Chapter 3 Exercises

Termanteus

2019-08-01

Contents

1 R-3.1 . . . . . **4**

2 R-3.2 . . . . . **5**

3 R-3.3 . . . . . **6**

4 R-3.4 . . . . . **7**

5 R-3.5 . . . . . **7**

6 R-3.6 . . . . . **7**

7 R-3.7 . . . . . **7**

8 R-3.8 . . . . . **8**

9 R-3.9 . . . . . **8**

10 R-3.10 . . . . . **8**

11 R-3.11 . . . . . **8**

12 R-3.12 . . . . . **9**

13 R-3.13 . . . . . **9**

14 R-3.14 . . . . . **9**

15 R-3.15 . . . . . **9**

16 R-3.16 . . . . . **10**

17 R-3.17 . . . . . **10**

18 R-3.18 . . . . . **10**

19 R-3.19 . . . . . **10**

20 R-3.20 . . . . . **10**

21 R-3.21 . . . . . **10**

22 R-3.22 . . . . . **11**

23 R-3.23 . . . . . **13**

24 R-3.24 . . . . . **13**

25 R-3.25 . . . . . **13**

26 R-3.26 . . . . . **14**

27 R-3.27 . . . . . **14**

28 R-3.28 . . . . . **15**

29 R-3.29 . . . . . **15**

30 R-3.30 . . . . . **16**

31 R-3.31 . . . . . **16**

32 R-3.32 . . . . . **16**

33 R-3.33 . . . . . **16**

34 R-3.34 . . . . . **17**

35 C-3.35 . . . . . **17**

36 C-3.36 . . . . . **17**

37 C-3.37 . . . . . **17**

38 C-3.38 . . . . . **18**

39 C-3.39 . . . . . **19**

40 C-3.40 . . . . . **21**

41 C-3.41 . . . . . **22**

42 C-3.42 . . . . . **23**

43 C-3.43 . . . . . **23**

44 C-3.44 . . . . . **24**

45 C-3.45 . . . . . **24**

46 C-3.46 . . . . . **25**

47 C-3.47 . . . . . **25**

48 C-3.48 . . . . . **25**

49 C-3.49 . . . . . **26**

50 C-3.50 . . . . . **27**

51 C-3.51 . . . . . **29**

52 C-3.52 . . . . . **29**

53 C-3.53 . . . . . **30**

54 C-3.54 . . . . . **30**

55 P-3.55-3.56 . . . . . **31**

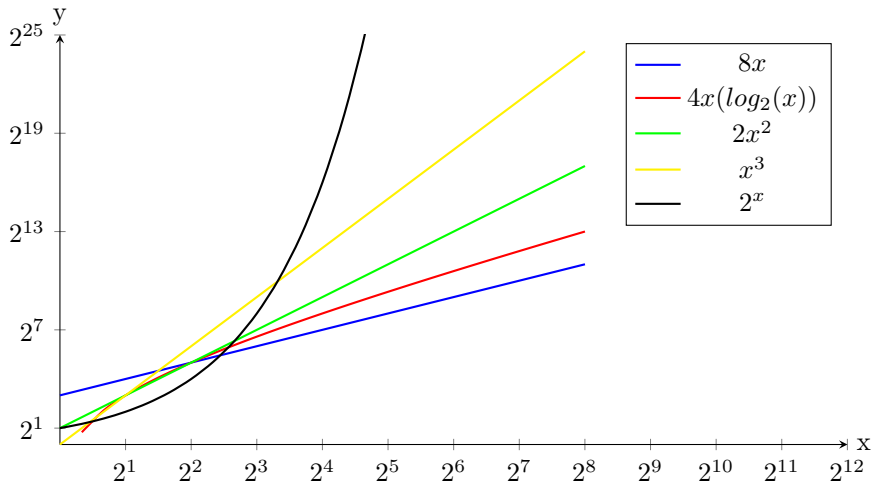
56 P-3.57 . . . . . **32**

57 C-3.54 . . . . . **33**

1. R-3.1

Statement Graph the functions $8n$, $4nlog(n)$, $2n^2$, n^3, and 2^n using a logarithmic scale for the x- and y-axes; that is, if the function value $f(n)$ is y, plot this as a point with x-coordinate at logn and y-coordinate at $log(y)$.

Solution Graph:



2. R-3.2

Statement The number of operations executed by algorithms A and B is $8nlog(n)$ and $2n^2$, respectively. Determine n_0 such that A is better than B for n $>= n_0$.

Solution Graph:

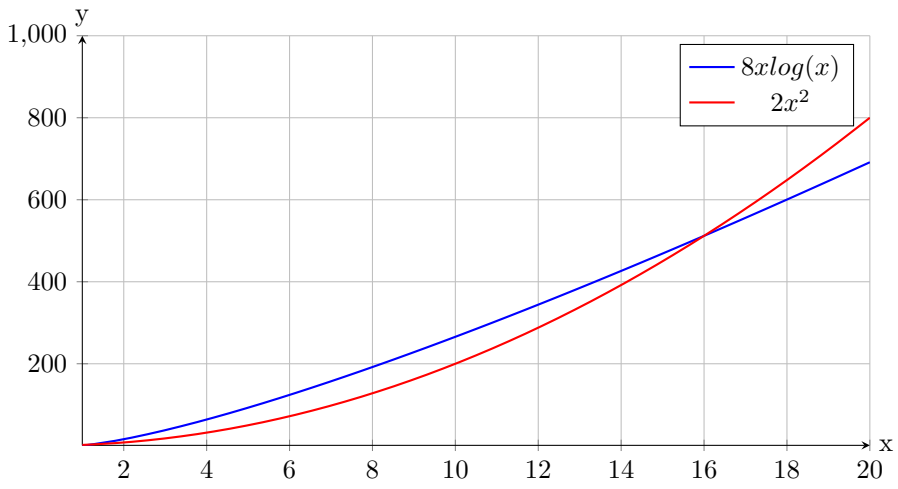


Figure 1: As you can see in the graph, at $n >= 16$ A is better than B.

3. R-3.3

Statement The number of operations executed by algorithms A and B is $40n^2$ and $2n^3$, respectively. Determine n_0 such that A is better than B for n >= n_0.

Solution Graph:

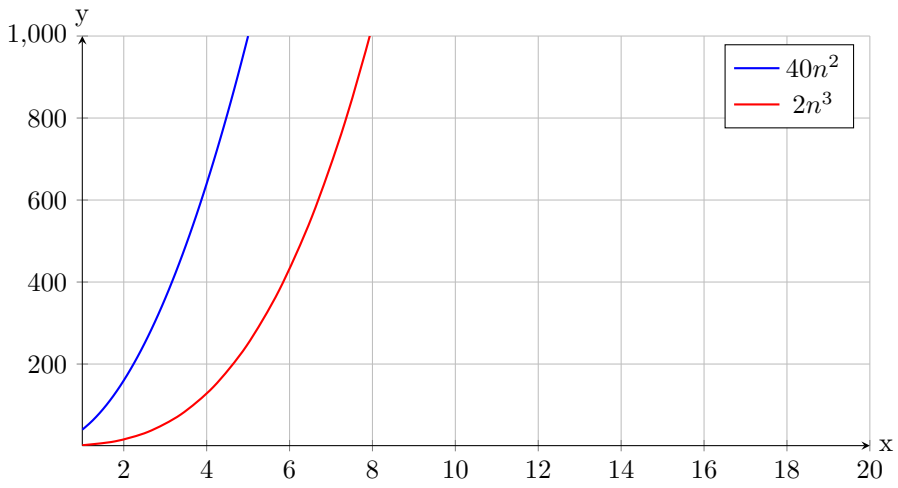


Figure 2: As you can see in the graph, at $n >= 0$ A is better than B.

4. R-3.4

Statement Give an example of a function that is plotted the same on a log-log scale as it is on a standard scale.

Solution $f(x) = const$

5. R-3.5

Statement Explain why the plot of the function nc is a straight line with slope c on a log-log scale.

Solution We have

$$\begin{aligned} & y = n^c \\ \Leftrightarrow & log(y) = log(n^c) \\ \Leftrightarrow & log(y) = clog(n) \end{aligned}$$

On a log-log scale graph, we can see $log(n)$ x in a standard graph, so it means that means the function above is a linear function, so the coefficent (c in this situation) is its slope.

6. R-3.6

Statement What is the sum of all the even numbers from 0 to 2n, for any positive integer n?

Solution

$$S = \frac{n(2+2n)}{2} = n(n+1) \tag{1}$$

7. R-3.7

Statement Show that the following two statements are equivalent:
(a) The running time of algorithm A is always $O(f(n))$.
(b) In the worst case, the running time of algorithm A is $O(f(n))$

Solution As the book said: "Therefore, for the remainder of this book, unless we specify otherwise, we will characterize running times in terms of the worst case, as a function of the input size, n, of the algorithm."

So basically the two sentences is the same.

8. R-3.8

Statement Order the following functions by asymptotic growth rate.

Solution

$$2^{10} < 4n < 3n + 100log(n) < nlog(n) < ...$$
$$... < 4nlog(n) + 2n < n^2 + 10 < 2^{log(n)} < 2^n$$

9. R-3.9

Statement Show that if $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.

Solution Because as the definition has state: $f(n) <= cg(n)$ for $n >= n_0$

10. R-3.10

Statement Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the product $d(n)e(n)$ is $O(f(n)g(n))$.

Solution Because as the definition has state: $f(n) <= cg(n)$ for $n >= n_0$
So we have

$$\begin{aligned} & d(n)e(n) <= c_1 f(n) * c_2 g(n), \quad n >= n_0 \\ \Leftrightarrow\ & d(n)e(n) <= c_0(f(n) * g(n)) \\ \Leftrightarrow\ & d(n)e(n) <= O(f(n) * g(n)) \end{aligned}$$

11. R-3.11

Statement Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the sum of $d(n) + e(n)$ is $O(f(n) + g(n))$.

Solution Because as the definition has state: $f(n) <= cg(n)$ for $n >= n_0$
So we have:

$$\begin{aligned} & d(n)e(n) <= c_1 f(n) + c_2 g(n), \quad n >= n_0 \\ \Leftrightarrow\ & d(n)e(n) <= c_0(f(n) + g(n)) \\ \Leftrightarrow\ & d(n)e(n) <= O(f(n) + g(n)) \end{aligned}$$

12. R-3.12

Statement Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)-e(n)$ is **not necessarily** $O(f(n)-g(n))$.

Solution Assume the statement is true:
For example: $d(n) = n^2 = O(n^2)$ and $e(n) = 3n^2 = O(n^2)$
Then: $d(n) - e(n) = O(n^2 - n^2) = O(0) \rightarrow$ **False**

13. R-3.13

Statement Show that if $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.

Solution Based on the definition, we will have:

$$\begin{cases} d(n) & <= c_0 f(n) \\ f(n) & <= c_1 g(n) \end{cases}$$
$$\Rightarrow d(n) <= c_2 g(n)$$
$$\Leftrightarrow d(n) = O(g(n))$$

14. R-3.14

Statement Show that $O(max f(n), g(n)) = O(f(n) + g(n))$.

Solution Based on excercise **3.11** it is proved to be true

15. R-3.15

Statement Show that $f(n)$ is $O(g(n))$ if and only if $g(n)$ is $\Omega(f(n))$.

Solution Based on **Big-Omega's definition** and **Big-O's definition** it is proved to be true.

16. R-3.16

Statement Show that if $p(n)$ is a polynomial in n, then $log(p(n))$ is $O(log(n))$.

Solution

$$
\begin{aligned}
& p(n) = \sum_{k=0}^{n} a_k x^k \\
\Leftrightarrow\ & log(p(n)) = log(\sum_{k=0}^{n} a_k x^k) \\
\Leftrightarrow\ & log(p(n)) = log(a_0 x^0) + log(a_1 x^1) + ... + log(a_n x^n) \\
\Leftrightarrow\ & log(p(n)) = 0 * log(a_0 x) + 1 * log(a_1 x^1) + ... + n * log(a_n x) \\
\Leftrightarrow\ & log(p(n)) = O(log(n))
\end{aligned}
$$

17. R-3.17

Statement Show that $(x+1)^5$ is $O(n^5)$.

Solution

$$
\begin{aligned}
& f(x) = x^5 + 5 * x^4 + 10 * x^3 + 10 * x^2 + 5 * x + 1 \\
\Leftrightarrow\ & f(x) = O(n^5 0)
\end{aligned}
$$

18. R-3.18

Skip because easy...

19. R-3.19

Skip because easy...

20. R-3.20

Skip because easy...

21. R-3.21

Skip because easy...

22. R-3.22

Statement Show that $\left\lceil f(n) \right\rceil$ is $O(f(n))$, if $f(n)$ is a positive nondecreasing function that is always greater than 1.

Solution If $f(n)$ is a positive nondecreasing function that is always greater than 1, then $\left\lceil f(n) \right\rceil <= f(n) + 1$.

```
# Copyright 2013, Michael H. Goldwasser
#
# Developed for use with the book:
#
#    Data Structures and Algorithms in Python


def example1(S):
    n = len(S)
    total = 0
    for j in range(n):  # loop from 0 to n-1
        total += S[j]
    return total


def example2(S):
    n = len(S)
    total = 0
    for j in range(0, n, 2):  # note the increment of 2
        total += S[j]
    return total


def example3(S):
    n = len(S)
    total = 0
    for j in range(n):  # loop from 0 to n-1
        for k in range(1 + j):  # loop from 0 to j
            total += S[k]
    return total


def example4(S):
    n = len(S)
    prefix = 0
    total = 0
    for j in range(n):
        prefix += S[j]
        total += prefix
    return total


def example5(A, B):
    n = len(A)
    count = 0
    for i in range(n):  # loop from 0 to n-1
        total = 0
        for j in range(n):  # loop from 0 to n-1
            for k in range(1 + j):  # loop from 0 to j
```

```
50                     total += A[k]
51             if B[i] == total:
52                 count += 1
53         return count
```

23. R-3.23

Statement Give a big-Oh characterization, in terms of n, of the running time of the example1 function shown in Code Fragment 3.10.

Solution

- Loop through n items $\rightarrow$ $O(n)$.
- Each loop do constant $O(1)$ time.
- **Total:** $O(n)$

24. R-3.24

Statement Give a big-Oh characterization, in terms of n, of the running time of the example2 function shown in Code Fragment 3.10.

Solution

- Loop through $\lceil n/2 \rceil$ items $\rightarrow$ $O(n)$.
- Each loop do constant $O(1)$ time.
- **Total:** $O(n)$

25. R-3.25

Statement Give a big-Oh characterization, in terms of n, of the running time of the example3 function shown in Code Fragment 3.10.

Solution

- Loop through $(1 * 2 * 3 * ... * (n-1) * n)$ items $\rightarrow$ $O(n^2)$.
- Each loop do constant $O(1)$ time.
- **Total:** $O(n^2)$

26. R-3.26

Statement Give a big-Oh characterization, in terms of n, of the running time of the example4 function shown in Code Fragment 3.10.

Solution

- Loop through n items $\rightarrow$ $O(n)$.
- Each loop do constant $O(1)$ time.
- **Total:** $O(n)$

27. R-3.27

Statement Give a big-Oh characterization, in terms of n, of the running time of the example5 function shown in Code Fragment 3.10.

Solution

- Loop through $n * n * (n + 1)/2$ items $\rightarrow$ $O(n^3)$.
- Each loop do constant $O(1)$ time.
- **Total:** $O(n^3)$

28. R-3.28

Statement For each function $f(n)$ and time t in the following table determine the largest size n of a problem P that can be solved in time t if the algorithm for solving P takes $f(n)$ microseconds (one entry is already completed).

Solution We have:

- 1 second = 10^6 microseconds
- 1 hour = $3.6 * 10^9$ microseconds
- 1 month = $2.592 * 10^12$ microseconds
- 1 century = $3.1104 * 10^15$ microseconds

And we also know that $f(n)$ microseconds will be taken for solving problem P of size n. So the maximum problems we can solve for $f(n) = log(n)$ in 1 second is such that:

$$\begin{aligned} log(n) &<= 10^6 \\ 2^{log(n)} &<= 2^{10^6} \\ n &<= 10^{300000}, \quad 2^{10} \approx 10^3 \end{aligned}$$

	1 Second	**1 Hour**	**1 Month**	**1 Century**
$logn$	$\approx 10^{300000}$	$\approx 10^{1080000000}$	$\approx 10^{7.776*10^{11}}$	$\approx 10^{9.3312*10^{14}}$
$\sqrt{n}$	10^{12}	$\approx 12.96 * 10^{18}$	$\approx 6.718464 * 10^{24}$	$\approx 9.67458816 * 10^{30}$
n	$\approx 10^6$	$\approx 36 * 10^8$	$\approx 746496 * 10^{16}$	$\approx 995827586973696 * 10^{16}$
$nlogn$	62746	133378058	71870856404	68654697441062
n^2	1000	60000	1609968	56175382
2^n	19	31	41	51

29. R-3.29

Statement Algorithm A executes an $O(logn)$-time computation for each entry of an n-element sequence. What is its worst-case running time?

Solution $T = O(nlog(n))$

30. R-3.30

Statement Given an n-element sequence S, Algorithm B chooses $logn$ elements in S at random and executes an $O(n)$-time calculation for each. What is the worst-case running time of Algorithm B?

Solution Assuming picking random index for each element cost constant $O(1)$ time. In such case: $T = O(nlogn)$

31. R-3.31

Statement Given an n-element sequence S of integers, Algorithm C executes an $O(n)$-time computation for each even number in S, and an $O(logn)$-time computation for each odd number in S. What are the best-case and worstcase running times of Algorithm C?

Solution **Best case:** All element in S is odd: $T = O(nlogn)$
Worst case: All element in S is even: $T = O(n^2)$

32. R-3.32

Statement Given an n-element sequence S, Algorithm D calls Algorithm E on each element $S[i]$. Algorithm E runs in $O(i)$ time when it is called on element $S[i]$. What is the worst-case running time of Algorithm D?

Solution $T = O(n)$

33. R-3.33

Statement Al and Bob are arguing about their algorithms. Al claims his $O(nlogn)$-time method is always faster than Bob's $O(n^2)$-time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm runs faster, and only when $n >= 100$ is the $O(nlogn)$-time one better. Explain how this is possible

Solution Based on the definition $f(n) = O(g(n))$ iff $f(n) <= cg(n),\ n >= n_0$. So probably for $n < 100$ the $O(nlogn)$ is not held.

34. R-3.34

Statement There is a well-known city (which will go nameless here) whose inhabitants have the reputation of enjoying a meal only if that meal is the best they have ever experienced in their life. Otherwise, they hate it. Assuming meal quality is distributed uniformly across a person's life, describe the expected number of times inhabitants of this city are happy with their meals?

Solution "If the sequence is given to us in random order, the probability that the j^{th} element is the largest of the first j elements is $\frac{1}{j}$ (assuming uniqueness). Hence the expected number of times we see the current biggest (including initialization) is $H_n = \sum_{j=1}^{n} \frac{1}{j}$, which is known as the n^{th} **Harmonic number**."

35. C-3.35

Statement Assuming it is possible to sort n numbers in O(nlogn) time, show that it is possible to solve the three-way set disjointness problem in $O(nlogn)$-time.

Solution Sort: $O(nlogn)$
Binary Search: $O(logn)$ to search a element in the other two.
T= $O(nlogn)$

36. C-3.36

Statement Describe an efficient algorithm for finding the ten largest elements in a sequence of size n. What is the running time of your algorithm?

Solution Sort: $O(nlogn)$
T= $O(nlogn)$

37. C-3.37

Statement Give an example of a positive function $f(n)$ such that $f(n)$ is neither $O(n)$ nor $\Omega(n)$.

Solution $f(n) = 0 * n$

38. C-3.38

Statement Show that $\sum_{i=1}^{n} i^2$ is $O(n^3)$

Solution

$$S = \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + ... + n^2 \tag{2}$$

We gonna use Integral Bound for (2) because (2) is a weakly increasing series. For

$$\begin{aligned} I &= \int_1^n f(x)dx \\ I &= \int_1^n x^2 dx \\ I &= \frac{x^3}{3}\Big|_1^n \\ I &= \frac{n^3}{3} - \frac{1}{3} \end{aligned}$$

We will have:

$$\begin{aligned} I + f(1) <= S <= I + f(n) \\ S <= \frac{n^3}{3} - \frac{1}{3} + n^2 \\ S = O(n^3) \end{aligned}$$

39. C-3.39

Statement Show that

$$\sum_{i=1}^{n} \frac{i}{2^i} < 2 \tag{3}$$

Solution

$$\textbf{Let: } S(x) = \sum_{i=1}^{n} ix^i \tag{4}$$

1. We first prove (4) is a convergent series:

   - **Radius of convergent:** $(-1, 1)$, prove:

     $$l = \lim_{n \to +\infty} \sqrt[n]{|n|} = 1$$
     $$r = \frac{1}{l} = 1$$

     For x = 1:

     $$S(1) = \sum_{i=1}^{n} n$$

     →Divergent Series with x = 1.

     For x = -1:

     $$S(1) = \sum_{i=1}^{n} n * (-1)^n$$

     →Divergent Series with x = -1.

     Conclude:

     $$x \in (-1, 1)$$

- **Find sum of $S(x)$:**

  For $\forall x \in (-1, 1)$:

$$
\begin{aligned}
S(x) &= \sum_{i=1}^{n} i x^i \\
S(x) &= x * \sum_{i=1}^{n} i x^{i-1} \\
S(x) &= x * \sum_{i=1}^{n} (x^i)' \\
S(x) &= x * \Big(\sum_{i=1}^{n} x^i \Big)' \\
S(x) &= x * \Big(\frac{x}{1-x} \Big)' \\
S(x) &= \frac{x}{(1-x)^2}
\end{aligned}
$$

  Replace $x = \frac{1}{2}, we have$:

$$
S\Big(\frac{1}{2}\Big) = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2
$$

40. C-3.40

Statement Show that $log_b f(n)$ is $\Theta(log f(n))$ if $b > 1$ is a constant.

Solution

We have basic logarithmic formula:

$$log_a b = \frac{log_c b}{log_c a}$$

Similarly we apply to $log_b f(n)$:

$$log_b f(n) = \frac{log_2 f(n)}{log_2 b}$$
$$log f(n) = log_b f(n) log b$$

Apply the Big-Theta definition, we have this statement proved.

41. C-3.41

Statement Describe an algorithm for finding both the minimum and maximum of n numbers using fewer than $\frac{3n}{2}$ comparisons. (Hint: First, construct a group of candidate minimums and a group of candidate maximums.)

Solution Source code:

```
def findMinMax(arr):
    big = []
    small = []
    max = min = arr[0]
    for i in range(1, len(arr) + 1, 2):
        if i == len(arr):
            big.append(arr[i - 1])
            break
        larger, smaller = (
            (arr[i - 1], arr[i])
            if arr[i] >= arr[i]
            else
            (arr[i], arr[i - 1])
        )
        big.append(larger)
        small.append(smaller)
    for i in range(0, len(big)):
        if big[i] > max:
            max = big[i]
    for i in range(0, len(small)):
        if small[i] < min:
            min = small[i]
    return max, min
```

42. C-3.42

Statement Bob built a Web site and gave the URL only to his n friends, which he numbered from 1 to n. He told friend number i that he/she can visit the Web site at most i times. Now Bob has a counter, C, keeping track of the total number of visits to the site (but not the identities of who visits). What is the minimum value for C such that Bob can know that one of his friends has visited his/her maximum allowed number of times?

Solution For everyone (except the first one) visit until they only have one left, the total visited times will be:

$$S = 1 + 2 + ... + n - 1$$
$$S = \frac{(n-1)(n-1+1)}{2}$$
$$S = \frac{(n-1)n}{2}$$

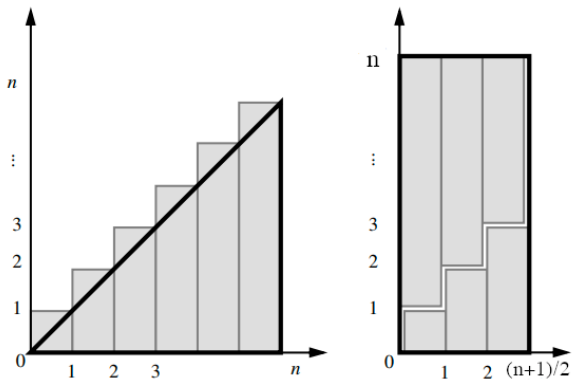
At this time, whoever watch will reach his/her limit, and C will know. So the minimun value for C is:

$$S = \frac{(n-1)n}{2} + 1$$

43. C-3.43

Statement Draw a visual justification of Proposition 3.3 analogous to that of Figure 3.3(b) for the case when n is odd.

Solution



(a) Proposition 3.3 when n is odd.

44. C-3.44

Statement Communication security is extremely important in computer networks, and one way many network protocols achieve security is to encrypt messages. Typical cryptographic schemes for the secure transmission of messages over such networks are based on the fact that no efficient algorithms are known for factoring large integers. Hence, if we can represent a secret message by a large prime number p, we can transmit, over the network, the number $r = p * q$, where $q > p$ is another large prime number that acts as the encryption key. An eavesdropper who obtains the transmitted number r on the network would have to factor r in order to figure out the secret message p.

Using factoring to figure out a message is very difficult without knowing the encryption key q. To understand why, consider the following naive factoring algorithm:

```
1     for p in range(2,r):
2       if r % p == 0: # if p divides r
3     return "The secret message is p!"
4
```

Solution a) For a message r has 100 bits, the worst case is when $p = 2^{100/2} = 2^{50}$ (because $r = p * q$ and $q > p$). So to decipher message r, this algorithm would have to take 2^{50} microseconds $\approx$ 35 years

b) Based on the answer of question a), we have known that the worst scenario is such that $p = 2^{m/2}, m = n * 8$,with m is the number of bits representing r. So we have to perform 2^{4n} divisions to find p. With the complexity of $O(n)$ for each division, the total running time will be $O(n2^{4n})$.

45. C-3.45

Statement A sequence S contains $n-1$ unique integers in the range [0, $n-1$], that is, there is one number from this range that is not in S. Design an $O(n)$-time algorithm for finding that number. You are only allowed to use $O(1)$ additional space besides the sequence S itself.

Solution Sort, then run from 0 to n-1, if the current number is not equal previous number + 1, then previous number + 1 is the one we need to find.

46. C-3.46

Statement Al says he can prove that all sheep in a flock are the same color:
Base case: One sheep. It is clearly the same color as itself.
Induction step: A flock of n sheep. Take a sheep, a, out. The remaining $n-1$ are all the same color by induction. Now put sheep a back in and take out a different sheep, b. By induction, the $n-1$ sheep (now with a)are all the same color. Therefore, all the sheep in the flock are the same color. What is wrong with Al's "justification"?

Solution Al makes an implicit assumption that the set n+1 has at least 3 sheeps. This is not true when n + 1 = 2 sheeps.

47. C-3.47

Statement Let S be a set of n lines in the plane such that no two are parallel and no three meet in the same point. Show, by induction, that the lines in S determine $\Theta(n^2)$ intersection points.

Solution *Base case:* $(n = 0)$: 0 lines and 0 intersection
Induction step: $(n > 0)$: Assume claim true for n lines. When add one more line to a plane with n lines, since among lines, no two are parallel and no three meet in the same point, it will add n more intersect. So:
The total intersect with n+1 lines = currentIntersect + n, or
The total intersect with n+1 lines = $\Theta(n^2) + O(n) = \Theta(n^2)$
So this holds for n+1 lines while n is true, **proved**.

48. C-3.48

Statement Consider the following "justification" that the Fibonacci function, $F(n)$ (see Proposition 3.20) is $O(n)$:
Base case: $(n <= 2)$: $F(1) = 1$ and $F(2) = 2$.
Induction step: $(n > 2)$: Assume claim true for $n' < n$. Consider n. $F(n) = F(n-2)+F(n-1)$. By induction, $F(n-2)$ is $O(n-2)$ and $F(n-1)$ is $O(n-1)$. Then, $F(n)$ is $O((n-2)+(n-1))$, by the identity presented in Exercise R-3.11. Therefore, $F(n)$ is $O(n)$.
What is wrong with this "justification"?

Solution This "justification" is equivalent to proving a number is $O(1)$. So whenever you plug in a value of n to the function $F(n)$, you turn it into a constant, not a function anymore.

49. C-3.49

Statement Consider the Fibonacci function, $F(n)$ (see Proposition 3.20). Show by induction that $F(n)$ is $\Omega\left(\left(\frac{3}{2}\right)^n\right)$.

Solution *Base case:* $(n = 0)$: 0 lines and 0 intersection
Induction step: $(n > 0)$: Assume claim true for n = k, which means:

$$F(k) <= c * \left(\frac{3}{2}\right)^n, \quad n >= n_0, c \in \mathrm{R}$$
$$F(k+1) = F(k) + F(k-1)$$
$$F(k+1) = \Omega\left(\left(\frac{3}{2}\right)^n\right) + \Omega\left(\left(\frac{3}{2}\right)^n\right)$$
$$F(k+1) = \Omega\left(\left(\frac{3}{2}\right)^n\right)$$

This mean that k+1 still holds if k is true, **proved**.

50. C-3.50

Statement Let $p(x)$ be a polynomial of degree n, that is, $p(x) = \sum_{i=0}^{n} a_i x^i$.

(a) Describe a simple $O(n^2)$-time algorithm for computing $p(x)$.

(b) Describe an $O(nlogn)$-time algorithm for computing $p(x)$, based upon a more efficient calculation of x^i.

(c) Now consider a rewriting of $p(x)$ as

$$p(x) = a_0 + x\Big(a_2 + x\big(a_3 + ... + x(a_{n-1} + xa_n)...\big)\Big)$$

which is known as **Horner's method**. Using the big-Oh notation, characterize the number of arithmetic operations this method executes

Solution For-loop conventions: For $i = 0$ to n, the loop will stop when $i = n - 1$. reverse(i=0 to n) will corresponding to n-1,n-2,...0

Algorithm 1: A simple $O(n^2)$-time algorithm for computing $p(x)$

```
1  function evaluatePolynomial (coefficents, x);
   Input  : Array of corresponding coefficent for each x_i,x
   Output: Sum of the polynomial
2  sum ⟵ 0;
3  size ⟵ len(coefficents);
4  for i = 0 to size do
5  |  power ← 1;
6  |  for j = 1 to i+1 do
7  |  |  power ← power * x;
8  |  end
9  |  sum ← coefficents[i] * power
10 end
11 return sum;
```

Algorithm 2: An improved $O(nlogn)$-time algorithm for computing $p(x)$, based upon a more efficient calculation of x^i.

```
1  function evaluatePolynomial (coefficents, x);
   Input  : Array of corresponding coefficent for each x_i,x
   Output: Sum of the polynomial
2  sum ⟵ 0;
3  size ⟵ len(coefficents);
4  for i = 0 to size do
5  |  if i is odd then
6  |  |  power ← x;
7  |  end
8  |  else
9  |  |  power ← 1;
10 |  end
11 |  for j = 0 to ⌈log(i)⌉ do
12 |  |  power ← power * x^2;
13 |  end
14 |  sum ← coefficents[i] * power
15 end
16 return sum;
```

Algorithm 3: Horner's method

```
1  function evaluatePolynomial (coefficents, x);
   Input  : Array of corresponding coefficent for each x_i, x
   Output: Sum of the polynomial
2  sum ⟵ 0;
3  for i=reverse(0 to size) do
4  |  sum ← coefficents[i] + x * sum
5  end
6  return sum;
```

Running time: O(n), also O(n) multiplication.

51. C-3.51

Statement Show that the summation $\sum_{i=1}^{n} log(i)$ is $O(nlogn)$.

Solution Let P(n): $\sum_{i=1}^{n} log(i) <= nlogn$
Base case: $P(1) = 0 <= 0$. **True**
Inductive step: Assume P(n) true, prove P(n+1) true:

$$\sum_{i=1}^{n+1} log(i) = \sum_{i=1}^{n} log(i) + log(n+1)$$

$$\sum_{i=1}^{n+1} log(i) <= nlog(n) + log(n+1), \quad \text{with}$$

$$nlog(n) + log(n+1) <= nlog(n+1) + log(n+1)$$

$$\sum_{i=1}^{n+1} log(i) <= (n+1)log(n+1)$$

So P(n+1) holds when P(n) is true. **Proved**.

Based on the definition of Big-Oh, we have $\sum_{i=1}^{n} log(i)$ is $O(nlogn)$

52. C-3.52

Statement Show that the summation $\sum_{i=1}^{n} log(i)$ is $\Omega(nlogn)$.

Solution We can see that $\sum_{i=1}^{n} log(i) >= \sum_{i=n/2}^{n} log(i)$
Also:

$$\sum_{i=n/2}^{n} log(i) >= \frac{n}{2} * log(\frac{n}{2})$$

$$\sum_{i=n/2}^{n} log(i) >= \frac{n}{2} * (log(n) - log(2))$$

$$\sum_{i=n/2}^{n} log(i) = \Omega(nlogn)$$

53. C-3.53

Statement An evil king has n bottles of wine, and a spy has just poisoned one of them. Unfortunately, they do not know which one it is. The poison is very deadly; just one drop diluted even a billion to one will still kill. Even so, it takes a full month for the poison to take effect. Design a scheme for determining exactly which one of the wine bottles was poisoned in just one month's time while expending $O(logn)$ taste testers.

Solution To represent number n in bits we must use $k = \lfloor log(n) \rfloor + 1$ bits

Then we will use k people to test n bottles, mark each person/bottle with number from $1 \to n$. Then the rule to determine who will try which bottle is this: **With the number of that person converted to binary number, who has the value 1 at the i^{th} bit will try the i^{th} bottle. If he/her dies alone, then the bottle of has ($2^{\textbf{that man's number}-1}$) is poison, if more than one dies, (OR of their number)th bottle has poison.**

For example: 8 bottles, 4 people:

0001
0010
0011
0100
0101
0110
0111
1000

Person 1: bottle 1,3,5,7; Person 2: 2,3,6,7; Person 3: 4,5,6,7; Person 4: 8

If P1 dies: 1; If P2 dies: 2; If P3 dies: 4;If P4 dies: 8

If P1,P2 dies: 3, and so on...

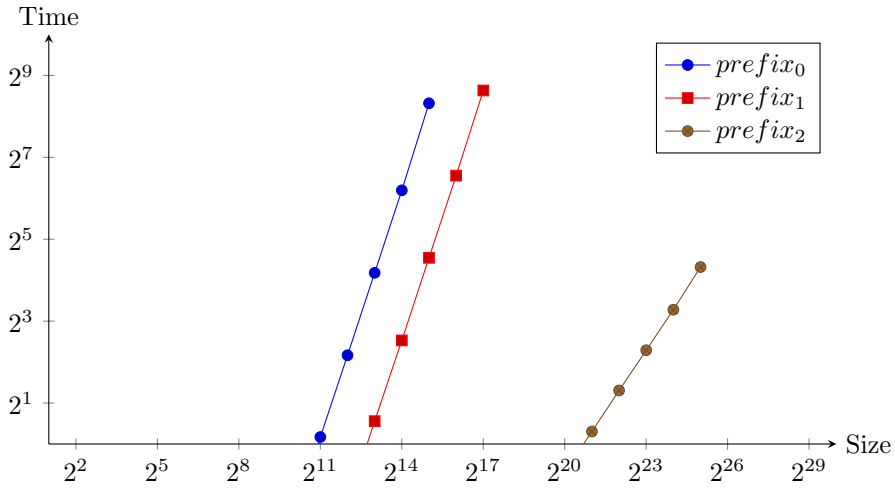
54. C-3.54

Statement A sequence S contains n integers taken from the interval $[0, 4n]$, with repetitions allowed. Describe an efficient algorithm for determining an integer value k that occurs the most often in S. What is the running time of your algorithm?

Solution Use a map, $T = O(n)$

55. P-3.55-3.56

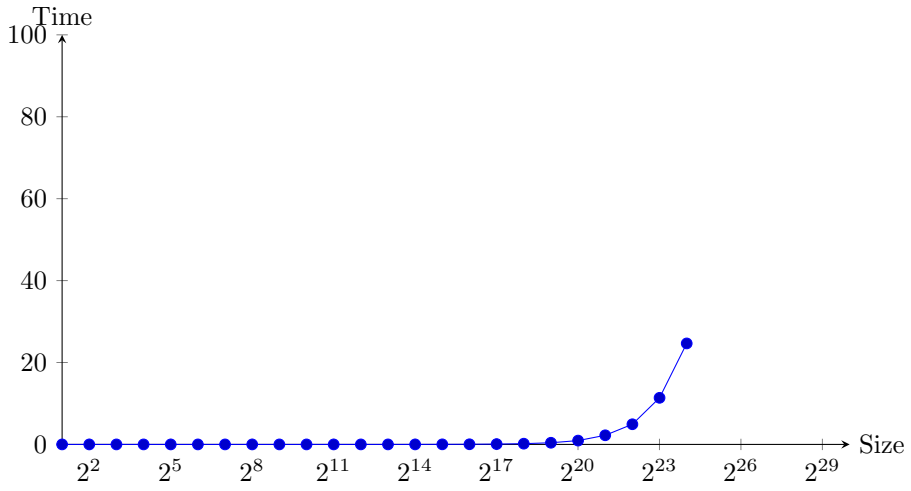
Statement Perform an experimental analysis of the three algorithms prefix average1, prefix average2, and prefix average3, from Section 3.3.3. Visualize their running times as a function of the input size with a log-log chart.



Solution

56. P-3.57

Statement Perform experimental analysis to test the hypothesis that Python sorted method runs in $O(nlogn)$-time on average.



57. C-3.54

Statement For each of the three algorithms, unique1, unique2, which solve the element uniqueness problem, perform an experimental analysis to determine the largest value of n such that the given algorithm runs in one minute or less.

Solution **unique1**: 60.0125 for $n = 2^14 + 2^10 + 2^9$ **unique2**: 59.96 for $n = 2^25 + 2^21$