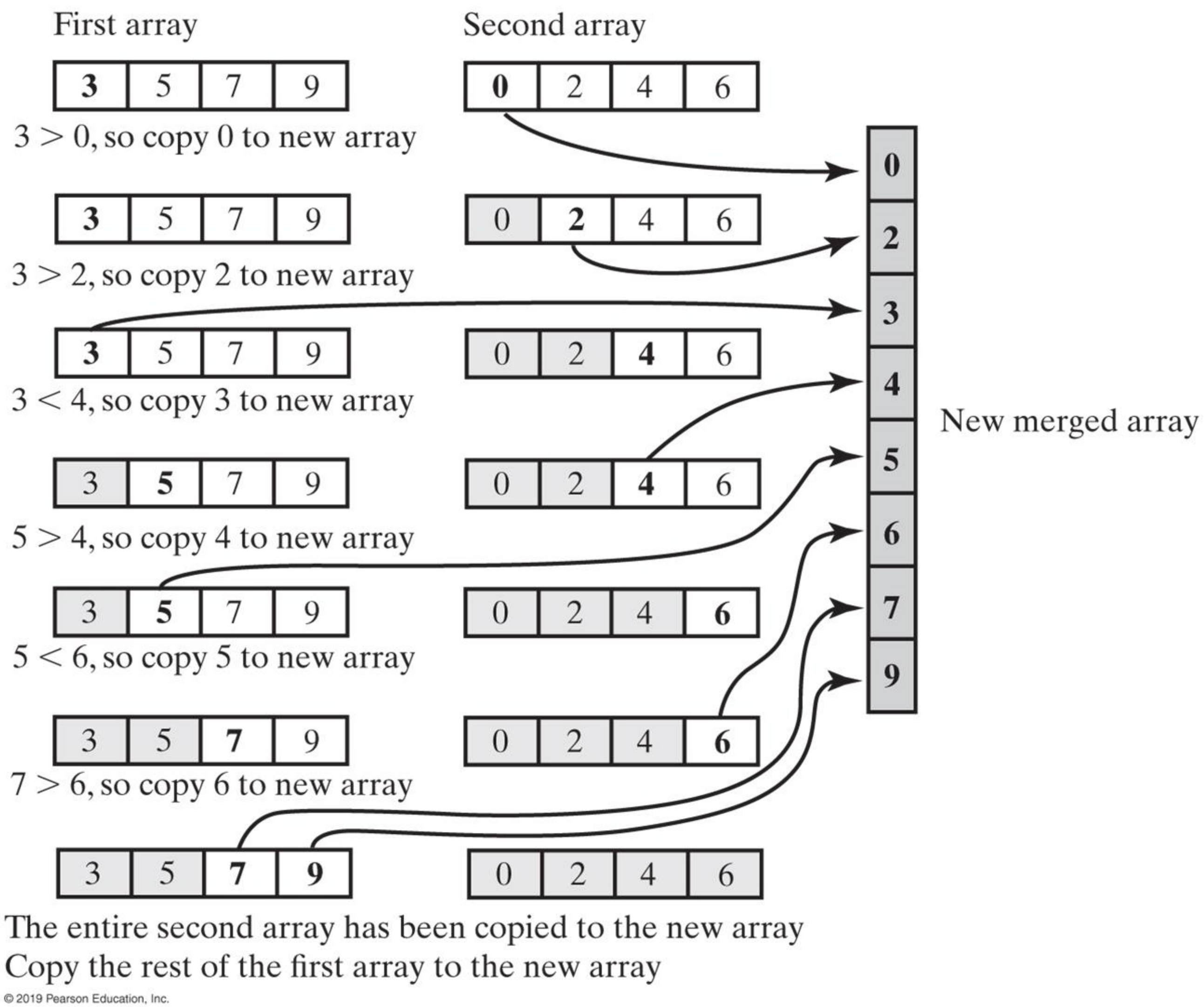


FIGURE 16-1 Merging two sorted arrays into one sorted array

Marge Sort

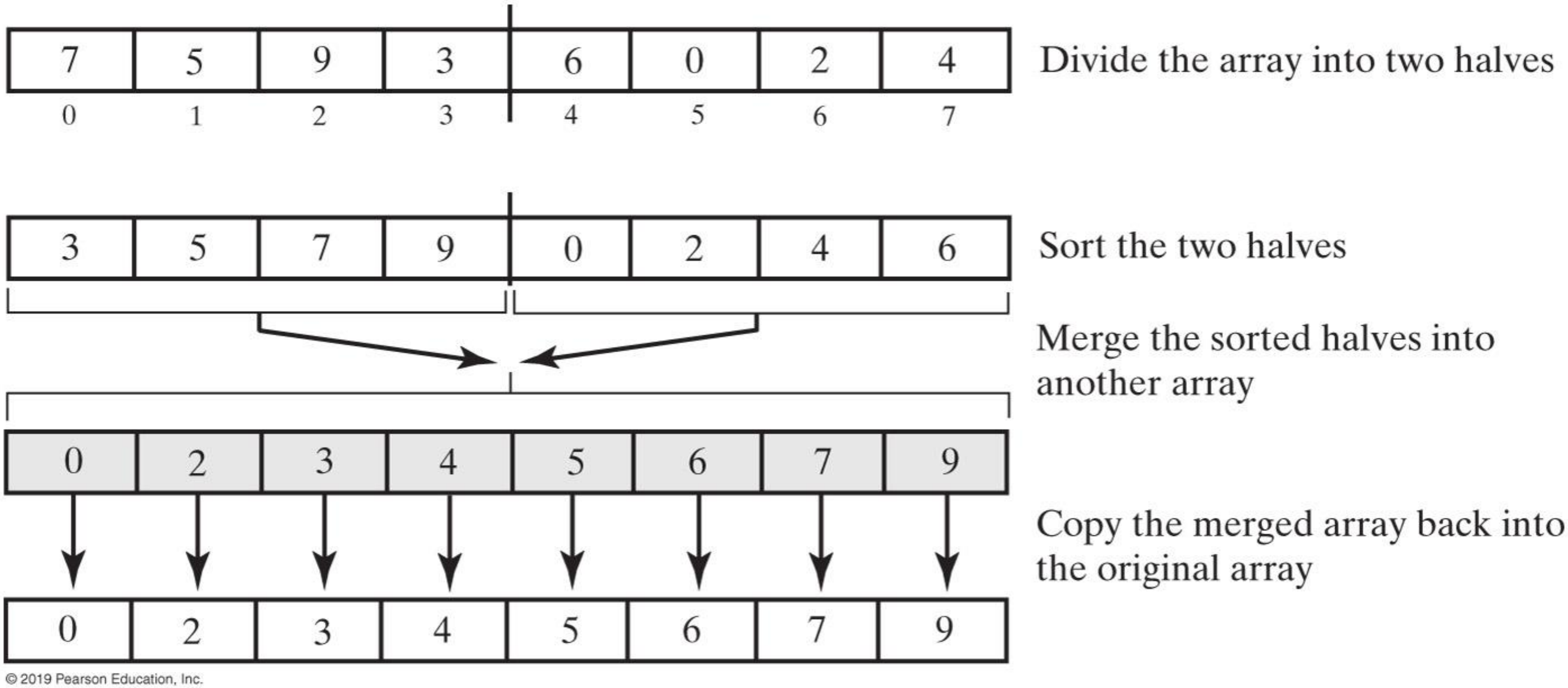


FIGURE 16-2 The major steps in a merge sort

Recursive Merge Sort

Algorithm mergeSort(a, tempArray, **first**, last)
// *Sorts the array entries* a[first..last] *recursively*.
if (first < last)
{
    mid = *approximate midpoint between* first *and* last
    mergeSort(a, tempArray, first, mid)
    mergeSort(a, tempArray, mid + 1, last)
    Merge the sorted halves a[first..mid] *and* a[mid + 1..last] *using the array* tempArray
}

Recursive algorithm for merge sort.

Merge Sort

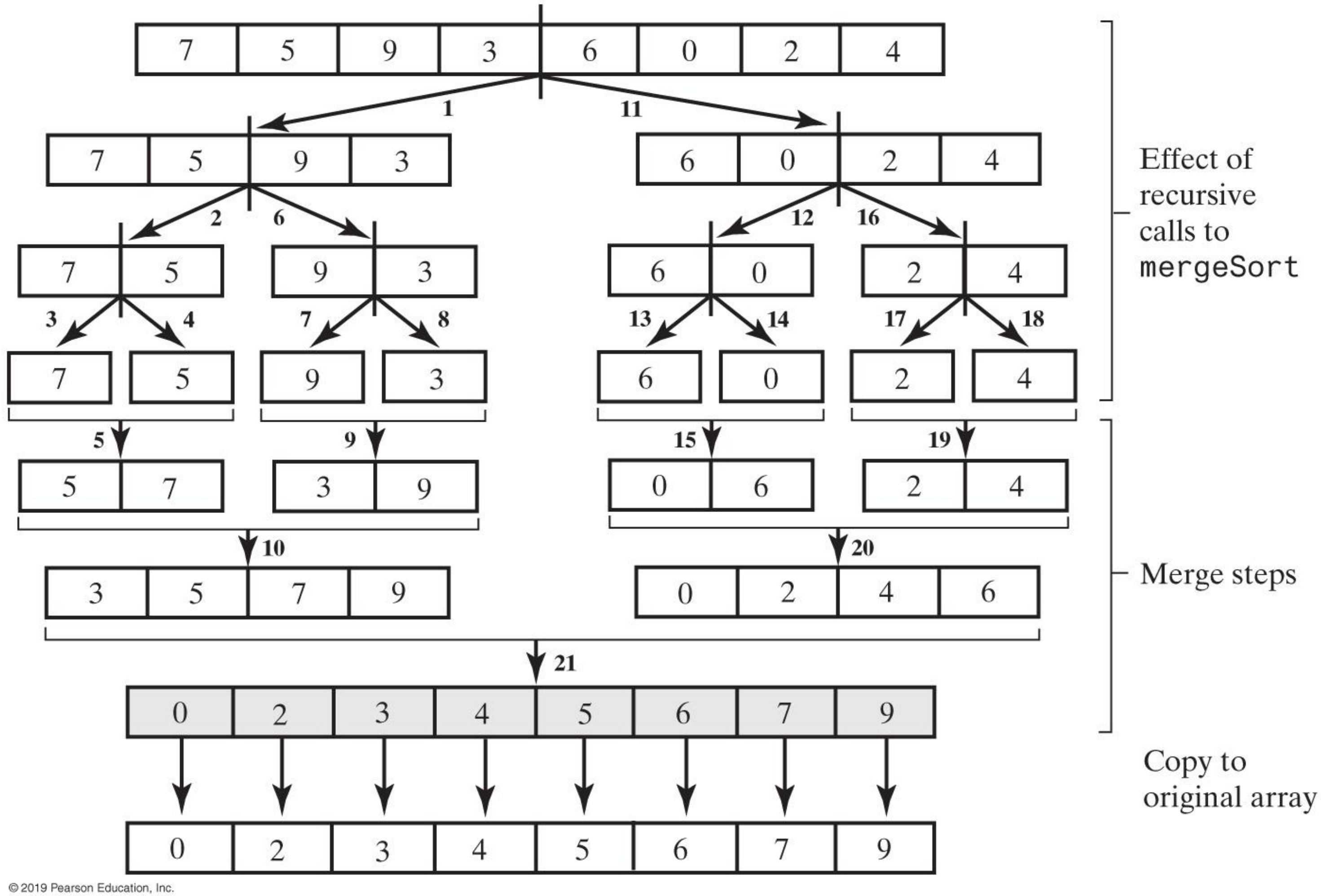
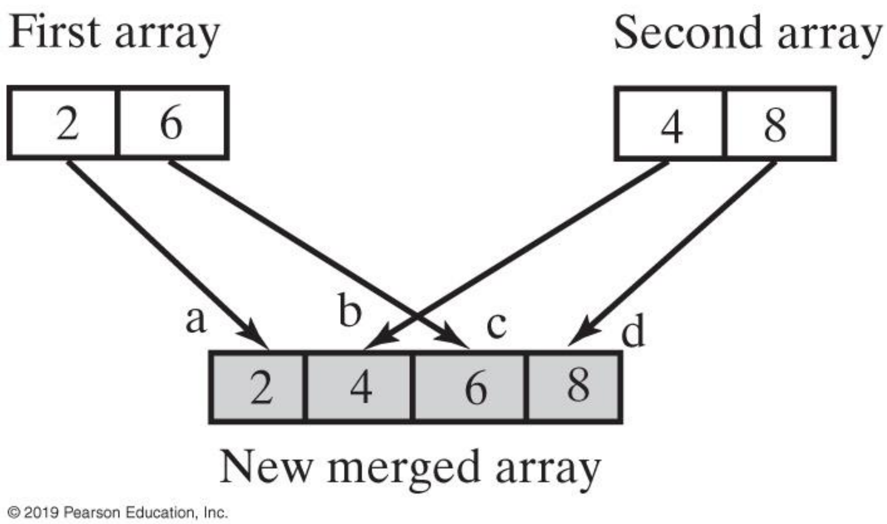


FIGURE 16-3 The effect of the recursive calls and the merges during a merge sort

Merge Sort



a. $2 < 4$, so copy 2 to new array

b. $6 > 4$, so copy 4 to new array

c. $6 < 8$, so copy 6 to new array

d. Copy 8 to new array

Efficiency is O(*n* log *n*)

FIGURE 16-4 A worst-case merge of two sorted arrays

Iterative Merge Sort

- Less simple than recursive version.
  - Need to control the merges.
- Will be more efficient of both time and space.
  - But, trickier to code without error.

Iterative Merge Sort

- Starts at beginning of array
  - Merges pairs of individual entries to form two-entry subarrays
- Returns to the beginning of array and merges pairs of the two-entry subarrays to form four-entry subarrays
  - And so on
- After merging all pairs of subarrays of a particular length, might have entries left over.

Merge Sort in the Java Class Library

- Class **`Arrays`** in the package **`java.util`** defines versions of a static method sort

```
public static void sort(Object[] a)

public static void sort(Object[] a, int first, int after)
```

Quick Sort

- Divides an array into two pieces
  - Pieces are not necessarily halves of the array
  - Chooses one entry in the array—called the pivot
- Partitions the array

Quick Sort

- When pivot chosen, array rearranged such that:
  - Pivot is in position that it will occupy in final sorted array
  - Entries in positions before pivot are less than or equal to pivot
  - Entries in positions after pivot are greater than or equal to pivot

Quick Sort

```
Algorithm quickSort(a, first, last)
// Sorts the array entries a[first..last] recursively.
if (first < last)
{
    Choose a pivot
    Partition the array about the pivot
    pivotIndex = index of pivot
    quickSort(a, first, pivotIndex − 1) // Sort Smaller
    quickSort(a, pivotIndex + 1, last) //Sort Larger
}
```

Algorithm that describes our sorting strategy

Quick Sort

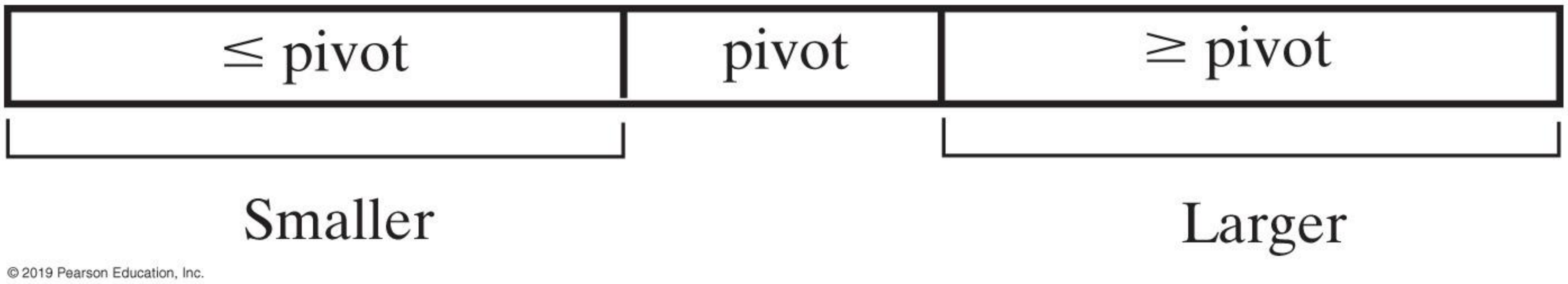


FIGURE 16-5 A partition of an array during a quick sort

Quick Sort Partitioning (Part 1)

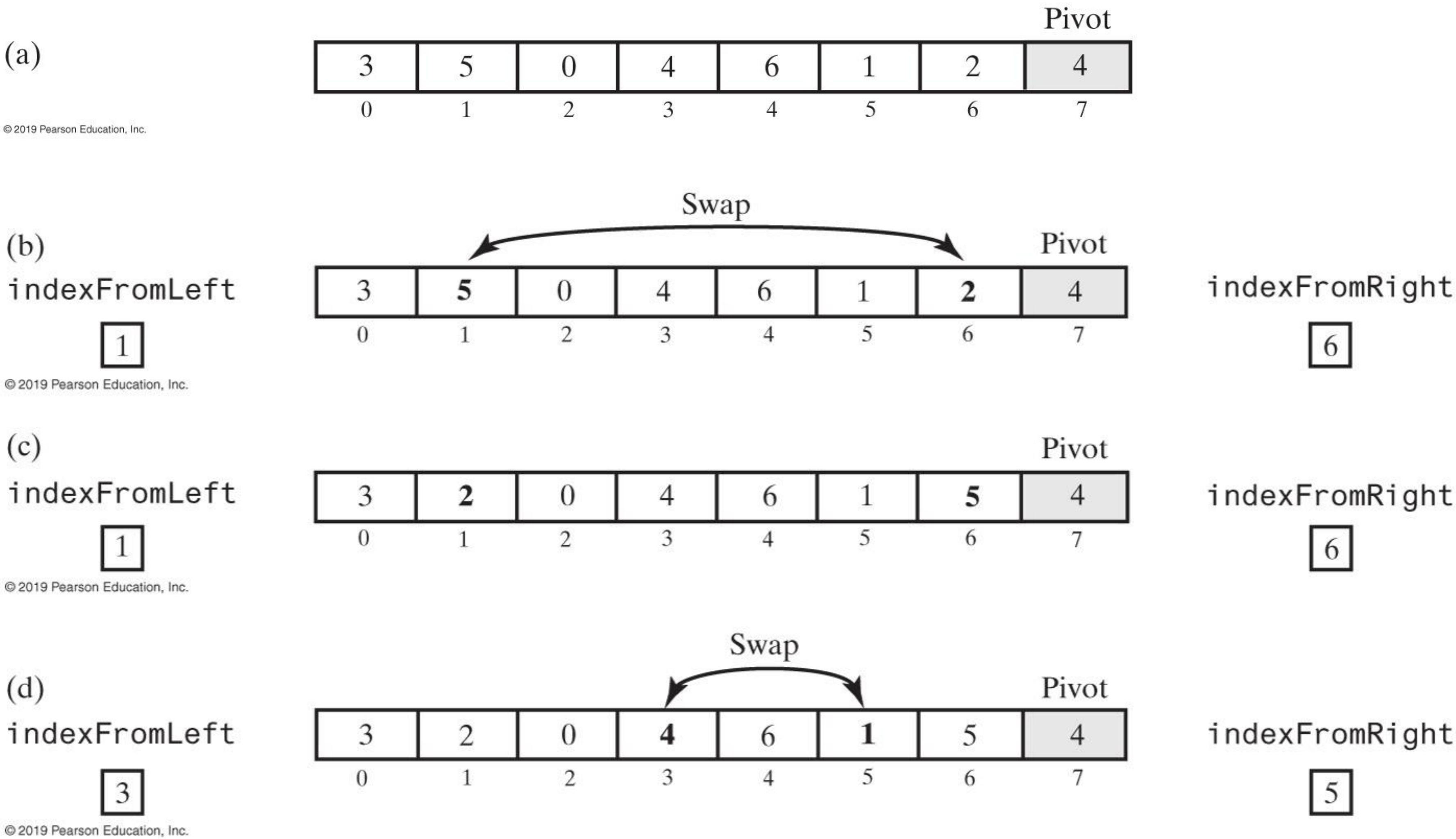


FIGURE 16-6 A partitioning strategy for quick sort

Quick Sort Partitioning (Part 2)

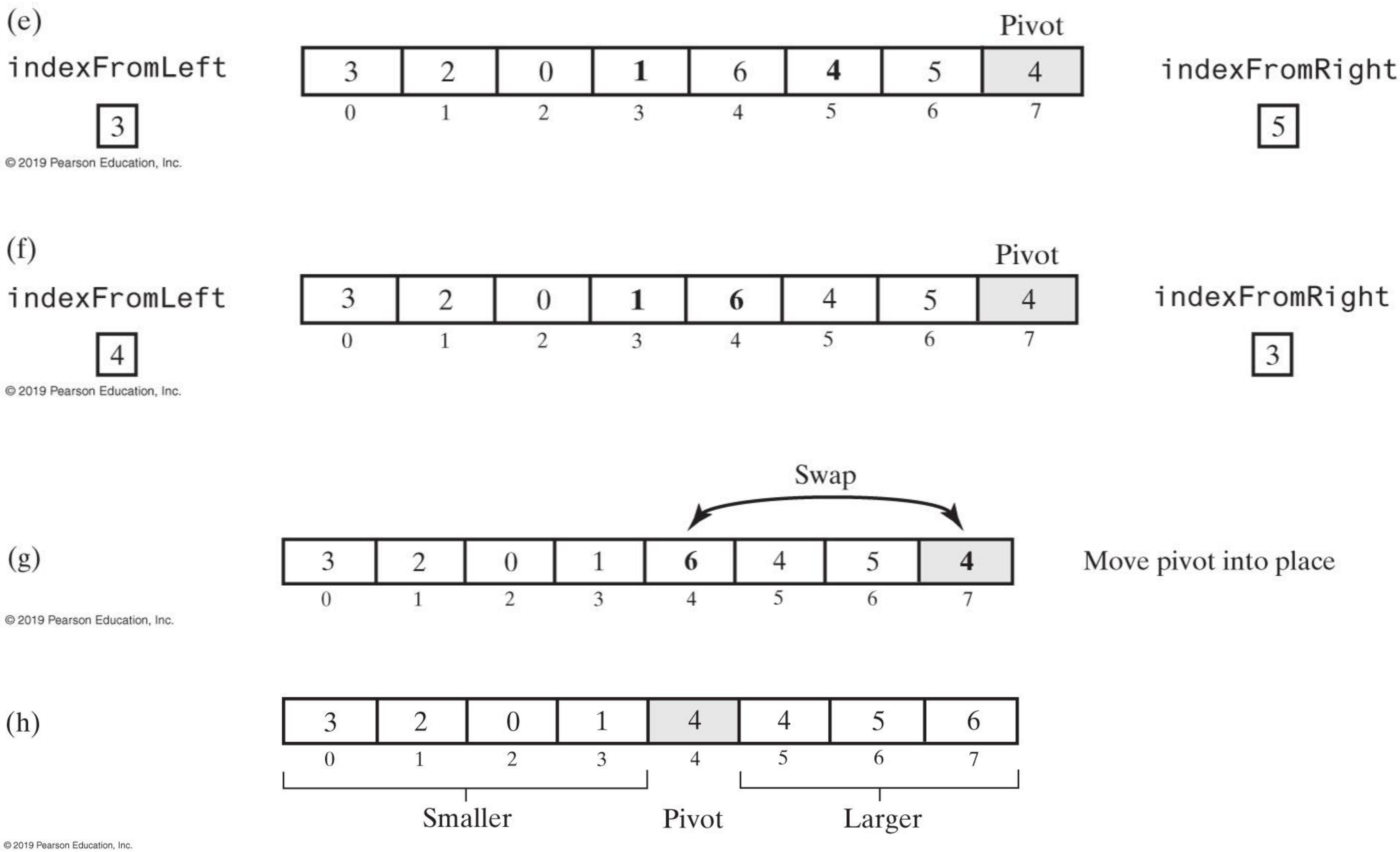


FIGURE 16-6 A partitioning strategy for quick sort

Quick Sort Partitioning

(a) The original array

(b) The array before partitioning and just after positioning the pivot

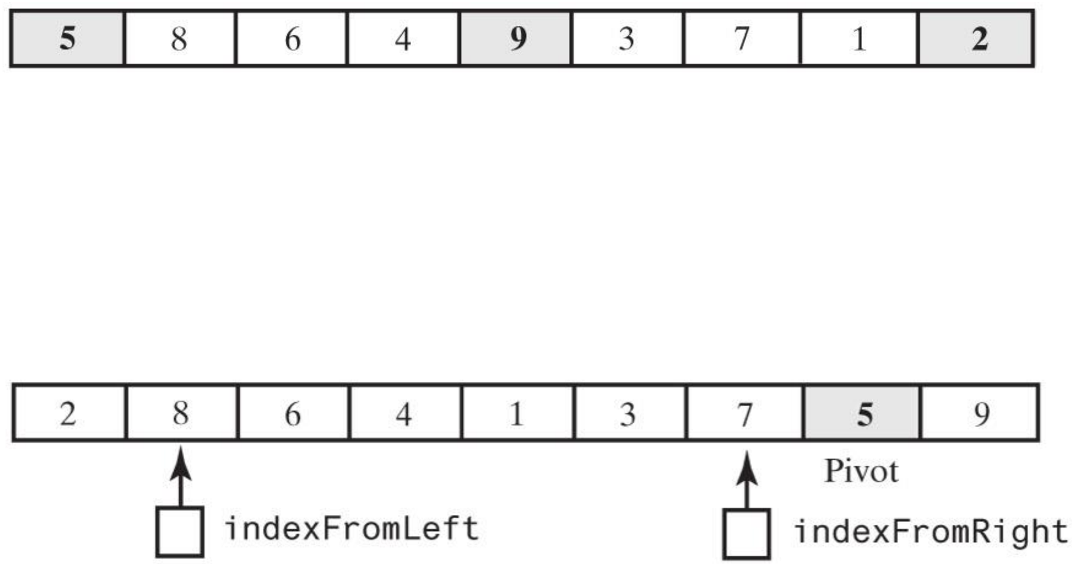
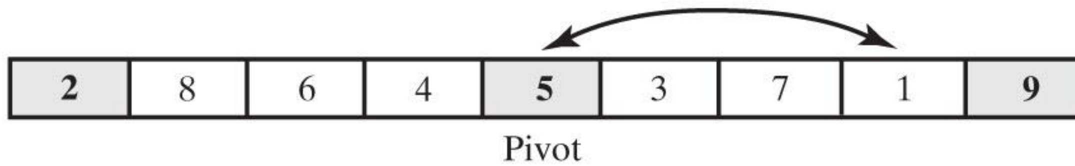


FIGURE 16-7 Median-of-three pivot selection

Quick Sort Partitioning

(a) The array after median-of-three pivot selection, as shown in Figure 16-7b



(b) The array before partitioning and just after positioning the pivot

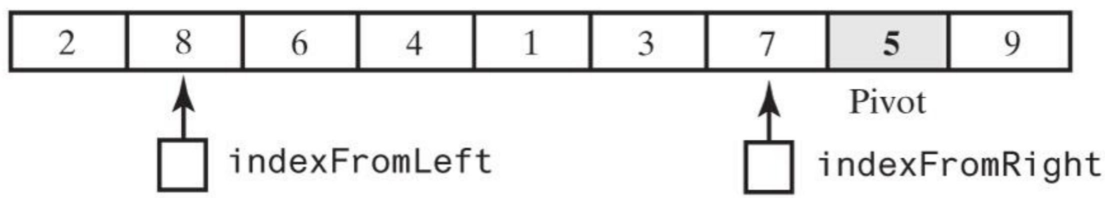


FIGURE 16-8 The array after selecting and positioning the pivot and just before partitioning

Quick Sort in the Java Class Library

- Class **`Arrays`** in the package **`java.util`** defines versions of a static method sort

```
public static void sort(type[] a)

public static void sort(type[] a, int first, int after)
```

Radix Sort

- Does not use comparison
- Treats array entries as if they were strings that have the same length.
  - Group integers according to their rightmost character (digit) into “buckets”
  - Repeat with next character (digit), etc.

Radix Sort (Part 1)

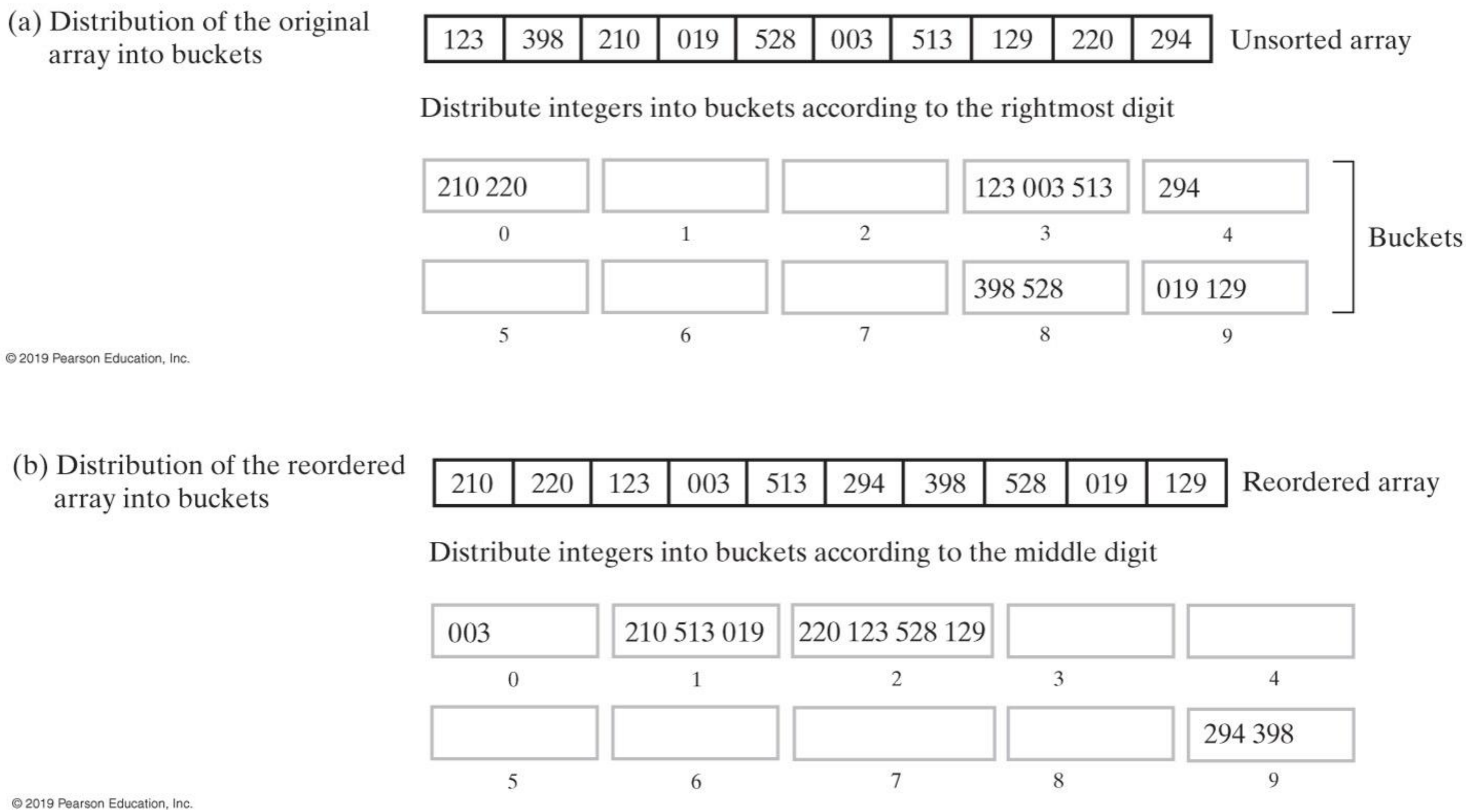


FIGURE 16-9 The steps of a radix sort

Radix Sort (Part 2)

(c) Distribution of the reordered array into buckets

003	210	513	019	220	123	528	129	294	398

Reordered array

Distribute integers into buckets according to the leftmost digit

0	1	2	3	4
003 019	123 129	210 220 294	398	

5	6	7	8	9
513 528				

(d) Sorting is complete

003	019	123	129	210	220	294	398	513	528

Sorted array

FIGURE 16-9 The steps of a radix sort

Algorithm Comparison

	Best Case	Average Case	Worst Case
Radix Sort	$O(n)$	$O(n)$	$O(n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Shell Sort	$O(n)$	$O(n^{1.5})$	$O(n^{1.5})$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

FIGURE 16-10 The time efficiency of various sorting algorithms, expressed in Big Oh notation

Comparing Function Growth Rates

	10	10^2	10^3	10^4	10^5	10^6
n	10	100	1,000	10,000	100,000	1,000,000
$n \log n$	33	664	9,966	132,877	1,660,964	19,931,569
$n^{1.5}$	32	1,000	31,623	1,000,000	319,622,777	109
n^2	100	10,000	1,000,000	108	1,010	1,012

FIGURE 16-11 A comparison of growth-rate functions as n increases

End

Chapter 16