Data Structures and Algorithms in Python

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter 12 Sorting and Selection

Hints

Reinforcement

R-12.1) Argue in more detail about why the merge-sort tree has height $O(\log n)$.

R-12.2) Recall the definition of a recursion trace from Chapter 4.

R-12.3) Consider "padding" out the input with infinities to make n a power of 2. How does this affect the running time?

R-12.4) Consider an input with duplicates.

R-12.5) Consider an input with duplicates.

R-12.6) You need a different way to handle the equal case in the merge procedure.

R-12.7) Consider using something like the merge for merge-sort.

R-12.8) Derive a recurrence equation for this algorithm assuming n is a power of 2. Does it look familiar? It should.

R-12.9) You want each choice of pivot to form a very bad split.

R-12.10) Recall what is the best possible split we can get for a given pivot and then derive a recurrence equation assuming we get this kind of a split. This equation should look familiar.

R-12.11) To gain intuition, work out the first few splits on the sequence $(1,1,1,1,1,1,1,1,2)$.

R-12.12) Clearly the flaw must involve a case where a pass of the outermost loop completes with the value of left precisely equal to right.

R-12.13) Develop a test case in which left equals right immediately prior to the evaluation of line 14.

R-12.14) $1/n^2$ is the same as $1/2^{2\log n}$.

R-12.15) What is the maximum number of external nodes that a binary tree of height n can have?

R-12.16) Recall that to sort n elements with a comparison-based algorithm requires $\Omega(n \log n)$ time.

R-12.17) No. Why not?

R-12.18) Work out some examples with triples first. Then move on to d-tuples.

R-12.19) The two running times are not the same.

R-12.20) There are only two possible key values.

R-12.21) Try to mimic the partition method used in the in-place quick-sort algorithm.

R-12.22) Check out the discussion comparing the various sorting algorithms.

R-12.22) Check out the discussion comparing the various sorting algorithms.

R-12.23) Consider the complexity of comparisons versus using elements as indices into an array.

R-12.24) Think of the worst possible way to choose pivots in this algorithm.

Creativity

C-12.25) How do you know S and T have the same elements in them?

C-12.26) Sort first.

C-12.27) Can you adapt the merge algorithm of Code Fragment 12.3 to directly manipulate nodes of the list.

C-12.28) Merge-sort is a particularly good choice for a linked list.

C-12.29) A queue of queues can be very helpful.

C-12.30) It would be easier if the last element in the array were still the pivot...

C-12.31) For the overall worst case, recall the worst case for choosing the last element as the pivot.

C-12.32) You need to use an induction hypothesis that $T(n) \leq cn \log n$, for some constant c.

C-12.33) Carefully consider how to maintain the stated invariant when classifying each additional element.

C-12.34) Sort the votes, and then determine who received the maximum number of votes.

C-12.35) Think of a data structure that can be used for sorting in a way that only stores k elements when there are only k keys.

C-12.36) Shoot for an $O(n)$ expected running time.

C-12.37) Develop a meaningful way to break ties during comparisons .

C-12.38) Sort A and B first.

C-12.39) Think of alternate ways of viewing the elements.

C-12.40) Find a way of sorting them as a group that keeps each sequence contiguous in the final listing.

C-12.41) Sort first.

C-12.42) Try to modify the merge-sort algorithm to solve this problem.

C-12.43) Try to modify the insertion-sort algorithm to solve this problem.

C-12.44) 1

C-12.45) Consider the graph of the equation $m = a + b$ for a fixed value of m.

C-12.46) Perform a selection first on some appropriate order statistics.

C-12.47) Try to design an efficient divide-and-conquer algorithm.

C-12.48) You will need two-passes through the data at each level of recursion.

C-12.49) Use in-place quick-sort as a starting point.

C-12.50) Think about what would be the perfect pivot in a an algorithm like quick-sort.

C-12.51) Use linear-time selection in an appropriate way.

C-12.52) Think of an alien version of quick-sort.

C-12.53) You could dynamically define a new key type with its own definition for comparisons.

C-12.54) For (a), revisit the definition of the randomized quick-sort algorithm. For (b), argue why the probability that $C_{i,j}(x) = 1$ is at most $1/2^j$ and why the dependence between $C_{i,j}(x)$'s only helps. For (c), review the book's discussion of geometric sums. For (d), just plug in the equation for μ and do the math. For (e), argue about all n elements from the bound on a single one.

C-12.55) The recurrence equation denotes two recursive calls, but one is smaller than the other.

Projects

P-12.56) Think about how to define subproblems concisely and store them on the stack. You then can use a while loop to process problems from and to this stack. Also, please see the chapter discussion about in-place quick-sort for more hints.

P-12.57) Implement the version that is not in-place first.

P-12.58) An almost sorted sequence could be one with at most a linear number of inversions.

P-12.59) An almost sorted sequence could be one with at most a linear number of inversions.

P-12.60) Be sure to perform enough tests so that your results are trustworthy.

P-12.61) Use good testing inputs to verify that your method is stable. Also, be sure to copy the elements of the list in and out of the bucket array.

P-12.62) One good animation style uses vertical lines various lengths to represent the different elements.