

## Log file with my AI prompts and machine-generated code

- LLM USED- DEEPSEEK

### Prompt-1:

Write a racket code where your program will implement an integer-sorting routine – a counting sort. Your program will be given a text file of integers (positive or negative, but all will fit within 32

bits). You should carry out two steps:

- Consolidate the list into a list of value-frequency pairs. Given the list (1 2 3 2 2 3 5 -1), you would produce ( (-1 1) (1 1) (2 3) (3 2) (5 1) ).
- Given the list of value-count pairs, produce a sorted list; that is, from the above you would produce (-1 1 2 2 2 3 3 5)

### Code generated:

```
#lang racket
```

```
;; Function to read integers from a file
```

```
(define (read-integers-from-file filename)
```

```
  (with-input-from-file filename
```

```
    (lambda ()
```

```
      (let loop ([integers '()])
```

```
        (let ([num (read)])
```

```
          (if (eof-object? num)
```

```
              (reverse integers) ; Return the list of integers in the correct order
```

```
              (loop (cons num integers)))))) ; Add the number to the list and continue
```

```
;; Function to create value-frequency pairs
```

```
(define (create-frequency-pairs integers)
```

```
  (let loop ([lst integers] [freq-map (hash)]) ; Start with an empty hash map
```

```
    (if (null? lst)
```

```
        (hash->list freq-map) ; Convert the hash map to a list of pairs
```

```
        (let ([num (car lst)])
```

```
          (loop (cdr lst) (hash-update freq-map num add1 0)))))) ; Update the frequency count
```

```
;; Function to sort value-frequency pairs by value
```

```
(define (sort-pairs pairs)
```

```
  (sort pairs (lambda (x y) (< (car x) (car y))))) ; Use built-in sort function
```

```
;; Function to rebuild a sorted list from value-frequency pairs
```

```
(define (rebuild-list pairs)
  (apply append (map (lambda (pair)
    (make-list (cdr pair) (car pair))) ; Create a list of repeated values
    pairs)))

;; Main function
(define (main)
  (printf "Enter the filename (in the same folder): ")
  (let ([filename (read-line)])
    (if (file-exists? filename)
      (let* ([integers (read-integers-from-file filename)]
        [pairs (create-frequency-pairs integers)]
        [sorted-pairs (sort-pairs pairs)]
        [sorted-list (rebuild-list sorted-pairs)])
        (printf "Value-Frequency Pairs: ~a\n" pairs)
        (printf "Sorted List: ~a\n" sorted-list))
      (printf "File not found. Please try again.\n"))))

;; Start the program
(main)
```

**Prompt 2:**

"Hey, I need a Racket program that asks the user for a filename, reads a text file of integers, and stores them as a list. Can you give me the basic structure for that?"

**Code Generated:**

```
#lang racket

;; Function to read integers from a file
(define (read-integers-from-file filename)
  (with-input-from-file filename
    (lambda ()
      (let loop ([integers '()])
        (let ([num (read)])
          (if (eof-object? num)
              (reverse integers) ; Return the list of integers in the correct order
              (loop (cons num integers))))))) ; Add the number to the list and continue

;; Function to get a filename from the user
(define (get-filename)
  (display "Enter the filename: ")
```

```
(read-line))
```

```
;; Read integers from the file and display them  
(define integers (read-integers-from-file (get-filename)))  
(displayln integers)
```

**Prompt 3:** "The program works fine, but if the file is missing or has stuff that's not numbers, it just crashes. Can you make it handle those situations better and ask the user to try again if something goes wrong?"

**Code Generated:**

```
#lang racket
```

```
;; Function to read integers from a file with error handling  
(define (safe-read-integers filename)  
  (with-handlers ([exn:fail? (lambda (exn) 'error)]) ; Catch any errors and return 'error  
    (with-input-from-file filename  
      (lambda ()  
        (map string->number (filter number? (string-split (port->string (current-input-port))))))))
```

```
;; Function to get a valid filename from the user  
(define (get-validated-filename)  
  (let loop ()  
    (display "Enter the filename: ") ; Ask the user for a filename  
    (define filename (read-line)) ; Read their input  
    (define data (safe-read-integers filename)) ; Try to read the file  
    (if (eq? data 'error)  
        (begin  
          (displayln "Invalid file or non-integer values. Try again.") ; Show an error message  
          (loop)) ; Ask again  
        data))) ; Return the valid data
```

```
;; Read integers from the file and display them  
(define integers (get-validated-filename))  
(displayln integers)
```

**Prompt 3:** "Now, I want the program to count how many times each integer appears in the list and store them as pairs, like (value frequency). For example, (1 2 3 2 2 3 5 -1) should become ((-1 1) (1 1) (2 3) (3 2) (5 1)). Can you help me with that?"

**Code Generated:**

```
#lang racket
```

```
;; Function to create value-frequency pairs
(define (create-frequency-pairs integers)
  (let loop ([lst integers] [freq-map (hash)]) ; Start with an empty hash map
    (if (null? lst)
        (hash->list freq-map) ; Convert the hash map to a list of pairs
        (let ([num (car lst)])
            (loop (cdr lst) (hash-update freq-map num add1 0)))))) ; Update the frequency count
```

```
;; Example usage
(define integers '(1 2 3 2 2 3 5 -1))
(define pairs (create-frequency-pairs integers))
(displayln pairs)
```

**Prompt 4:** "Cool, the pairs are working! But the output looks messy. Can you make it display the pairs nicely, one per line, with labels like Value: Frequency?"

**Code Generated:**

```
#lang racket
```

```
;; Function to display value-frequency pairs
(define (display-pairs pairs)
  (displayln "Value-Frequency Pairs:")
  (for-each (lambda (pair)
              (displayln (format "~a: ~a" (car pair) (cdr pair)))) pairs))
```

```
;; Example usage
(define pairs '((-1 1) (1 1) (2 3) (3 2) (5 1)))
(display-pairs pairs)
```

**Prompt 5:**

"So, Next, I need to sort the value-frequency pairs by the integer values. But here's the rule: I can't use the built-in sort function. Can you implement a manual sorting algorithm, like insertion sort or selection sort?"

**Code Generated:**

```
#lang racket
```

```
;; Manual sorting function (using insertion sort)
(define (manual-sort pairs)
  (define (insert x sorted)
```

```
(if (null? sorted)
  (list x) ; If the list is empty, just add x
  (let ([current (car sorted)])
    (if (< (car x) (car current)) ; Compare the first element of the pairs
      (cons x sorted) ; Insert x before the current element
      (cons current (insert x (cdr sorted)))))) ; Keep the current element and insert x into the
rest
(foldl insert '() pairs)) ; Insert each element into the sorted list
```

```
;; Example usage
(define pairs '((-1 1) (1 1) (2 3) (3 2) (5 1)))
(define sorted-pairs (manual-sort pairs))
(displayln sorted-pairs)
```

**Prompt 6:**

"Now that the pairs are sorted, I want to rebuild the original list of integers but in sorted order.

**Code Generated:**

```
#lang racket
```

```
;; Function to rebuild a sorted list from value-frequency pairs
(define (rebuild-list pairs)
  (apply append (map (lambda (pair)
    (make-list (cdr pair) (car pair))) ; Create a list of repeated values
    pairs)))
```

```
;; Example usage
(define sorted-pairs '((-1 1) (1 1) (2 3) (3 2) (5 1)))
(define sorted-list (rebuild-list sorted-pairs))
(displayln sorted-list)
```

**Prompt 7:**

"Almost done, now I want the program to check if the final list is actually sorted. If it is, print 'List is sorted!'; otherwise, print 'List is NOT sorted!'. Can you please add that?"

**Code Generated:**

```
#lang racket
```

```
;; Function to check if a list is sorted in ascending order
(define (is-sorted? lst)
  (or (empty? lst) ; If the list is empty, it's sorted
    (empty? (cdr lst)) ; If there's only one element, it's sorted
```

```
(and (<= (car lst) (cadr lst)) ; Check if the first two elements are in order
      (is-sorted? (cdr lst)))) ; Recursively check the rest of the list
```

;; Example usage

```
(define sorted-list '(-1 1 2 2 2 3 3 5))
(if (is-sorted? sorted-list)
    (displayln "List is sorted!")
    (displayln "List is NOT sorted!"))
```

### **Prompt 8:**

"Okay, the program works for only one file, but I want it to keep asking the user if they want to process another file or quit. If they say 'yes', it should restart; if they say 'no', it should exit. Can you add that loop to my code?"

### **Code Generated:**

```
#lang racket
```

;; Main loop for user interaction

```
(define (main-loop)
  (let loop ()
    (define integers (get-validated-filename)) ; Get a valid list of integers
    (define pairs (create-frequency-pairs integers)) ; Create value-frequency pairs
    (display-pairs pairs) ; Display the pairs
    (define sorted-pairs (manual-sort pairs)) ; Sort the pairs manually
    (define sorted-list (rebuild-list sorted-pairs)) ; Rebuild the sorted list
    (displayln sorted-list) ; Display the sorted list
    (if (is-sorted? sorted-list) ; Check if the list is sorted
        (displayln "List is sorted!")
        (displayln "List is NOT sorted!"))
    (display "Do you want to go again? (yes/no): ") ; Ask the user if they want to continue
    (if (string-ci=? (read-line) "yes")
        (loop) ; Restart the loop
        (displayln "Exiting...")))) ; Exit the program

;; Start the program
(main-loop)
```

### **Prompt 9:**

"Everything works, but the code is a bit messy. Can you clean it up, add some comments, and

make the output more user-friendly so that it increases readability. Also, make sure the loop works smoothly and the program repeats until the user says 'quit'."

**Code Generated:**

```
#lang racket

;; Function to read integers from a file with error handling
(define (safe-read-integers filename)
  (with-handlers ([exn:fail? (lambda (exn) 'error)]) ; Catch any errors and return 'error
    (with-input-from-file filename
      (lambda ()
        (map string->number (filter number? (string-split (port->string (current-input-port))))))))

;; Function to get a valid filename from the user
(define (get-validated-filename)
  (let loop ()
    (display "Enter the filename: ") ; Ask the user for a filename
    (define filename (read-line)) ; Read their input
    (define data (safe-read-integers filename)) ; Try to read the file
    (if (eq? data 'error)
        (begin
          (displayln "Invalid file or non-integer values. Try again.") ; Show an error message
          (loop)) ; Ask again
        data))) ; Return the valid data

;; Function to create value-frequency pairs
(define (create-frequency-pairs integers)
  (let loop ([lst integers] [freq-map (hash)]) ; Start with an empty hash map
    (if (null? lst)
        (hash->list freq-map) ; Convert the hash map to a list of pairs
        (let ([num (car lst)])
          (loop (cdr lst) (hash-update freq-map num add1 0)))))) ; Update the frequency count

;; Manual sorting function (using insertion sort)
(define (manual-sort pairs)
  (define (insert x sorted)
    (if (null? sorted)
        (list x) ; If the list is empty, just add x
        (let ([current (car sorted)])
          (if (< (car x) (car current)) ; Compare the first element of the pairs
```

```
(cons x sorted) ; Insert x before the current element
(cons current (insert x (cdr sorted)))))) ; Keep the current element and insert x into the
rest
(foldl insert '() pairs)) ; Insert each element into the sorted list

;; Function to rebuild a sorted list from value-frequency pairs
(define (rebuild-list pairs)
  (apply append (map (lambda (pair)
    (make-list (cdr pair) (car pair))) ; Create a list of repeated values
    pairs)))

;; Function to check if a list is sorted in ascending order
(define (is-sorted? lst)
  (or (empty? lst) ; If the list is empty, it's sorted
    (empty? (cdr lst)) ; If there's only one element, it's sorted
    (and (<= (car lst) (cadr lst)) ; Check if the first two elements are in order
      (is-sorted? (cdr lst))))) ; Recursively check the rest of the list

;; Main loop for user interaction
(define (main-loop)
  (let loop ()
    (define integers (get-validated-filename)) ; Get a valid list of integers
    (define pairs (create-frequency-pairs integers)) ; Create value-frequency pairs
    (display-pairs pairs) ; Display the pairs
    (define sorted-pairs (manual-sort pairs)) ; Sort the pairs manually
    (define sorted-list (rebuild-list sorted-pairs)) ; Rebuild the sorted list
    (displayln sorted-list) ; Display the sorted list
    (if (is-sorted? sorted-list) ; Check if the list is sorted
      (displayln "List is sorted!")
      (displayln "List is NOT sorted!"))
    (display "Do you want to go again? (yes/no): ") ; Ask the user if they want to continue
    (if (string-ci=? (read-line) "yes")
      (loop) ; Restart the loop
      (displayln "Exiting...")))) ; Exit the program

;; Start the program
(main-loop)
```



**Prompt 10:**

"great! so now, can you combine all the pieces into one final program? Make sure it includes everything- reading the file, handling errors, creating value-frequency pairs, sorting manually(no in-built sort fn) , rebuilding the list, checking if it's sorted, and looping until the user quits. Oh, and keep the comments casual and easy to understand!"

**Code Generated:**

```
#lang racket

;; Function to read integers from a file
(define (read-integers-from-file filename)
  (with-input-from-file filename
    (lambda ()
      (let loop ([integers '()])
        (let ([num (read)])
          (if (eof-object? num)
              (reverse integers) ; Return the list of integers in the correct order
              (loop (cons num integers)))))) ; Add the number to the list and continue

;; Function to create value-frequency pairs
(define (create-frequency-pairs integers)
  (let loop ([lst integers] [freq-map (hash)]) ; Start with an empty hash map
    (if (null? lst)
        (hash->list freq-map) ; Convert the hash map to a list of pairs
        (let ([num (car lst)])
          (loop (cdr lst) (hash-update freq-map num add1 0)))))) ; Update the frequency count

;; Manual sorting function (using insertion sort)
(define (manual-sort pairs)
  (define (insert x sorted)
    (if (null? sorted)
        (list x) ; If the list is empty, just add x
        (let ([current (car sorted)])
          (if (< (car x) (car current)) ; Compare the first element of the pairs
              (cons x sorted) ; Insert x before the current element
              (cons current (insert x (cdr sorted)))))) ; Keep the current element and insert x into the
    rest
    (foldl insert '() pairs)) ; Insert each element into the sorted list

;; Function to print the sorted value-frequency pairs in the desired format
```

```
(define (print-pairs pairs)
  (printf "~a\n" pairs)) ; Print the list of pairs directly

;; Function to ask the user if they want to go again
(define (ask-to-go-again)
  (printf "Do you want to process another file? (yes/no): ")
  (let ([response (read-line)])
    (string-ci=? response "yes"))))

;; Main function
(define (main)
  (let loop ()
    (printf "Enter the filename (in the same folder): ")
    (let ([filename (read-line)])
      (if (file-exists? filename)
          (let* ([integers (read-integers-from-file filename)]
                 [pairs (create-frequency-pairs integers)]
                 [sorted-pairs (manual-sort pairs)])
            (print-pairs sorted-pairs)) ; Print the pairs in the desired format
          (printf "File not found. Please try again.\n"))))
    (if (ask-to-go-again)
        (loop) ; Restart the loop
        (printf "Goodbye!\n")))) ; Exit the program

;; Start the program
(main)
```