```
;;Dylan Orpin
;;CS 441 Assignment 1 Code
;;ChatGPT was utilized to generate this code in all of its entirety
;;NOTE: I commented out the test code at the bottom of the program. If needed, feel free to uncomment it and test it.

#lang racket

;; Selection sort function to sort a small list and find the median
(define (selection-sort lst)
  (define (find-min-index lst)
    (let loop ([lst lst] [index 0] [min-index 0] [min-val (car lst)])
      (cond [(empty? lst) min-index]
            [(< (car lst) min-val) (loop (cdr lst) (+ index 1) index (car lst))]
            [else (loop (cdr lst) (+ index 1) min-index min-val)])))
  (define (sort-helper lst sorted)
    (if (empty? lst)
        sorted
        (let* ([min-index (find-min-index lst)]
               [min-val (list-ref lst min-index)]
               [remaining (append (take lst min-index) (drop lst (+ min-index 1)))])
          (sort-helper remaining (cons min-val sorted)))))
  (reverse (sort-helper lst '())))

;; Function to find the median value of a list
(define (find-median lst)
  (let* ([sorted (selection-sort lst)]
         [mid-index (quotient (length sorted) 2)])
    (list-ref sorted mid-index)))

;; Helper to check if there are at least 5 elements in a list
(define (has-five-elements? lst)
  (and (pair? lst)
       (pair? (cdr lst))
       (pair? (cddr lst))
       (pair? (cdddr lst))
       (pair? (cddddr lst))))

;; Function to split a list into sublists of up to 5 elements each
(define (split-into-fives lst)
  (define (take-five lst)
    (cond [(has-five-elements? lst) (take lst 5)]
          [else lst]))
  (if (empty? lst)
```

```
      '()
      (let ([chunk (take-five lst)])
        (cons chunk (split-into-fives (drop lst (length chunk)))))))

;; Function to recursively find the median of medians
(define (median-of-medians lst)
  (let ([sublists (split-into-fives lst)]
        [medians (map find-median (split-into-fives lst))])
    (if (not (has-five-elements? medians))
        (find-median medians)
        (median-of-medians medians))))

;; Quicksort function using median-of-medians for partitioning
(define (quicksort lst)
  (if (not (has-five-elements? lst))
      (selection-sort lst) ; Sort small lists directly
      (let* ([pivot (median-of-medians lst)]
             [lesser (filter (λ (x) (< x pivot)) lst)]
             [equal (filter (λ (x) (= x pivot)) lst)]
             [greater (filter (λ (x) (> x pivot)) lst)])
        (append (quicksort lesser) equal (quicksort greater)))))

;; Wrapper function to display start and finish messages
(define (quicksort-wrapper lst)
  (printf "Quicksort begins\n")   ; Display message once at the start
  (define sorted-list (quicksort lst))
  (printf "Quicksort finishes\n")  ; Display message once at the end
  (verify-sorted sorted-list))      ; Verify if the list is sorted correctly

;; Function to generate a list of random integers
(define (generate-random-integers count min-value max-value)
  (define (generate n)
    (if (zero? n)
        '()
        (cons (+ min-value (random (+ 1 (- max-value min-value))))
              (generate (- n 1)))))
  (generate count))

;; Function to verify if a list is sorted
(define (verify-sorted lst)
  (define (sorted-helper lst)
    (cond [(or (empty? lst) (empty? (cdr lst))) #t] ; Empty or single-element list is sorted
          [(<= (car lst) (cadr lst)) (sorted-helper (cdr lst))]
          [else #f]))
```

```
  (if (sorted-helper lst)
      (printf "The list was sorted correctly.\n")
      (printf "The list was NOT sorted correctly.\n")))

;; Function to test the program with various list sizes
(define (test-quicksort-sizes)
  (define sizes '(4 43 403 400003 10000003))
  (for-each
   (λ (size)
     (printf "Testing quicksort with list size: ~a\n" size)
     (define random-list (generate-random-integers size 1 100))
     (quicksort-wrapper random-list)
     (printf "\n")) ; Adds spacing between each test
   sizes))

(test-quicksort-sizes)

#|
;; Simple test cases for each function

(printf "\n\n\n")
(printf "Test code:\n\n")

;; Test selection-sort
(printf "Testing selection-sort...\n")
(printf "Expected: (1 2 3)\n")
(printf "Result: ~a\n\n" (selection-sort '(3 1 2)))  ; Expected: (1 2 3)

;; Test find-median
(printf "Testing find-median...\n")
(printf "Expected: 2\n")
(printf "Result: ~a\n\n" (find-median '(1 3 2)))    ; Expected: 2

;; Test has-five-elements?
(printf "Testing has-five-elements?...\n")
(printf "Expected: #t\n")
(printf "Result: ~a\n\n" (has-five-elements? '(1 2 3 4 5)))  ; Expected: #t

;; Test split-into-fives
(printf "Testing split-into-fives...\n")
(printf "Expected: '((1 2 3 4 5) (6 7))\n")
(printf "Result: ~a\n\n" (split-into-fives '(1 2 3 4 5 6 7))) ; Expected: ((1 2 3 4 5) (6 7))

;; Test median-of-medians
```

```
(printf "Testing median-of-medians...\n")
(printf "Expected: 3\n")
(printf "Result: ~a\n\n" (median-of-medians '(1 2 3 4 5))) ; Expected: 3

;; Test quicksort
(printf "Testing quicksort...\n")
(printf "Expected: (1 2 3 4 5)\n")
(printf "Result: ~a\n\n" (quicksort '(5 1 4 2 3))) ; Expected: (1 2 3 4 5)

;; Test quicksort-wrapper
(printf "Testing quicksort-wrapper...\n")
(printf "Expected: Quicksort begins\nQuicksort finishes\nThe list was sorted correctly.\n")
(printf "Result: ")
(quicksort-wrapper '(5 1 4 2 3)) ; Should print "Quicksort begins" and "Quicksort finishes"

(printf "\n")

;; Test generate-random-integers
(printf "Testing generate-random-integers...\n")
(printf "Expected: A list of 5 random numbers between 1 and 10\n")
(printf "Result: ~a\n\n" (generate-random-integers 5 1 10)) ; Expected: List of 5 random numbers
between 1 and 10

;; Test verify-sorted
(printf "Testing verify-sorted...\n")
(printf "Expected: The list was sorted correctly.\n")
(printf "Result: ")
(verify-sorted '(1 2 3 4 5)) ; Expected to print "The list was sorted correctly."

(printf "\n")

;; Test test-quicksort-sizes
(printf "Testing test-quicksort-sizes...\n")
(printf "Expected: Running quicksort on lists of sizes 4, 43, 403, 400003, 10000003 and verifying
sorting correctness.\n")
(printf "Result:\n")
(test-quicksort-sizes) ; Runs the size tests and prints results
|#
```