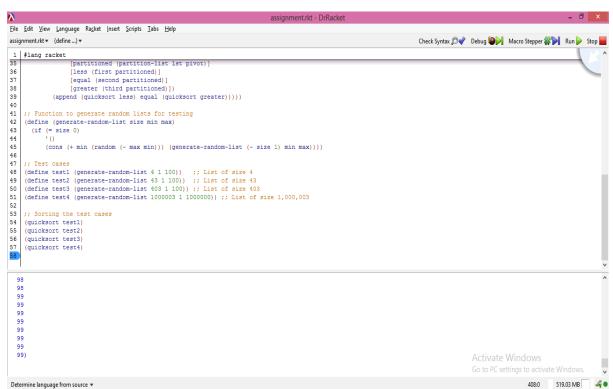
Code:

```
#lang racket
(define (find-median lst)
 (let ([sorted (sort lst <)])
  (list-ref sorted (quotient (length sorted) 2))))
(define (partition-into-sublists lst)
 (if (empty? lst)
    ()
    (let ([sublist (take lst (min 5 (length lst)))])
     (cons sublist (partition-into-sublists (drop lst (length sublist))))))
(define (median-of-medians lst)
 (let* ([sublists (partition-into-sublists lst)]
      [medians (map find-median sublists)])
  (if (<= (length medians) 5)
     (find-median medians)
     (median-of-medians medians))))
(define (partition-list lst pivot)
 (define less (filter (lambda (x) (< x pivot)) lst))
 (define equal (filter (lambda (x) (= x pivot)) lst))
 (define greater (filter (lambda (x) (> x pivot)) lst))
 (list less equal greater))
(define (quicksort lst)
 (if (\leq (length lst) 1)
    lst
    (let* ([pivot (median-of-medians lst)]
        [partitioned (partition-list lst pivot)]
        [less (first partitioned)]
        [equal (second partitioned)]
```

```
[greater (third partitioned)])
(append (quicksort less) equal (quicksort greater)))))
(define (generate-random-list size min max)
(if (= size 0)

'()
(cons (+ min (random (- max min))) (generate-random-list (- size 1) min max)))))
(define test1 (generate-random-list 4 1 100))
(define test2 (generate-random-list 43 1 100))
(define test3 (generate-random-list 403 1 100))
(define test4 (generate-random-list 1000003 1 1000000))
(quicksort test1)
(quicksort test2)
(quicksort test3)
(quicksort test4)
```

Output Screenshot:



Git-Hub link:

https://github.com/naveendharshanala/PLD