7

Cost

Exercise 7.1. Suppose you are defining a procedure that needs to append two lists, one short list, *short* and one very long list, *long*, but the order of elements in the resulting list does not matter. Is it better to use (*list-append short long*) or (*list-append long short*)? (A good answer will involve both experimental results and an analytical explanation.)

Solution.

Exercise 7.2. For each of the g functions below, answer whether or not g is in the set $O(n)$. Your answer should include a proof. If g is in $O(n)$ you should identify values of c and n_0 that can be selected to make the necessary inequality hold. If g is not in $O(n)$ you should argue convincingly that no matter what values are chosen for c and n_0 there are values of $n \geq n_0$ such the inequality in the definition of O does not hold.

a. $g(n) = n + 5$

b. $g(n) = .01n$

c. $g(n) = 150n + \sqrt{n}$

d. $g(n) = n^{1.5}$

e. $g(n) = n!$

Exercise 7.3. [⋆] Given f is some function in $O(h)$, and g is some function not in $O(h)$, which of the following must always be true:

a. For all positive integers m, $f(m) \leq g(m)$.

b. For some positive integer m, $f(m) < g(m)$.

c. For some positive integer m_0, and all positive integers $m > m_0$,

$$f(m) < g(m).$$

Exercise 7.4. Repeat Exercise 7.2 using Ω instead of O.

Solution.

Exercise 7.5. For each part, identify a function g that satisfies the property.

a. g is in $O(n^2)$ but not in $\Omega(n^2)$.

Solution.

b. g is not in $O(n^2)$ but is in $\Omega(n^2)$.

Solution.

c. g is in both $O(n^2)$ and $\Omega(n^2)$.

Solution.

Exercise 7.6. Explain why the *list-map* procedure from Section 5.4.1 has running time that is linear in the size of its List input. Assume the procedure input has constant running time.

Solution.

Exercise 7.7. Consider the *list-sum* procedure (from Example 5.2):

```
(define (list-sum p) (if (null? p) 0 (+ (car p) (list-sum (cdr p)))))
```

What assumptions are needed about the elements in the list for the running time to be linear in the number if elements in the input list?

Solution.

Exercise 7.8. For the decimal six-digit odometer (shown in the picture on page 145), we measure the amount of work to add one as the total number of wheel digit turns required. For example, going from 000000 to 000001 requires one work unit, but going from 000099 to 000100 requires three work units.

a. What are the worst case inputs?

Solution.

b. What are the best case inputs?

Solution.

c. [⋆] On average, how many work units are required for each mile? Assume over the lifetime of the odometer, the car travels 1,000,000 miles.

Solution.

d. Lever voting machines were used by the majority of American voters in the 1960s, although they are not widely used today. Most level machines used a three-digit odometer to tally votes. Explain why candidates ended up with 99 votes on a machine far more often than 98 or 100 on these machines.

Solution.

Exercise 7.9. [⋆] The *list-get-element* argued by comparison to $+$, that the $-$ procedure has constant running time when one of the inputs is a constant. Develop a more convincing argument why this is true by analyzing the worst case and average case inputs for $-$.

Solution.

Exercise 7.10. [⋆] Our analysis of the work required to add one to a number argued that it could be done in constant time. Test experimentally if the DrRacket $+$ procedure actually satisfies this property. Note that one $+$ application is too quick to measure well using the *time* procedure, so you will need to design a procedure that applies $+$ many times without doing much other work.

Solution.

Exercise 7.11. [⋆] Analyze the running time of the elementary school long division algorithm.

Solution.

Exercise 7.12. [⋆] Define a Scheme procedure that multiplies two multi-digit numbers (without using the built-in $*$ procedure except to multiply single-digit numbers). Strive for your procedure to have running time in $\Theta(n)$ where n is the total number of digits in the input numbers.

Solution.

Exercise 7.13. [⋆⋆⋆⋆] Devise an asymptotically faster general multiplication algorithm than Fürer's, or prove that no faster algorithm exists.

Solution.

Exercise 7.14. Analyze the asymptotic running time of the *list-sum* procedure (from Example 5.2):

```
(define (list-sum p)
  (if (null? p)
      0
      (+ (car p) (list-sum (cdr p)))))
```

You may assume all of the elements in the list have values below some constant (but explain why this assumption is useful in your analysis).

Solution.

Exercise 7.15. Analyze the asymptotic running time of the *factorial* procedure (from Example 4.1):

```
(define (factorial n) (if (= n 0) 1 (* n (factorial (- n 1)))))
```

Be careful to describe the running time in terms of the *size* (not the magnitude) of the input.

Solution.

Exercise 7.16. Consider the *intsto* problem (from Example 5.8).

a. [⋆] Analyze the asymptotic running time of this *intsto* procedure:

```
(define (revintsto n)
  (if (= n 0)
      null
      (cons n (revintsto (- n 1)))))
(define (intsto n) (list-reverse (revintsto n)))
```

b. [⋆] Analyze the asymptotic running time of this *instto* procedure:

```
(define (intsto n)
  (if (= n 0) null (list-append (intsto (- n 1)) (list n))))
```

c. Which version is better?

d. [⋆⋆] Is there an asymptotically faster *intsto* procedure?

Solution.

Exercise 7.17. Analyze the running time of the *board-replace-peg* procedure (from Exploration 5.2):peg-board puzzle

```
(define (row-replace-peg pegs col val)
  (if (= col 1) (cons val (cdr pegs))
    (cons (car pegs) (row-replace-peg (cdr pegs) (- col 1) val))))
(define (board-replace-peg board row col val)
  (if (= row 1) (cons (row-replace-peg (car board) col val) (cdr board))
    (cons (car board) (board-replace-peg (cdr board) (- row 1) col val))))
```

Solution.

Exercise 7.18. Analyze the running time of the *deep-list-flatten* procedure from Section 5.5:

```
(define (deep-list-flatten p)
  (if (null? p) null
    (list-append (if (list? (car p))
                     (deep-list-flatten (car p))
                     (list (car p)))
                 (deep-list-flatten (cdr p)))))
```

Solution.

Exercise 7.19. [⋆] Find and correct at least one error in the *Orders of Growth* section of the Wikipedia page on *Analysis of Algorithms* (http://en.wikipedia.org/wiki/Analysis_of_algorithms). This is rated as [⋆] now (July 2011), since the current entry contains many fairly obvious errors. Hopefully it will soon become a [⋆⋆⋆] challenge, and perhaps, eventually will become impossible!

Solution.