Data Structures and Algorithms in Java™

Sixth Edition

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 4

Algorithm Analysis

Hints

Reinforcement

R-4.1) Use powers of two as your values for n.

R-4.2) Set the running times equal, use algebra to simplify the equation, and then various powers of two to home in on the right answer.

R-4.3) Set both formulas equal to each other to determine this.

R-4.4) Any growing function will have a "flatter" curve on a log-log scale than it has on a standard scale.

R-4.5) Think of another way to write $\log n^c$.

R-4.6) Characterize this in terms of the sum of all integers from 1 to n.

R-4.7) Use the fact that if $a < b$ and $b < c$, then $a < c$.

R-4.8) Simplify the expressions, and then use the ordering of the seven important algorithm-analysis functions to order this set.

R-4.9) Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

R-4.10) Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

R-4.11) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the outer loop.

R-4.12) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the outer loop.

R-4.13) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the two outer loops.

R-4.14) Review the definition of big-Oh and use the constant from this definition.

R-4.15) Start with the product and then apply the definition of the big-Oh for $d(n)$ and then $e(n)$.

R-4.16) Use the definition of the big-Oh and add the constants (but be sure to use the right n_0).

R-4.17) You need to give a counterexample. Try the case when $d(n)$ and $e(n)$ are both $O(n)$ and be specific.

R-4.18) Use the definition of the big-Oh first to $d(n)$ and then to $f(n)$ (but be sure to use the right n_0).

R-4.19) First show that the max is always less than the sum.

R-4.20) Simply review the definitions of big-Oh and big-Omega. This one is easy.

R-4.21) Recall that $\log n^k = k \log n$.

R-4.22) Notice that $(n+1) \leq 2n$ for $n \geq 1$.

R-4.23) $2^{n+1} = 2 \cdot 2^n$.

R-4.24) Make sure you don't get caught by the fact that $\log 1 = 0$.

R-4.25) Use the definition of big-Omega, but don't get caught by the fact that $\log 1 = 0$.

R-4.26) Use the definition of big-Omega, but don't get caught by the fact that $\log 1 = 0$.

R-4.27) If $f(n)$ is a positive nondecreasing function that is always greater than 1, then $\lceil f(n) \rceil \leq f(n) + 1$.

R-4.28) You can do all rows except for $n \log n$ just by setting the function equal to the value and solving for n. For the $n \log n$ function, the easiest technique is unfortunately to simply use trial-and-error on a calculator.

R-4.29) The $O(\log n)$ calculation is performed n times.

R-4.30) The $O(n)$ calculation is performed $\log n$ times.

R-4.31) Consider the cases when all entries of X are even or odd.

R-4.32) First characterize the running time of Algorithm D using a summation.

R-4.33) Discuss how the definition of the big-Oh fits into Al's claim.

R-4.34) Recall the definition of the Harmonic number, H_n.

Creativity

C-4.35) Use sorting as a subroutine.

C-4.36) Note that 10 is a constant!

C-4.37) Think of a function that grows and shrinks at the same time without bound.

C-4.38) Use induction, a visual proof, or bound the sum by an integral.

C-4.39) Try to bound this sum term by term with a geometric progression.

C-4.40) Recall the formula for the sum of the terms of a geometric progression.

C-4.41) Use the log identity that translates $\log bx$ to a logarithm in base 2.

C-4.42) First, construct a group of candidate minimums and a group of candidate maximums.

C-4.43) Consider the sum of the maximum number of visits each friend can make without visiting his/her maximum number of times.

C-4.44) You need to line up the columns a little differently.

C-4.45) Consider computing a function of the integers in A that will immediately identify which one is missing.

C-4.46) Some informal discussion of the algorithm efficiency was given at the conclusion of Section 3.1.2.

C-4.47) Characterize the number of bits needed first.

C-4.48) Consider the first induction step.

C-4.49) Look carefully at the definition of big-Oh and rewrite the induction hypothesis in terms of this definition.

C-4.50) Use the definition of big-Omega, and make $n = 1$ and $n = 2$ your base cases.

C-4.51) Consider the contribution made by one line.

C-4.52) Try to bound from above each term in this summation.

C-4.53) Try to bound a significant number of the terms from below.

C-4.54) Consider writing a pseudocode description of this algorithm and note its loop structure.

C-4.55) Number each bottle and think about the binary expansion of each bottle's number.

C-4.56) Use an auxiliary array that keeps counts for each value.

C-4.57) You might wish to use an auxiliary array of size at most $4n$.

C-4.58) Argue why you have to look at all the integers in A.

C-4.59) Start out by finding the integer in A with maximum absolute value.

Projects

P-4.60) Choose representative values of the input size n, and run at least 5 tests for each size value n.

P-4.61) Try to reuse your code as much as possible.

P-4.62) You should try several runs over many different problem sizes.

P-4.63) Do a type of "binary search" to determine the maximum effective value of n for each algorithm.