It is the next episode in the quest for a better map, and it anticipates the tree data structures of Chapter 5.

Together these topics continue the conversation of this book: What ADTs are needed to support a given algorithm? What data structures implement a given ADT? What algorithms provide the operations of a given data structure?

3.1 *Linear-time sorting algorithms*

When you studied sorting algorithms—in Sections 1.3 and 1.4 or in an earlier course—you learned to discern good sorts from bad sorts. Good sorts do work proportional to the size of the sequence for each level in a binary tree, altogether costing $\Theta(n \lg n)$. Bad sorts do work proportional to the size of the sequence for each element in the sequence, which costs $\Theta(n^2)$.

Good sorts	**Bad sorts**
Merge	Selection
Quick (expected case)	Insertion
Shell (see Project 1.2)	Bubble
Heap (see Section 3.3)	

The ultimate measure for algorithms is their running time, but as we saw in Section 1.4, counting the number of comparisons between elements in a sequence is a reasonable proxy for running time when comparing sorting algorithms. At least we can say that if the expected case of a sorting algorithm makes $\Theta(n \lg n)$ comparisons, then its running time must be $\Omega(n \lg n)$—that is, it can't be better than $n \lg n$.

It turns out that $\Theta(n \lg n)$ is in fact the best we can do for the expected case of sorting algorithms that make decisions by comparing elements in the sequence. Put formally,

Theorem 1 *If T is an algorithm that sorts a sequence by comparing pairs of elements in the sequence, then the expected running time of T is $\Omega(n \lg n)$.*

Here is the intuition behind that result.[1] The execution of such a sorting algorithm entails a series of decisions based on comparing two elements in the sequence, "Which comes first, this one or that one?" Since any permutation of the sorted

[1] For an accessible presentation of the proof of this theorem, see Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009, pp. 191–194, pp. 191–194. For the definitive exposition, see Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. 2nd ed. Addison Wesley Longman Publishing Co., Inc., 1998, pp. 180-197.