Page 601

Projects

P-12.56 Implement a nonrecursive, in-place version of the quick-sort algorithm, as described at the end of Section 12.3.2.

P-12.57 Experimentally compare the performance of in-place quick-sort and a version of quick-sort that is not in-place.

P-12.58 Perform a series of benchmarking tests on a version of merge-sort and quick-sort to determine which one is faster. Your tests should include sequences that are “random” as well as “almost” sorted.

P-12.59 Implement deterministic and randomized versions of the quick-sort algorithm and perform a series of benchmarking tests to see which one is faster. Your tests should include sequences that are very “random” looking as well as ones that are “almost” sorted.

P-12.60 Implement an in-place version of insertion-sort and an in-place version of quick-sort. Perform benchmarking tests to determine the range of values of n where quick-sort is on average better than insertion-sort.

P-12.61 Design and implement a version of the bucket-sort algorithm for sorting a list of n entries with integer keys taken from the range [0, N – 1], for N ≥ 2. The algorithm should run in O(n + N) time.

P-12.62 Design and implement an animation for one of the sorting algorithms described in this chapter. Your animation should illustrate the key properties of this algorithm in an intuitive manner.

Chapter Notes

Knuth’s classic text on Sorting and Searching [65] contains an extensive history of the sorting problem and algorithms for solving it. Huang and Langston [53] show how to merge two sorted lists in-place in linear time. The standard quick-sort algorithm is due to Hoare [51]. Several optimizations for quick-sort are described by Bentley and McIlroy [16]. More information about randomization, including Chernoff bounds, can be found in the appendix and the book by Motwani and Raghavan [80]. The quick-sort analysis given in this chapter is a combination of the analysis given in an earlier Java edition of this book and the analysis of Kleinberg and Tardos [60]. Exercise C-12.32 is due to Littman. Gonnet and Baeza-Yates [44] analyze and compare experimentally several sorting algorithms. The term “prune-and-search” comes originally from the computational geometry literature (such as in the work of Clarkson [26] and Megiddo [75]). The term “decrease-and-conquer” is from Levitin [70].