Sorting
19.1 Introduction (Cont.)

• Sorting data
  – Place data in order
    • Typically ascending or descending
    • Based on one or more sort keys
  – Algorithms
    • Insertion sort
    • Selection sort
    • Merge sort
      – More efficient, but more complex
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Algorithm</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Sorting Algorithms</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Insertion sort</td>
<td>Section 7.8</td>
</tr>
<tr>
<td>8</td>
<td>Selection sort</td>
<td>Section 8.6</td>
</tr>
<tr>
<td>19</td>
<td>Recursive merge sort</td>
<td>Section 19.3.3</td>
</tr>
<tr>
<td></td>
<td>Bubble sort</td>
<td>Exercises 19.5 and 19.6</td>
</tr>
<tr>
<td></td>
<td>Bucket sort</td>
<td>Exercise 19.7</td>
</tr>
<tr>
<td></td>
<td>Recursive quicksort</td>
<td>Exercise 19.10</td>
</tr>
<tr>
<td>20</td>
<td>Binary tree sort</td>
<td>Section 20.7</td>
</tr>
<tr>
<td>22</td>
<td><strong>sort</strong> standard library function</td>
<td>Section 22.5.6</td>
</tr>
<tr>
<td></td>
<td>Heap sort</td>
<td>Section 22.5.12</td>
</tr>
</tbody>
</table>

**Fig. 19.1** | **Searching and sorting algorithms in this text. (Part 2 of 2)**
19.3 Sorting Algorithms

- **Sorting algorithms**
  - Placing data into some particular order
    - Such as ascending or descending
  - One of the most important computing applications
  - End result, a sorted vector, will be the same no matter which algorithm is used
    - Choice of algorithm affects only runtime and memory use
19.3.1 Efficiency of Selection Sort

• Selection sort
  – At $i$th iteration
    • Swaps the $i$th smallest element with element $i$
  – After $i$th iteration
    • Smallest $i$ elements are sorted in increasing order in first $i$ positions
  – Requires a total of $(n^2 - n)/2$ comparisons
    • Iterates $n - 1$ times
      – In $i$th iteration, locating $i$th smallest element requires $n - i$ comparisons
    • Has Big O of $O(n^2)$
19.3.2 Efficiency of Insertion Sort

• Insertion sort
  – At $i$th iteration
    • Insert $(i + 1)$th element into correct position with respect to first $i$ elements
  – After $i$th iteration
    • First $i$ elements are sorted
  – Requires a worst-case of $n^2$ inner-loop iterations
    • Outer loop iterates $n - 1$ times
      – Inner loop requires $n - 1$ iterations in worst case
    • For determining Big O, nested statements mean multiply the number of iterations
    • Has Big O of $O(n^2)$
19.3.3 Merge Sort (A Recursive Implementation)

• Merge sort
  – Sorts vector by
    • Splitting it into two equal-size subvectors
      – If vector size is odd, one subvector will be one element larger than the other
    • Sorting each subvector
    • Merging them into one larger, sorted vector
      – Repeatedly compare smallest elements in the two subvectors
      – The smaller element is removed and placed into the larger, combined vector
19.3.3 Merge Sort (A Recursive Implementation) (Cont.)

• Merge sort (Cont.)
  – Our recursive implementation
    • Base case
      – A vector with one element is already sorted
    • Recursion step
      – Split the vector (of $\geq 2$ elements) into two equal halves
        • If vector size is odd, one subvector will be one element larger than the other
      – Recursively sort each subvector
      – Merge them into one larger, sorted vector
19.3.3 Merge Sort (A Recursive Implementation) (Cont.)

- **Merge sort (Cont.)**
  - Sample merging step
    - Smaller, sorted vectors
      - A: 4 10 34 56 77
      - B: 5 30 51 52 93
    - Compare smallest element in A to smallest element in B
      - 4 (A) is less than 5 (B)
        - 4 becomes first element in merged vector
      - 5 (B) is less than 10 (A)
        - 5 becomes second element in merged vector
      - 10 (A) is less than 30 (B)
        - 10 becomes third element in merged vector
    - Etc.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Location</th>
<th>Big O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Searching Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear search</td>
<td>Section 7.7</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Binary search</td>
<td>Section 19.2.2</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Recursive linear search</td>
<td>Exercise 19.8</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Recursive binary search</td>
<td>Exercise 19.9</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>Sorting Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion sort</td>
<td>Section 7.8</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Selection sort</td>
<td>Section 8.6</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Merge sort</td>
<td>Section 19.3.3</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>Exercises 16.3 and 16.4</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
| Quick sort           | Exercise 19.10    | Worst case: $O(n^2)$  
|                      |                   | Average case: $O(n \log n)$ |

**Fig. 19.8** | **Searching and sorting algorithms with Big O values.**
<table>
<thead>
<tr>
<th>$n$</th>
<th>Approximate decimal value</th>
<th>O$(\log n)$</th>
<th>O$(n)$</th>
<th>O$(n \log n)$</th>
<th>O$(n^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{10}$</td>
<td>1000</td>
<td>10</td>
<td>$2^{10}$</td>
<td>10</td>
<td>$2^{10}$</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>1,000,000</td>
<td>20</td>
<td>$2^{20}$</td>
<td>20</td>
<td>$2^{20}$</td>
</tr>
<tr>
<td>$2^{30}$</td>
<td>1,000,000,000</td>
<td>30</td>
<td>$2^{30}$</td>
<td>30</td>
<td>$2^{30}$</td>
</tr>
</tbody>
</table>

**Fig. 19.9** | Approximate number of comparisons for common Big O notations.