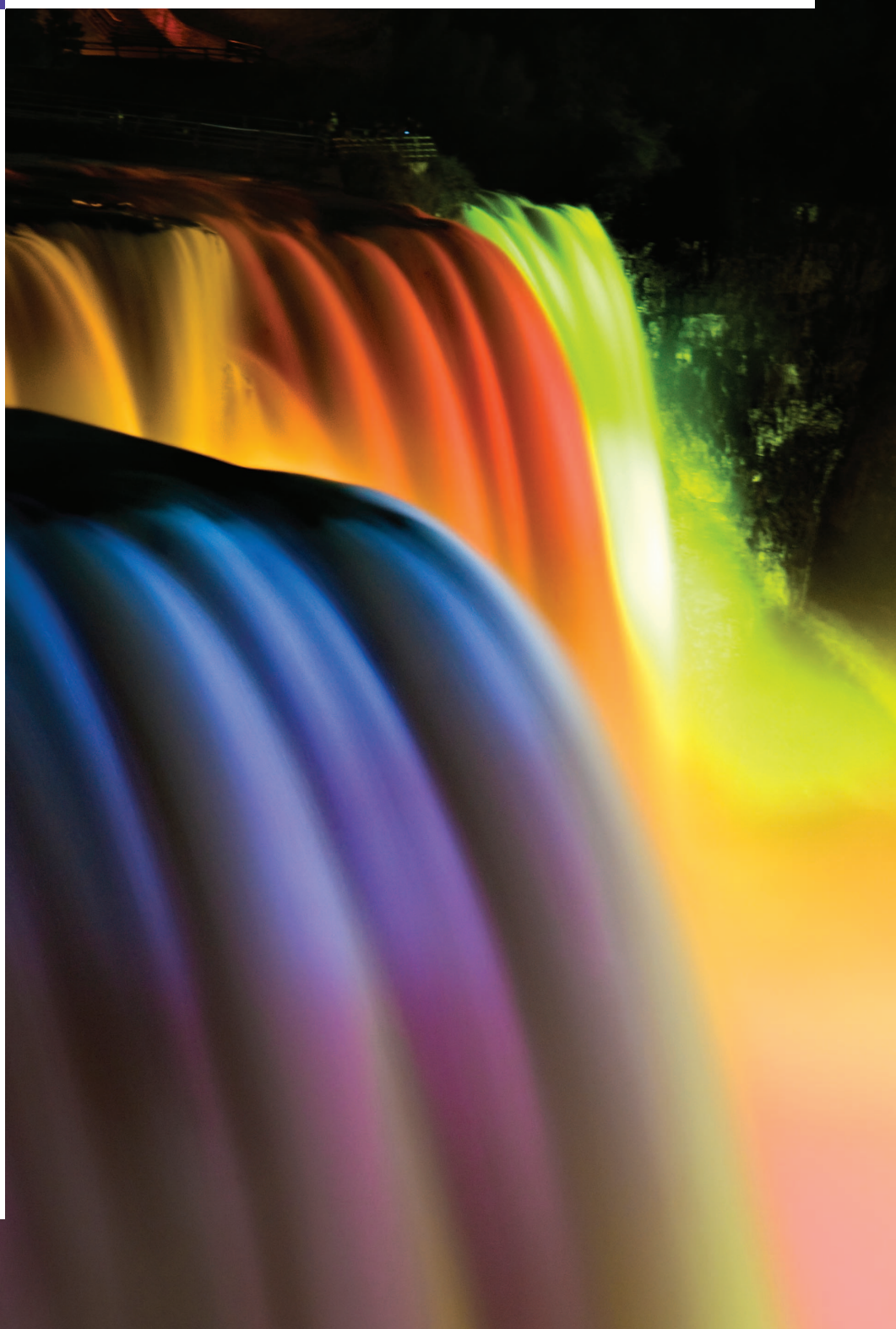
19

Searching, Sorting and Big O: Solutions



With sobs and tears he sorted out
Those of the largest size …
—Lewis Carroll

Attempt the end, and never stand to doubt;
Nothing's so hard, but search will find it out.
—Robert Herrick

'Tis in my memory lock'd,
And you yourself shall keep the key of it.
—William Shakespeare

It is an immutable law in business that words are words, explanations are explanations, promises are promises — but only performance is reality.
—Harold S. Green

Objectives

In this chapter you'll learn:

- To search for a given value in an array using linear search and binary search.
- To sort arrays using the iterative selection and insertion sort algorithms.
- To sort arrays using the recursive merge sort algorithm.
- To determine the efficiency of searching and sorting algorithms.

Self-Review Exercises

19.1 Fill in the blanks in each of the following statements:

a) A selection sort application would take approximately _____ times as long to run on a 128-element array as on a 32-element array.

ANS: 16, because an $O(n^2)$ algorithm takes 16 times as long to sort four times as much information.

b) The efficiency of merge sort is _____.

ANS: $O(n \log n)$.

19.2 What key aspect of both the binary search and the merge sort accounts for the logarithmic portion of their respective Big Os?

ANS: Both of these algorithms incorporate "halving"—somehow reducing something by half. The binary search eliminates from consideration one-half of the array after each comparison. The merge sort splits the array in half each time it's called.

19.3 In what sense is the insertion sort superior to the merge sort? In what sense is the merge sort superior to the insertion sort?

ANS: The insertion sort is easier to understand and to program than the merge sort. The merge sort is far more efficient [$O(n \log n)$] than the insertion sort [$O(n^2)$].

19.4 In the text, we say that after the merge sort splits the array into two subarrays, it then sorts these two subarrays and merges them. Why might someone be puzzled by our statement that "it then sorts these two subarrays"?

ANS: In a sense, it does not really sort these two subarrays. It simply keeps splitting the original array in half until it provides a one-element subarray, which is, of course, sorted. It then builds up the original two subarrays by merging these one-element arrays to form larger subarrays, which are then merged, and so on.

Exercises

NOTE: Solutions to the programming exercises are located in the `ch19solutions` folder. Each exercise has its own folder named `ex19_##` where ## is a two-digit number representing the exercise number. For example, exercise 19.10's solution is located in the folder `ex19_10`.

19.5 *(Bubble Sort)* Implement bubble sort—another simple yet inefficient sorting technique. It's called bubble sort or sinking sort because smaller values gradually "bubble" their way to the top of the array (i.e., toward the first element) like air bubbles rising in water, while the larger values sink to the bottom (end) of the array. The technique uses nested loops to make several passes through the array. Each pass compares successive pairs of elements. If a pair is in increasing order (or the values are equal), the bubble sort leaves the values as they are. If a pair is in decreasing order, the bubble sort swaps their values in the array.

The first pass compares the first two elements of the array and swaps their values if necessary. It then compares the second and third elements in the array. The end of this pass compares the last two elements in the array and swaps them if necessary. After one pass, the largest element will be in the last index. After two passes, the largest two elements will be in the last two indices. Explain why bubble sort is an $O(n^2)$ algorithm.

ANS: Bubble sort contains two nested `for` loops. The outer `for` loop (lines 24–33) iterates over `data.length - 1` passes. The inner `for` loop (lines 27–32) iterates over `data.length - 1` elements in the array. When loops are nested, the respective orders are multiplied. This is because for each of $O(n)$ iterations of the outside loop, there are $O(n)$ iterations of the inner loop. This results in a total Big O of $O(n^2)$.