

---

# 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

---

## Instructor's Solutions Manual

WILEY

# Chapter

# 3

## Algorithm Analysis

### Hints and Solutions

#### Reinforcement

**R-3.1) Hint** Use powers of two as your values for  $n$ .

**R-3.2) Hint** 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-3.2) Solution** Setting the two equations equal and simplifying, we see that the cross-over occurs at  $n = 4 \log n$ . Now, we can confirm that the cross-over occurs at  $n = 2^4 = 16$ .

**R-3.3) Hint** Set both sides equal to each other to determine this.

**R-3.3) Solution** Setting the two sides equal and simplifying confirms that the cross-over occurs at  $n = 20$ . That is,  $40n^2 \leq 2n^3$  for  $n \geq 20$ .

**R-3.4) Hint** Any growing function will have a “flatter” curve on a log-log scale than it has on a standard scale.

**R-3.4) Solution** The constant function.

**R-3.5) Hint** Think of another way to write  $\log n^c$ .

**R-3.5) Solution** It is because  $\log n^c = c \log n$ .

**R-3.6) Hint** Characterize this in terms of the sum of all integers from 1 to  $n$ .

**R-3.6) Solution** If this sum is  $E(n)$ , then it is clearly equal to  $2S(n)$ , where  $S(n)$  is the sum of all integers from 1 to  $n$ ; hence,  $E(n) = n(n+1)$ .

**R-3.7) Hint** Use the fact that if  $a < b$  and  $b < c$ , then  $a < c$ .

**R-3.7) Solution** If  $cf(n)$  is an upper bound on the running time, for some constant  $c$ , for all inputs, then this must also be an upper bound on the worst-case time.

**R-3.8) Hint** Simplify the expressions, and then use the ordering of the seven important algorithm-analysis functions to order this set.

**R-3.8) Solution**

$$2^{10}, 2^{\log n}, 3n + 100 \log n, 4n, n \log n, 4n \log n + 2n, n^2 + 10n, n^3, 2^n$$

**R-3.9) Hint** Review the definition of big-Oh and use the constant from this definition.

**R-3.9) Solution** There are constants  $c$  and  $n_0$  such that  $d(n) \leq cf(n)$  for  $n \geq n_0$ . Thus,  $ad(n) \leq acf(n)$  for  $n \geq n_0$ .

**R-3.10) Hint** Start with the product and then apply the definition of the big-oh for  $d(n)$  and then  $e(n)$ .

**R-3.10) Solution** We have, by definition that  $d(n) \leq c_1f(n)$  for  $n \geq n_1$ , and  $e(n) \leq c_2g(n)$  for  $n \geq n_2$ . Thus, for  $n \geq \max\{n_1, n_2\}$ ,

$$d(n)e(n) \leq c_1f(n)e(n) \leq c_1c_2f(n)g(n).$$

**R-3.11) Hint** Use the definition of the big-oh and add the constants (but be sure to use the right  $n_0$ ).

**R-3.12) Hint** You need to give a counterexample. Try the case when  $d(n)$  and  $e(n)$  are both  $O(n)$  and be specific.

**R-3.13) Hint** 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-3.14) Hint** First show that the max is always less than the sum.

**R-3.15) Hint** Simply review the definitions of big-oh and big-omega. This one is easy.

**R-3.16) Hint** Recall that  $\log n^k = k \log n$ .

**R-3.17) Hint** Notice that  $(n+1) \leq 2n$  for  $n \geq 1$ .

**R-3.17) Solution** By the definition of big-Oh, we need to find a real constant  $c > 0$  and an integer constant  $n_0 \geq 1$  such that  $(n+1)^5 \leq c(n^5)$  for every integer  $n \geq n_0$ . Since  $(n+1) \leq 2n$  for  $n \geq 1$ , we have that  $(n+1)^5 \leq (2n)^5 = 32n^5 = c(n^5)$  for  $c = 32$  and  $n \geq n_0 = 1$ .

**R-3.18) Hint**  $2^{n+1} = 2 \cdot 2^n$ .

**R-3.18) Solution** By the definition of big-Oh, we need to find a real constant  $c > 0$  and an integer constant  $n_0 \geq 1$  such that  $2^{n+1} \leq c(2^n)$  for  $n \geq n_0$ . One possible solution is choosing  $c = 2$  and  $n_0 = 1$ , since  $2^{n+1} = 2 \cdot 2^n$ .

**R-3.19) Hint** Make sure you don't get caught by the fact that  $\log 1 = 0$ .

**R-3.19) Solution**  $n \leq n \log n$  for  $n \geq 2$  (but this is not true for  $n = 1$ ).

**R-3.20) Hint** Use the definition of big-omega, but don't get caught by the fact that  $\log 1 = 0$ .

**R-3.21) Hint** Use the definition of big-omega, but don't get caught by the fact that  $\log 1 = 0$ .

**R-3.21) Solution** By the definition of big-Omega, we need to find a real constant  $c > 0$  and an integer constant  $n_0 \geq 1$  such that  $n \log n \geq cn$  for  $n \geq n_0$ . Choosing  $c = 1$  and  $n_0 = 2$ , shows  $n \log n \geq cn$  for  $n \geq n_0$ , since  $\log n > \geq 1$  in this range.

**R-3.22) Hint** If  $f(n)$  is a positive nondecreasing function that is always greater than 1, then  $\lceil f(n) \rceil \leq f(n) + 1$ .

**R-3.23) Hint** Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

**R-3.23) Solution** The example1 function runs in  $O(n)$  time.

**R-3.24) Hint** Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

**R-3.24) Solution** The example2 function runs in  $O(n)$  time.

**R-3.25) Hint** 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-3.25) Solution** The example3 function runs in  $O(n^2)$  time.

**R-3.26) Hint** 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-3.26) Solution** The example4 function runs in  $O(n)$  time.

**R-3.27) Hint** 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-3.27) Solution** The example5 function runs in  $O(n^3)$  time.

**R-3.28) Hint** 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-3.28) Solution** The numbers in the first row are quite large. The table below calculates it approximately in powers of 10. People might also choose to use powers of 2. Being close to the answer is enough for the big numbers (within a few factors of 10 from the answers shown).

	1 Second	1 Hour	1 Month	1 Century
$\log n$	$2^{10^6} \approx 10^{300000}$	$2^{3.6 \times 10^9} \approx 10^{10^9}$	$2^{2.6 \times 10^{12}} \approx 10^{0.8 \times 10^{12}}$	$2^{3.1 \times 10^{15}} \approx 10^{10^{15}}$
$n$	$10^6$	$3.6 \times 10^9$	$\approx 2.6 \times 10^{12}$	$\approx 3.12 \times 10^{15}$
$n \log n$	$\approx 10^5$	$\approx 10^9$	$\approx 10^{11}$	$\approx 10^{14}$
$n^2$	1000	$6 \times 10^4$	$\approx 1.6 \times 10^6$	$\approx 5.6 \times 10^7$
$n^3$	100	$\approx 1500$	$\approx 14000$	$\approx 1500000$
$2^n$	19	31	41	51

**R-3.29) Hint** The  $O(\log n)$  calculation is performed  $n$  times.

**R-3.30) Hint** The  $O(n)$  calculation is performed  $\log n$  times.

**R-3.31) Hint** Consider the cases when all entries of  $S$  are even or odd.

**R-3.32) Hint** First characterize the running time of Algorithm D using a summation.

**R-3.32) Solution** The running time of Algorithm D is proportional to  $\sum_{i=1}^n i$ , which is  $O(n^2)$ .

**R-3.33) Hint** Discuss how the definition of the big-oh fits into Al's claim.

**R-3.33) Solution** To say that Al's algorithm is "big-oh" of Bill's algorithm implies that Al's algorithm will run faster than Bill's for all input greater than some nonzero positive integer  $n_0$ . In this case,  $n_0 = 100$ .

**R-3.34) Hint** Recall the definition of the Harmonic number,  $H_n$ .

## Creativity

**C-3.35) Hint** Use sorting as a subroutine.

**C-3.35) Solution** Assuming we have all three sets stored in sequences, combine all three sequences into one sequence. Next, sort this combined sequence and look to see if any element is repeated three times. If so, the three original sets are not disjoint.

**C-3.36) Hint** Note that 10 is a constant!

**C-3.36) Solution** Since 10 is a constant, we can solve this problem for any size sequence in  $O(n)$  time. We can begin by looping to find the largest element, and then record the index of that element in an auxiliary list of size 10. Then we find the next largest element with another loop, making sure to ignore the previously found element, and recording the index of the second largest element. Each such loop requires  $O(n)$  time; the time to check if an index has already been used can be done in  $O(1)$  time as there are  $O(1)$  entries in the auxiliary list. Therefore the overall time is  $O(10 \cdot n)$  which is  $O(n)$ .

**C-3.37) Hint** Think of a function that grows and shrinks at the same time without bound.

**C-3.37) Solution** One possible solution is  $f(n) = n^2 \cdot (1 + \sin(n))$ .

**C-3.38) Hint** Use induction, a visual proof, or bound the sum by an integral.

**C-3.38) Solution**

$$\sum_{i=1}^n i^2 < \int_0^{n+1} x^2 dx < \frac{(n+1)^3}{3} = O(n^3)$$

**C-3.39) Hint** 1

**C-3.39) Solution** Let  $S = \sum_{i=1}^n i/2^i < 2$ . Note that  $S = (\sum_{i=1}^n 1/2^i) + (\sum_{i=2}^n (i-1)/2^i) = (\sum_{i=1}^n 1/2^i) + (\sum_{i=1}^{n-1} i/2^{i+1}) < 1 + \frac{S}{2}$ . Since  $S < 1 + \frac{S}{2}$ ,  $\frac{S}{2} < 1$  and thus  $S < 2$ .

**C-3.40) Hint** Use the log identity that translates  $\log bx$  to a logarithm in base 2.

**C-3.40) Solution**  $\log_b f(n) = \log f(n) / \log b$ , but  $1/\log b$  is a constant.

**C-3.41) Hint** 1

**C-3.41) Solution** Pair up all the items and compare them, producing a set of candidate minima and candidate maxima. Now with  $n/2 - 1$  comparisons we can find the minimum of the minima and the maximum of the maxima. The total number of comparisons is  $3n/2 - 2$ .

**C-3.42) Hint** Consider the sum of the maximum number of visits each friend can make without visiting his/her maximum number of times.

**C-3.42) Solution** The number is one more than the total number of visits each friend can make while still being able to make one more allowed visit, that is, the sum where each friend  $i$  visits  $i - 1$  times. In other words, the minimum value for  $C$  such that Bob should know that one of his friends has visited his/her maximum allowed number of times is  $n(n - 1)/2 + 1$ .

**C-3.43) Hint** You need to line up the columns a little differently.

**C-3.44) Hint** Characterize the number of bits needed first.

**C-3.44) Solution** Since  $r$  is represented with 100 bits, any candidate  $p$  that the eavesdropper might use to try to divide  $r$  uses also at most 100 bits. Thus, this very naive algorithm requires  $2^{100}$  divisions, which would take about  $2^{80}$  seconds, or at least  $2^{55}$  years. Even if the eavesdropper uses the fact that a candidate  $p$  need not ever be more than 50 bits, the problem is still difficult. For in this case,  $2^{50}$  divisions would take about  $2^{30}$  seconds, or about 34 years.

Since each division takes time  $O(n)$  and there are  $2^{4n}$  total divisions, the asymptotic running time is  $O(n \cdot 2^{4n})$ .

**C-3.45) Hint** Consider computing a function of the integers in  $S$  that will immediately identify which one is missing.

**C-3.45) Solution** First calculate the sum  $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ . Then calculate the sum of all values in the sequence  $A$ . The missing element is the difference between these two numbers.

**C-3.46) Hint** Consider the first induction step.

**C-3.46) Solution** The induction assumes that the set of  $n - 1$  sheep without  $a$  and the set of  $n - 1$  sheep without  $b$  have a sheep in common. Clearly this is not true for  $n = 2$  sheep. If a base case of 2 sheep could be shown, then the induction would be valid.

**C-3.47) Hint** Consider the contribution made by one line.

**C-3.48) Hint** Look carefully at the definition of big-Oh and rewrite the induction hypothesis in terms of this definition.

**C-3.49) Hint** Use the definition of big-omega, and make  $n = 1$  and  $n = 2$  your base cases.

**C-3.49) Hint** Use the definition of big-Omega, and make  $n = 1$  and  $n = 2$  your base cases.

**C-3.50) Hint** Consider writing a pseudo-code description of this algorithm and note its loop structure.

**C-3.51) Hint** Try to bound from above each term in this summation.

**C-3.51) Solution**

$$\sum_{i=1}^n \log_2 i \leq \sum_{i=1}^n \log_2 n = n \log_2 n.$$

**C-3.52) Hint** Try to bound a significant number of the terms from below.

**C-3.52) Solution** For convenience assume that  $n$  is even. Then

$$\sum_{i=1}^n \log_2 i \geq \sum_{i=n/2}^n \log_2 i \geq \sum_{i=n/2}^n \log_2 n/2 = \sum_{i=n/2}^n (\log_2 n) - 1 =$$

$$(n/2 + 1) \log_2 n - n/2 = \frac{1}{4} n \log_2 n + \left( \frac{1}{4} n \log_2 n + \log_2 n - n/2 \right) \geq \frac{1}{4} n \log n.$$

for  $n \geq 4$ .

**C-3.53) Hint** Number each bottle and think about the binary expansion of each bottle's number.

**C-3.53) Solution** Number each bottle from 1 to  $n$ . Select  $\lceil \log n \rceil$  tasters and map each taster to a bit. On the first day of the month, a taster samples a wine if, in the binary representation of the wine's number, his bit is 1. For example, if taster  $A$  is assigned to the lowest order bit and there are 5 bottles, he will sample bottles 1, 3, and 5. If taster  $B$  is assigned to the highest order bit, he will sample bottles 4 and 5.

After the month is over, the number of the poisoned bottle can be determined. If a taster dies, then the bit they mapped to is a 1 in the poisoned bottle's number. Otherwise, the bit is a 0.

**C-3.54) Hint** Use an auxiliary array that keeps counts for each value.

---

## Projects

**P-3.55) Hint** Choose representative values of the input size  $n$ , and run at least 5 tests for each size value  $n$ .

**P-3.56) Hint** Try to reuse your code as much as possible.

**P-3.57) Hint** You should try several runs over many different problem sizes.

**P-3.58) Hint** Do a type of “binary search” to determine the maximum effective value of  $n$  for each algorithm.