

1. Explain what the scheme procedures **filter**, **map**, and **apply** do. Give a scheme expression that returns a value that behaves like  $-\infty$ .
2. Write a procedure **unique** such that given a sorted list **lst** of numbers, **(unique lst)** returns a sorted list whose numbers come from **lst** but with only one copy of any repeated number retained. For example,  
`(unique '(1 1 3 4 7 7 7))`  
returns the list `(1 3 4 7)`.
3. Write a merge procedure that works on three sorted lists instead of two.
4. Write a procedure **is-a-leaf?** such that **(is-a-leaf T)** returns `#t` if and only if **T** is a binary tree of exactly one node.
5. Write a procedure **count-leaves** such that **(count-leaves T)** returns the number of leaves in the binary tree **T**.
6. Write a procedure **count-internal-nodes** such that **(count-internal-nodes T)** returns the number of internal nodes in the binary tree **T**.
7. Write a procedure **count-descendants** such that **(count-descendants T)** returns the number of descendants that the root node of **T** has. What is this number equal to?
8. Write a procedure **mirror-image-tree** such that **(mirror-image-tree T)** returns the binary tree that is the mirror image of **T**.
9. Write a procedure **delete-leaves** such that **(delete-leaves T)** returns the binary tree that is **T** with all of its leaves removed.
10. Write a procedure **insert** such that **(insert bst value)** inserts a node with number **value** into the binary search tree **bst** and returns the resulting binary search tree.
11. Using the **insert** procedure from last problem, write a procedure **make-bst-with-keys** such that **(make-bst-with-keys key-list)** returns the binary search tree whose nodes hold the keys from **key-list** inserted in the reverse order specified by **key-list** into an originally empty tree.
12. Write a procedure **count-inbetween** such that **(count-inbetween lo hi T)** returns the number of nodes in the **binary search tree** **T** of numbers whose keys are greater than **lo** but smaller than **hi**.
13. Write a procedure **twice-tree** such that **(twice-tree T)** doubles the value of every node in the binary tree **T** of numbers and returns the resulting tree.

14. Write a procedure `tree-ref` such that `(tree-ref bst k)` returns the key value of the  $k$ th smallest node in the binary search tree `bst`. Assume we start counting from 0, i.e., `(tree-ref bst 0)` returns the key of the smallest node.

15. Write a procedure `median` such that `(median T)` returns the median value of keys in the binary search tree  $T$  of numbers.

16. Write a procedure `mode` such that `(mode T)` returns the mode value of keys in the binary search tree  $T$  of numbers.

17. Which of the procedures you wrote in Problems 1–16 generate recursive processes and which generate iterative ones? Explain.

18. Let us define an *elementary operation* in scheme to be either an arithmetic operation, or a comparison operation, or one of the three basic list operations `null?`, `cons`, `car`, or `cdr`. To illustrate, suppose `f` is the following scheme procedure

```
(define f
  (lambda (lst)
    (if (> (car lst) 1)
        (* (car lst) 2)
        (cdr lst))))
```

A call to `(f '(1 2 3))` will incur three elementary operations, while a call to `(f '(3 2 1))` will incur four. In fact, calling `(f lst)` on any nonempty list `lst` will cost four elementary operations *in the worst case*.

The following scheme procedure `select-sort` sorts its numeric list argument.

```
(define select-sort
  (lambda (num-list)
    (if (null? num-list)
        '()
        (cons (minimum num-list)
              (select-sort (delete-from (minimum num-list) num-list))))))
```

```
(define infinity (/ 1.0 0.0))
```

```
(define minimum
  (lambda (num-list)
    (if (null? num-list)
        infinity
        (min (car num-list)
             (minimum (cdr num-list))))))
```

```
(define delete-from
  (lambda (this-number num-list)
    (cond ((null? num-list) '())
          ((= (car num-list) this-number) (cdr num-list))))
```

```
(else (cons (car num-list)
            (delete-from this-number (cdr num-list))))))
```

(a) Give a formula  $S(n)$  for the number of elementary operations incurred in the worst case by a call to `(select-sort lst)` on a list `lst` of  $n$  numbers.

(b) Express  $S(n)$  in  $\Theta$  notation.

(c) Implement a more efficient version of `select-sort`. What is the worst-case running time of your procedure in  $\Theta$  notation?

**19.** Write a procedure `geometric-sequence-with-from-by` that takes as arguments a length, a starting value, and a multiplicative constant and returns the corresponding geometric sequence. For example, `(geometric-sequence-with-from-by 5 1 2)` would return the sequence  $\langle 1, 2, 4, 8, 16 \rangle$ . Note that by the term *sequence* we mean one with the procedures `empty-sequence?`, `head`, `tail`, `sequence-length`, and `sequence-ref` defined as on page 250 of the text.