Sheyla Ramirez ***COMP_SCI-441-0001***
2/15/2025

For this project I used Claude, Deepseek, and Chatgpt, tho mostly Claud, and when I ran out of messages for the day I used the other LLMs.
1st Part:
For the first function, the original prompt was: “Show original data then sorted data” which gave the code:

```
; Create value-frequency pairs from a list of numbers
(define (consolidate-numbers numbers)
  (let loop ([nums (sort numbers <)]
             [current-num #f]
             [current-count 0]
             [result '()])
    (cond
      [(null? nums)
       (if current-num
           (cons (list current-num current-count) result)
           result)]
      [(not current-num)
       (loop (cdr nums) (car nums) 1 result)]
      [(= current-num (car nums))
       (loop (cdr nums) current-num (add1 current-count) result)]
      [else
       (loop (cdr nums)
             (car nums)
             1
             (cons (list current-num current-count) result))])))

; Expand value-frequency pairs into sorted list
(define (expand-pairs pairs)
  (apply append
         (map (lambda (pair)
                (make-list (second pair) (first pair)))
              pairs)))
```

The last update to this part of the code came from this prompt: “My assignment says this: Performs well even for large (10M+ item) lists. (Exact timing studies are unnecessary, but a correct implementation should be able to sort 10M+ numbers within a few seconds of wall-clock time.) I have a very large file, and it takes very long”

```
; Utility function 1: Convert list to value-count pairs efficiently
(define (make-value-count-pairs numbers)
  (define counts (make-hash))
  (for ([n numbers]) (hash-update! counts n add1 0))
  (sort (hash-map counts list) < #:key first))

; Utility function 2: Convert pairs to sorted list efficiently
(define (pairs->sorted-list pairs)
```

```
  (apply append (map (lambda (pair) (build-list (second pair) (λ (_) (first pair))))
pairs)))

; Utility function 3: Check if list is sorted
(define (is-sorted? lst)
  (for/and ([a lst] [b (cdr lst)]) (<= a b)))
```

This implemented a hash table, which bettered the complexity from O(n^2) to O(n).
2nd part:
I had a lot of issues reading the file, most of my LLM conversation was regarding this, and everything else was easy to deal with the LLM. I had both issues withing the code and with the directory of my code.
The original prompt was: “how to read file and ask user for file name on racket”

```
#lang racket

(define (read-file-contents filename)
  (define in (open-input-file filename))
  (define contents (port->string in))
  (close-input-port in)
  contents)

(define (ask-for-filename)
  (display "Enter the file name: ")
  (read-line))

(define (main)
  (define filename (ask-for-filename))
  (with-handlers ([exn:fail? (lambda (exn)
                               (displayln "Error: File not found or unable to read.")
                               (exit))])
    (define contents (read-file-contents filename))
    (displayln "File contents:")
    (displayln contents)))

(main)
```

This was not reading the files at all, by this point, I was not sure whether it was the directory or the code, so I tried for a while to debug. After a while, I deleted the racket packet and extension and downloaded it again and after trying for some more time it ended up working out.

the end result was:

```
; File reading function optimized for large files
(define (read-file-contents filename)
  (displayln (format "Attempting to open ~a" filename))
  (display "\t...")
  (with-handlers ([exn:fail? (lambda (exn)
                               (define msg (exn-message exn))
                                       (displayln (format "Debug: Exception     caught - ~a" msg))
                               (cond
                                 [(string-contains? msg "No such file or directory")
                                  (displayln "Error: File not found.")]
                                 [(string-contains? msg "Permission denied")
                                  (displayln "Error: Permission denied.")]
```

```
                              [else
                               (displayln "Error: Unable to read file.")])
                             #f)])
   (call-with-input-file filename
     (lambda (in)
       (define numbers '())
       (for ([line (in-lines in)])
         (set! numbers (append numbers (map string->number (string-split line)))))
       numbers))))
```

This result includes handling exceptions for different reasons why the file might not open, the main reason I wanted this was just to figure out whether the file was in the right directory and the code was not working out or whether the file was not found.

2nd part (part 2): asking for file name
This is pretty simple, this is one of the first requests I had for the LLM, along with reading the file, this is the end result:

```
(define (ask-for-filename)
  (newline)
  (display "Enter the file name (or 'q' to quit): ")
  (let ([filename (string-trim (read-line))])
    (if (string-ci=? filename "q")
        #f
        filename)))
```

3rd part:
After reading the assignment requirements I saw this part: Your program will implement an integer-sorting routine – a counting sort. You will need to do this in a couple of steps. Note that your code cannot call the library sort function.
This is the code I was given from this prompt:

```
; First define a function to find the maximum value in the list
(define (find-max lst)
  (if (empty? lst)
      0
      (foldl max (first lst) (rest lst))))

; Counting sort implementation
(define (counting-sort lst)
  (if (empty? lst)
      '()  ; Return empty list if input is empty
      (let* ([max-val (find-max lst)]
             [counts (make-vector (add1 max-val) 0)])  ; Create count array

        ; Step 1: Count occurrences of each number
        (for ([num lst])
          (vector-set! counts num (add1 (vector-ref counts num))))

        ; Step 2: Reconstruct the sorted list
        (let loop ([index 0]
                   [result '()])
          (if (> index max-val)
              (reverse result)  ; Return the final sorted list
              (let inner-loop ([count (vector-ref counts index)]
```

```
                            [current-result result])
                (if (zero? count)
                    (loop (add1 index) current-result)
                    (inner-loop (sub1 count)
                                (cons index current-result)))))))))
```

Then I noticed a more detailed requirement, so I gave this "Give this option to the user: Utility functions you will probably find useful:
◦Given a list of integers, return a list of value-count pairs
◦Sort a list of value-count pairs by value
◦Given a sorted list of value-count pairs, produce a sorted list of integers
◦Given a list of integers, confirm the list is sorted" to the LLm and I got this code;

```
; Utility function 1: Convert list of integers to value-count pairs using hash
(define (make-value-count-pairs numbers)
  (let ([freq-hash (make-hash)])  ; Create mutable hash
    ; Count frequencies
    (for ([num numbers])
      (hash-set! freq-hash num (add1 (hash-ref freq-hash num 0))))
    ; Convert hash to list of pairs
    (hash->list freq-hash)))

; Utility function 2: Sort value-count pairs by value
(define (sort-pairs pairs)
  (sort pairs < #:key first))

; Utility function 3: Convert sorted pairs to sorted list
(define (pairs->sorted-list pairs)
  (apply append
         (map (lambda (pair)
                (make-list (second pair) (first pair)))
              pairs)))

; Utility function 4: Check if list is sorted
(define (is-sorted? lst)
  (or (null? lst)
      (null? (cdr lst))
      (and (<= (car lst) (cadr lst))
           (is-sorted? (cdr lst)))))
```

The final code for this part has been already shown in 1st part; it was updated as this whenever I asked for the update in this prompt: "My assignment says this: Performs well even for large (10M+ item) lists. (Exact timing studies are not necessary, but a correct implementation should be able to sort 10M+ numbers within a few seconds of wall-clock time.) I have a very large file, and it takes very long"

```
; Utility function 1: Convert list to value-count pairs efficiently
(define (make-value-count-pairs numbers)
  (define counts (make-hash))
  (for ([n numbers]) (hash-update! counts n add1 0))
  (sort (hash-map counts list) < #:key first))
```

```
; Utility function 2: Convert pairs to sorted list efficiently
(define (pairs->sorted-list pairs)
  (apply append (map (lambda (pair) (build-list (second pair) (λ (_) (first pair))))
pairs)))

; Utility function 3: Check if list is sorted
(define (is-sorted? lst)
  (for/and ([a lst] [b (cdr lst)]) (<= a b)))
```

I also asked for user input options to select what they wanted to do with the file, the first prompt I asked was; “Give this options to the user: Utility functions you will probably find useful:◦ Given a list of integers, return a list of value-count pairs◦ Sort a list of value-count pairs by value◦ Given a sorted list of value-count pairs, produce a sorted list of integers◦ Given a list of integers, confirm the list is sorted. Also use the preferred hash structured”
Whenever I asked this, this code was also given;

```
(define (display-menu)
  (newline)
  (displayln "Available Operations:")
  (displayln "1. Convert list to value-count pairs")
  (displayln "2. Sort value-count pairs")
  (displayln "3. Convert pairs to sorted list")
  (displayln "4. Check if list is sorted")
  (displayln "5. Load numbers from file")
  (displayln "q. Quit")
  (display "Select an operation (1-5, q): "))

(define (process-numbers numbers op)
  (case op
    ['1 (let ([pairs (make-value-count-pairs numbers)])
          (displayln "\nOriginal numbers:")
          (displayln numbers)
          (displayln "\nValue-count pairs:")
          (displayln pairs))]
    ['2 (let* ([pairs (make-value-count-pairs numbers)]
               [sorted-pairs (sort-pairs pairs)])
          (displayln "\nUnsorted pairs:")
          (displayln pairs)
          (displayln "\nSorted pairs:")
          (displayln sorted-pairs))]
    ['3 (let* ([pairs (make-value-count-pairs numbers)]
               [sorted-pairs (sort-pairs pairs)]
               [sorted-list (pairs->sorted-list sorted-pairs)])
          (displayln "\nOriginal numbers:")
          (displayln numbers)
          (displayln "\nSorted numbers:")
          (displayln sorted-list))]
    ['4 (begin
          (displayln "\nChecking if list is sorted...")
          (displayln (if (is-sorted? numbers)
                         "The list is sorted!"
                         "The list is NOT sorted.")))]
    [else (displayln "Invalid operation")]))
```

However, for some reason, the LMM was trying to lose the number for the input, and I was getting errors every time I tried to run it. For a while, I was trying to sort this out, but after a couple of tries, I was able to get it to work with this prompt: “It is not accepting the input, and instead it is outputting "Invalid choice. Please enter A, B, C, D, or Q." After this prompt, it started accepting the input, however, it messed up how the file was being read, so I took that code and a previous one that was reading the file and combined them, which gave me the outcome I wanted.

```
; User input functions
(define (get-operation)
  (displayln "\nEnter 'Q' once to return to the file selection menu.")
  (displayln "\nWhat would you like to do?")
  (displayln "A) Convert list to value-count pairs")
  (displayln "B) Sort value-count pairs")
  (displayln "C) Given a sorted list of value-count pairs, produce a sorted list of
integers")
  (displayln "D) Given a list of integers, confirm the list is sorted")
  (displayln "Q) Quit")
  (display "Your choice (A/B/C/D/Q): ")
  (read-line))
//
(define (process-operation choice numbers)
  (let ([choice (string-upcase (string-trim choice))])
    (cond
      [(string=? choice "A")
       (let ([pairs (make-value-count-pairs numbers)])
         (displayln "\nOriginal numbers:")
         (displayln numbers)
         (displayln "\nValue-count pairs:")
         (displayln pairs))
       #t]
      [(string=? choice "B")
       (let ([sorted-pairs (make-value-count-pairs numbers)])
         (displayln "\nSorted pairs:")
         (displayln sorted-pairs))
       #t]
      [(string=? choice "C")
       (let* ([pairs (make-value-count-pairs numbers)]
              [sorted-list (pairs->sorted-list pairs)])
         (displayln "\nOriginal numbers:")
         (displayln numbers)
         (displayln "\nSorted numbers:")
         (displayln sorted-list))
       #t]
      [(string=? choice "D")
       (displayln "\nChecking if list is sorted...")
       (displayln (if (is-sorted? numbers)
                      "The list is sorted!"
                      "The list is NOT sorted."))
       #t]
      [(string=? choice "Q")
       #f]
```

```
      [else
       (displayln "\nInvalid choice. Please enter A, B, C, D, or Q.")
       #t])))
```

4th part:
This part is just the main, I had no trouble with this part, and it was always just updated whenever I input the full code into the LLM.

```
(define (main)
  (let filename-loop ()
    (define filename (ask-for-filename))
    (when filename
      (let ([numbers (read-file-contents filename)])
        (if numbers
            (let operation-loop ()
              (let ([continue? (process-operation (get-operation) numbers)])
                (when continue?
                  (operation-loop))))
            (displayln "Please try again.")))
      (filename-loop)))
  (displayln "\nGoodbye! <3"))

(main)
```