For this document, I'll begin by describing the process of working with the code, and how the code ultimately became what it is in the final form. Then for the "second part", I'll compare the final code to the initial code.

For this assignment I first began by asking ChatGPT to generate a program that conducted quicksort using median of medians partitioning, without library sorting routines. I did this to ultimately get the ball rolling and give the LLM a general idea of my goals to get started. The LLM understood what median of medians partitioning was, as well as quicksort, and didn't use library routines, and it included the utility functions at first, but they were incorrect. The first iteration had an error with partitioning items that weren't evenly distributed in groups of 5, so I simply copied the error I was shown and pasted it into ChatGPT to see what would happen. The LLM noticed that we needed to be able to partition items when they were in groups of less than 5, and fixed the error by allowing partitions of up to 5 for as long as it could until it reached a list with less than 5, in which it would then handle it separately.

At this point I realized that I shouldn't be physically printing out each list getting sorted as that would take too long, so I asked ChatGPT to only display when the sorting begins and ends. It had the right idea, as it displayed it each time a new "chunk" of 5 was being sorted, but I only wanted it to display it when it ultimately began and ended, so I had to correct it, because it couldn't understand what I meant unless I typed it out. It then fixed it by displaying only when it was first called and when it finished. At this point, I also realized I needed to implement the function of generating random lists that you provided for us, so I asked it to add this feature and to have this be quicksorted, to which it did provide this to me how I would have expected it, so i was pleased.

At this moment is where a big change to the code was coming. I noticed when testing with the new randomly generated integer feature, that it would slow down a lot (in the minutes), for lists the size of 400003, that you asked us to test our code with. So I asked the LLM to give me a rundown of what the code was doing and why it thought it might be slowing down. It described to me that it was breaking the code down into chunks of 5 by reading through all of it (with the length L), and that it was using insertion sort, which were both stated to be frowned upon in your rubric. Thus I gave ChatGPT more detailed instructions this time to (in short) implement quicksort using median of medians partitioning, and I broke it down into steps for it by stating: divide the list into sublists of 5 (until it is less than 5), find the median with selection sort (as you suggested), build up the list of medians, and then repeat those steps. It listened and changed it to selection sort, but I realized it was still heavily using the (length L) function. I then informed ChatGPT what you suggested and inputted to only verify that there were 5 items in the list to sort each time, rather than going through the list every time, thus getting rid of the heavy use of length L. ChatGPT listened and provided me with code that worked very well, and I tested it with items of 10 million plus and the lists were getting sorted in roughly 30 seconds, so it got me to a great point.

I then needed to add some final features to ensure everything was working correctly so I asked ChatGPT to add a feature that verifies the list was sorted correctly by going through it and ensuring each adjacent number was greater than or equal to the one behind it. The LLM provided me with this code and it worked great for me on the first try so I was pleased. To further enhance this code for the specific requirements, I asked ChatGPT to change the code to quicksort the code with sizes of 4, 43, 403, 400003, and 10000003 (ten million three) to ensure

that it was up to standard. ChatGPT then gave me this code that integrated my existing code to now test these list sizes on the first try, so I was once again pleased with its ability to do so.

An issue arose when I tried to incorporate some simple test code as mentioned in the rubric. ChatGPT gave me code that was way too complex for what it needed to be, it was ridden with bugs, and involved changing some of my previous code (changed definitions of functions) to the point where it didn't operate as intended. I told ChatGPT to undo what it did and to return back to the previous version of the code (I provided it back to the LLM to ensure it had the correct version I wanted), to which it did with no issues. I tried a different method by trying to break it down into steps, beginning with asking it to list out all of the functions in my program, to which it did, for even the tiniest functions. After it recognized my functions, I told it to only focus on the main ones (as some functions contained minute internal functions that were so simple I deemed them unnecessary to even test, plus since they are internal, testing the encompassing function inherently tests the internal functions). I listed out the functions I wanted to test, and had ChatGPT generate the code, but I forgot to tell it to make them simple, so it provided some intense test cases that were just too complex and lengthy for this scenario. So I told it to make them simpler, and to place them at the bottom of the program, in which it finally provided me with something similar to what I was looking for. This got me to a great point in which the code met all of the requirements, and the only part that was left was to try to organize it a little bit, which ended up being a big pain, as it was hard to try to explain exactly how you wanted it to look like (in terms of spacing, exact words/numbers used in the display). For example, I wanted it to display "Results:" when printing out the results of the test case, I wanted spacing between the different test cases, and spacing between the quicksorting of the different list sizes (to which I meant for it to change the spacing that was used in the function and not in the test case scenario). ChatGPT did it to a degree, but since this was pertaining to very minute details that were very simple in logic that didn't really have any effect on the code itself, I quickly changed them myself, and this slight organization was the only thing I did myself for this program. After this quick tidying up, that resulted in the completion of the program, and my experience/usage with ChatGPT.

For the next part of this discussion, I will break the final edition of the code up into sections, and describe how and why it changed, as well as how hard it was to get it to change throughout the process.

```
#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 '())))
```

For this first part, it was originally an insertion sort function that would sort the list by interesting the element in the correct place. This stayed relatively the same until I had to tell the LLM to use selection sort for the smaller lists, to which after that instruction to change to selection sort, it remained the same for the rest of the program.

```
;; 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)))
```

For this function to find the median of the list, this ultimately remained the same throughout the entire program as it was deemed efficient enough for its purpose, so this was unchanged ever since the first prompt given to ChatGPT.

```
;; 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))))
```

For this part of checking if there are 5 elements in a list, this was not seen in the beginning of the program as one of the main issues at the start was the lack of this feature. In the beginning of this program, it would forcefully take 5 elements from the list no matter what, and if there weren't 5 elements it would break. This change was made from my prompt after I suggested (based on your suggestion) to implement this logic, rather than using the length L function to go through the list each time, and after that prompt, it then remained the same.

```
;; 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)))))))
```

For this part, it ultimately began as a function that takes the first 5 elements from the list if it isn't empty, which of course didn't prove to be very helpful. Similar to the previous section of code, after I prompted ChatGPT to check for 5 elements rather than going through the whole list, as well as telling the LLM that it had an error when there were list sizes non-divisible by 5, this was created. And now this function uses the previous function (in the code above) to check if there are 5 elements and takes them if so, and if it has less, it sorts them differently/separately.

```
;; 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))))
```

For this part, originally it only started out as a function that split the list, and that was all that it did (I guess that was its way of "defining" that the list would eventually be here). Throughout the development process, and the prompts I gave, specifically the one where I broke down the steps of quicksort and median of medians partitioning, this function eventually turned into what it is now in which it calls other helper functions to recursively sort the median of medians until it gets one median.

```
;; 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)))))
```

This function stayed remotely similar throughout the program aside from adding the feature that selection sorts the smaller list if there is less than 5 elements, which was added after the prompt where I told the LLM there was an error when it was expecting 5 elements and couldn't take anything less than that, but after that prompt this change was made, then stayed the same.

```
;; 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
```

This function was not in the original code as this was added to the program after I prompted ChatGPT to add the feature of only displaying when the quicksort started, and when it ended. After this prompt it would display each time each “chunk” was getting sorted, so I had to tell the LLM to only display it when it ultimately began and ended, and then it worked and stayed the same.

```
;; 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))
```

This function was a very quick add-on that was not a part of the original code, and this was just for generating numbers for use on sorting the lists. This was added with a prompt when I asked the LLM to implement this code that you provided in the rubric and asked it to be used to quicksort the lists generated. After this prompt it was added and worked how I expected and was never changed.

```
;; 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")))
```

This function was also not a part of the original code as it was added near the end of the program process. I simply asked the LLM to add a feature that verified the code was sorted by

checking adjacent numbers and ensuring they were greater than or equal to the number next to them. After this prompt, the LLM added it and it was never changed.

```
;; 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)
```

This function was also an addition towards the end of the development process after I prompted ChatGPT to add it specifically. I instructed the LLM to add a feature that tested the quicksorting with list sizes of 4, 43, 403, 4000003, and 10000003 as you instructed in the rubric. This worked for me after this prompt was given and the code was added, and then it never changed (aside from minor organization/spacing that I noted in part 1.)

```
#|
;; 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
```

|#

These test cases were added towards the end of the process as well. I originally asked ChatGPT for test cases to be produced for all my functions. It then gave me overly complex code that wasn't necessary and it altered some of my existing code. I then had to tell the LLM to undo what it did and revert back to the previous version before that prompt. After that I instructed it to identify the functions in my code to try to break it down into steps for the LLM. After it identified all the functions I told it to focus on the main ones (as they contained the internal smaller functions, and those would inherently be tested with the encompassing function). After that it provided test code for those functions, but it was still way too complicated as I forgot to tell it to make them simpler. I then told it to make them simpler, and it finally provided what I was looking for. I did make tiny improvements after this by telling the LLM to show the expected value and the result of the test, as well as fixing some spacing issues to make it easier to read. I did eventually have to do some minor adjustments to spacing and organization myself as mentioned in part 1 after I tried going back and forth with ChatGPT. At the very end, I decided to comment the test cases out as well, so if need be please uncomment them.

Overall, I think ChatGPT can be useful for very simple stuff, or just to get you off and running, and can even help you do some monotonous stuff very quickly. However, there are some pitfalls, such as the lack of human understanding and common sense that humans would have, as well as the tendency to sometimes just ignore what you input and focus on other things. I also had some issues with the LLM changing some of my previous code and definitions when adding test code that was overly complicated (which was another issue), in addition to the struggles of going back and forth with it over detailed things like organization and spacing. I also think another pitfall of LLMs would be the inefficiency of the code that they produce sometimes, in which if you don't exactly spell it out for them, it tends to take a very simple route that ends up being very lengthy or the flat out wrong way to approach the scenario, but to some degree that is understandable because it is just a computer. If push comes to shove, I still believe that is still a beneficial tool to know how to use, and can be a great help if used correctly.