Consider the following data definitions with examples:

```scheme
; A Pair is
; -- (make-pair String Number)
(define-struct pair (word number))
; key and value would have been better names for the fields
(define ex1 (make-pair "guava" 4))

; A ListOfWords is one of
; -- empty
; -- (cons String ListOfWords)
(define ex2 empty)
(define ex3 (cons "banana" ex2))
(define ex4 (cons "guava" ex3))
(define ex5 (cons "banana" ex4))
(define ex6 (cons "apple"
             (cons "pear"
                   (cons "apple"
                        (cons "orange"
                             (cons "pear" empty)))))
(define ex7 (list "apple" "pear" "apple" "orange" "pear"))

; A Hashtable is one of:
; -- empty
; -- (cons Pair Hashtable)
(define ex8 empty)
(define ex9 (cons (make-pair "guava" 3) empty))
(define ex10 (cons (make-pair "avocado" 2) ex9))
(define ex11 (list (make-pair "avocado" 2) (make-pair "guava" 2)))
```
1. Design the function `insert-into` whose template is given below:

```scheme
; 1. insert-into
; data representation: see above
; signature Hashtable String -> Hashtable
; purpose statement: add String to Hashtable
; examples given ex9 "guava" expected: (list (make-pair "guava" 4))
; given ex8 "guava" expected (cons (make-pair "guava" 1) empty)
; given ex11 "apple" expected (list (make-pair "avocado" 2)
; ; (make-pair "guava" 2)
; ; (make-pair "apple" 1))
; given ex11 "guava" expected (list (make-pair "avocado" 2)
; ; (make-pair "guava" 3))
; (define (insert-into hash word)
; (cond ((empty? hash) ...)
; (else (... word ... (first hash) ...
; ; ... (insert-into (rest hash) word) ...)))))
```

(check-expect (insert-into ex11 "guava")
 (list (make-pair "avocado" 2) (make-pair "guava" 3)))
2. Design the function \textit{sort} whose template is given below

\begin{verbatim}
; 2. sort
; data representation: mentioned already
; ListOfWords -> Hashtable
; purpose statement: turn a list of words into a dictionary
; examples given (cons "apple" ex7) expected:
; (list (make-pair "apple" 3) (make-pair "orange" 1) (make-pair "pear" 2))
; (define (sort low)
;   (cond ((empty? low) ...)
;     (else (... (first low) ... (sort (rest low)) ...))))
\end{verbatim}

(check-expect (sort (cons "apple" ex7))
  (list (make-pair "pear" 2)
    (make-pair "orange" 1)
    (make-pair "apple" 3)))
3. Design a big-bang program that starts by showing the number of your choice. Then, every time you press the up arrow key, the number gets incremented by 1; every time you press the down arrow key the number gets decremented by one. Here’s the way my program works.

I start the program with the initial number 6. After pressing the up arrow key three times and the down arrow key two times. Of, course, the number changes accordingly after every key press.