**Program 1 Discussion**

To start, here is a screenshot of the final version of the Racket program.

A screen shot of a computer program

AI-generated content may be incorrect.

You can see in my log file (LLMProjectLog.txt) that I first asked the LLM to generate a solution for the entire problem listed in the assignment. I copy/pasted the requirements directly into the LLM. Shockingly, it DID work the first time. However, it did not include test functions, a way to handle the 1m+ record file, a way to run the program on the “data-x” files.

**Here is the “first draft” generated by the LLM:**

#lang racket

; Function to count occurrences of each integer in a list and return value-count pairs

(define (count-occurrences lst)

(define hash-table (make-hash))

(for-each (lambda (x) (hash-update! hash-table x add1 0)) lst)

(sort (hash-map hash-table cons) < #:key car))

; Function to sort the value-count pairs by value (already sorted in count-occurrences)

(define (sort-value-count-pairs pairs)

(sort pairs < #:key car))

; Function to expand value-count pairs into a sorted list of integers

(define (expand-value-count-pairs pairs)

(apply append (map (lambda (pair) (make-list (cdr pair) (car pair))) pairs)))

; Function to verify if a list is sorted

(define (is-sorted? lst)

(or (null? lst)

(null? (cdr lst))

(and (<= (car lst) (cadr lst)) (is-sorted? (cdr lst)))))

; Function to read a list of integers from a file

(define (read-integers filename)

(with-input-from-file filename

(lambda ()

(let loop ((nums '()))

(define val (read))

(if (eof-object? val)

(reverse nums)

(loop (cons val nums)))))))

; Function to sort the integers from a file using counting sort

(define (counting-sort filename)

(let\* ((nums (read-integers filename))

(value-counts (count-occurrences nums))

(sorted-integers (expand-value-count-pairs value-counts)))

(displayln "Sorted List: ")

(displayln sorted-integers)

(displayln (string-append "List is sorted: " (if (is-sorted? sorted-integers) "Yes" "No")))

sorted-integers))

; Example usage:

; (counting-sort "numbers.txt")

The only *real* changes that were required were the changes made to the counting-sort function. The main reason was desired text output and dealing with the 1m+ file. Previously, the function would return the results for each function which, as you can imagine, was 3m+ numbers and caused DrRacket to need a uh.. Doctor.

Prompted the LLM to add changes to account for that. See screenshot below.

The “when” statement checks and makes sure the list-size is less than 1000. If it is less than 1000, it does not print the results of all the functions.

A screen shot of a computer program

AI-generated content may be incorrect.

This did fix the issue. Now, it will only return the “List is sorted: Yes/No”.

**Tests section:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

These tests are *almost* exactly what the LLM produced when I asked it to generate tests for each function.

I had to change Test 1 to expect a list instead of a hash.

Also had to include the Racket unit testing library.



After figuring out the 1m+ line issue. I went ahead and uncommented all the data-X files. The program runs on each of them in order and produces the correct output.

A screen shot of a computer

AI-generated content may be incorrect.

What I am most impressed about, is the other functions (screenshot below) were not changed *at all* the first generation was enough to fulfill the requirements. I am sure they are not 100% correct and could be better, but my LIMITED knowledge of Racket prevents me from confirming that.

\*List size was added to the counting-sort function\*

A screenshot of a computer program

AI-generated content may be incorrect.

**Results**

Here are the results of running the program on files **data-1-7** using DrRacket.

\*The bottom highlighted “List is sorted Yes” is from the 1m+ file. This is the only intended output\*

A screenshot of a computer

AI-generated content may be incorrect.