1. Explain what the scheme procedures \texttt{filter}, \texttt{map}, and \texttt{apply} do. Give a scheme expression that returns a value that behaves like \(-\infty\).

2. Write a procedure \texttt{unique} such that given a sorted list \texttt{lst} of numbers, \((\texttt{unique lst})\) returns a sorted list whose numbers come from \texttt{lst} but with only one copy of any repeated number retained. For example,
   \[
   (\texttt{unique }'(1 1 3 4 7 7 7))
   \]
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 \texttt{is-a-leaf?} such that \((\texttt{is-a-leaf? T})\) returns \#t if and only if \(T\) is a binary tree of exactly one node.

5. Write a procedure \texttt{count-leaves} such that \((\texttt{count-leaves T})\) returns the number of leaves in the binary tree \(T\).

6. Write a procedure \texttt{count-internal-nodes} such that \((\texttt{count-internal-nodes T})\) returns the number of internal nodes in the binary tree \(T\).

7. Write a procedure \texttt{count-descendants} such that \((\texttt{count-descendants T})\) returns the number of descendants that the root node of \(T\) has. What is this number equal to?

8. Write a procedure \texttt{mirror-image-tree} such that \((\texttt{mirror-image-tree T})\) returns the binary tree that is the mirror image of \(T\).

9. Write a procedure \texttt{delete-leaves} such that \((\texttt{delete-leaves T})\) returns the binary tree that is \(T\) with all of its leaves removed.

10. Write a procedure \texttt{insert} such that \((\texttt{insert bst value})\) inserts a node with number \texttt{value} into the binary search tree \texttt{bst} and returns the resulting binary search tree.

11. Using the \texttt{insert} procedure from last problem, write a procedure \texttt{make-bst-with-keys} such that \((\texttt{make-bst-with-keys key-list})\) returns the binary search tree whose nodes hold the keys from \texttt{key-list} inserted in the reverse order specified by \texttt{key-list} into an originally empty tree.

12. Write a procedure \texttt{count-inbetween} such that \((\texttt{count-inbetween lo hi T})\) returns the number of nodes in the binary search tree \(T\) of numbers whose keys are greater than \texttt{lo} but smaller than \texttt{hi}.

13. Write a procedure \texttt{twice-tree} such that \((\texttt{twice-tree T})\) doubles the value of every node in the binary tree \(T\) of numbers and returns the resulting tree.
14. Write a procedure \texttt{tree-ref} such that \((\texttt{tree-ref \ bst \ k})\) returns the key value of the \(k\)th smallest node in the binary search tree \(\texttt{bst}\). Assume we start counting from 0, i.e., \((\texttt{tree-ref \ bst \ 0})\) returns the key of the smallest node.

15. Write a procedure \texttt{median} such that \((\texttt{median \ T})\) returns the median value of keys in the binary search tree \(T\) of numbers.

16. Write a procedure \texttt{mode} such that \((\texttt{mode \ T})\) 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 \textit{elementary operation} in scheme to be either an arithmetic operation, or a comparison operation, or one of the three basic list operations \texttt{null?}, \texttt{cons}, \texttt{car}, or \texttt{cdr}. To illustrate, suppose \(f\) is the following scheme procedure

\[
(\text{define } f \\
(\lambda (l s t) \\
  (\text{if} (> (\text{car} l s t) 1) \\
   (* (\text{car} l s t) 2) \\
   (\text{cdr} l s t)))))
\]

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 \((f \ \text{lst})\) on any nonempty list \(\text{lst}\) will cost four elementary operations \textit{in the worst case}.

The following scheme procedure \texttt{select-sort} sorts its numeric list argument.

\[
(\text{define } \text{select-sort} \\
(\lambda (\text{num-list}) \\
  (\text{if} (\text{null?} \ \text{num-list}) \\
   '() \\
   (\text{cons} (\text{minimum} \ \text{num-list}) \\
    (\text{select-sort} (\text{delete-from} (\text{minimum} \ \text{num-list} \ \text{num-list})))))))
\]

\[
(\text{define } \text{infinity} (/ 1.0 0.0))
\]

\[
(\text{define } \text{minimum} \\
(\lambda (\text{num-list}) \\
  (\text{if} (\text{null?} \ \text{num-list}) \text{infinity} \\
   (\text{min} (\text{car} \ \text{num-list}) \\
    (\text{minimum} (\text{cdr} \ \text{num-list})))))))
\]

\[
(\text{define } \text{delete-from} \\
(\lambda (\text{this-number} \ \text{num-list}) \\
  (\text{cond} ((\text{null?} \ \text{num-list}) '()) \\
   ((= (\text{car} \ \text{num-list}) \text{this-number}) (\text{cdr} \ \text{num-list}))))
\]
(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 ⟨1, 2, 4, 8, 16⟩. 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.