Assignment 1 – Basic Racket Programming, LLM Use

Learning Goals:

- Use of ChatGPT, Claude, CoPilot, or other LLM as a debugging and code generating tool
  - Strengths (what it's good at)
  - Weaknesses (what it's not good at)
- Use of LLM as assistant to produce working code, and limits thereof
- Documentation of AI usage
- Familiarity with fundamental programming in Racket

Skills:

- Use of LLM to generate draft code
- Use of LLM to debug & fix code
- Testing machine-generated code

For this assignment, you'll use the large language model (LLM) of your choice, such as ChatGPT, CoPilot, Claude, etc., to implement a moderately-complex operation in Racket.

This assignment is about using the LLM to draft and improve the code; it's much more about the process than the outcome. This assignment isn't about code quality. **If all you turn in is code, you will receive a 0 for the assignment, even if the code is perfect in every way.** Likewise, it is not about your skills in Google-Fu. If you simply search online for an implementation, and then do a copy-paste, you will receive a 0 and be referred to the Dean's office for academic dishonesty. If you do such a copy-paste and give full references for where you found it, you won't be referred to the Dean's office, but you'll still get a 0.

The assignment:

Using the LLM (ChatGPT, Claude, CoPilot, etc) of your choice, write a function or set of functions in Racket that can carry out a quicksort using median-of-medians partitioning. Median-of-medians uses a recursive algorithm to select a pivot item close to the middle of the distribution. (More on median-of-medians below.)

Since the point of this is to build your own code, **do not call library sorting routines.** The sorting problem was chosen as a problem most students in the class are already familiar with.

You will submit 3 documents:

1. A transcript/log of your LLM interaction. It is not necessary to include interactions with the AI about general algorithms or problem decomposition, but do include any interactions in which you ask the LLM to generate or modify code. Include the LLM's answer, and show any further interactions in which you refine the code, attempt to get it to fix errors, etc.
2. A document discussing, for each block of code, how close to correct the initial code was, what changes you needed to make, etc. Basically, walk me through **the process of working with the LLM.** DO NOT JUST DESCRIBE WHAT THE CODE IS DOING. I can see what the code is doing, and any LLM can generate (badly-written) descriptions of what the code is doing (that might even be correct).

3. Correctly-implemented, working code. It does not need to be elegant or optimized, only correct. The discussion in document #2 should show a clear connection between the code produced by the LLM and the finished product. (Again, code correctness is only part of the goal here. If the final code bears no resemblance to the LLM's output, your grade will probably be marked down. An exception would be if there were a relatively simple task the LLM was having particular difficulty carrying out—it certainly happens—and after some attempts your comments end with something like "and I finally gave up on GPT and just wrote the code myself.")

You must turn in all 3 documents to be eligible for full credit.

Other thoughts & information:

Algorithmic note: There is a straightforward proof that if we have N items in our original distribution, at least N * 3/10 will be less than or equal to the item we select as pivot, and the same number must be equal or greater. In practice, the division is usually very close to equal. This guarantees that as our list becomes larger, the minimum size of our partition grows accordingly, guaranteeing N log N overall behavior.

Median of medians: With this partitioning, we begin with the list of values we wish to sort. For this assignment, you can assume the values are numeric. Divide the list into sublists of 5 items each (the last sublist may have fewer than 5). Find the median of each sublist—since the sublists are known to contain no more than 5 items, we can just selection-sort them and take the middle value. Build up a list of the median values of the sublists. Then, recursively, find the median of those medians by doing a further division into sublists of 5, and so on, until there is only a single list of size 5 or fewer remaining. Take the middle value of that sublist as the (approximate) median of the original distribution, and thus our pivot.

(Yes, there are other ways to estimate the median quickly, particularly when you do not need the exact median, but only an estimate. Don't use them; use this method.)

Programming Notes: Be careful not to call the (length L) function if it isn't needed. Specifically, avoid `(if (> (length L) 5)` in deciding whether to continue. Here's why:

- We count the items in the list. `(length L)` returns 10,000,000; there are 10,000,000 items in the list.
- We use `(take L 5)` to make the first sublist, leaving us with `(drop L 5)` items remaining. We call `(length L)` on the remaining list, and get a value of 9,999,995. But we didn't need the full count; we only needed to know we still had more than 5.
- We slice off the next sublist, and `(length L)` returns 9,999,990. But we didn't need to know the exact count, only that we still had more than 5. In short, we're spending much more time counting items than we need to.
- You may want to avoid passing the length at all; or pass the length as an additional parameter; or write a function that verifies a list has at least 5 items (without calling the `(length L)` function) and returns; or something else.
- To see if this is an issue in your code, here's a function to generate a list of random integers:

```
(define (generate-random-integers count min-value max-value)
   (define (generate n)
     (if (zero? n)
        '()
         (cons (random min-value max-value)
```

```
                (generate (- n 1)))))
  (generate count))
```

- Test your code with lists of size 4, 43, 403, and 400,003. (Ending with 3 items guarantees the "fewer than 5 items" case exists, for testing.) If longer lists make your code painfully slow, you will need to revise your code.
- Utility functions you will probably need, or bits of code you'll need to write. And yes, you can have the LLM do these for you, and it'll (probably) be correct; just include it as part of your submission.
  - Find the minimum (or maximum) value of a list
  - Given a value and a list, filter all items {less than / equal to / greater than} that value. (Hint: Make the comparison a parameter!)
  - Given the above, implement selection sort. (You'll use this on the short sublists.)
  - Given a list, confirm that it's in nondecreasing order (that is, that it's sorted).

Academic notes: Your prompts and interactions with the LLM should be your own original work. Obviously, responses will be different for everyone, even for the same prompts. (Identical LLM responses will be reviewed for academic-honesty issues.) *Exceptions*: If you start off by copying part of the assignment text as part of your prompt, that's fine, provided you then develop the results further with your own prompts. Simple functions (e.g., find the maximum element of a numeric list) are likely to be near-identical anyway.

You may consult other sources, but should do so *only* after the LLM has demonstrated its uselessness at fixing that error, writing that function, etc. This should include more than 1 attempt to fix the problem before falling back on something else. Again, this assignment is mostly about your interaction with the LLM.

Rubric:
To be eligible for an A:

- All 3 documents specified are submitted.
- Transcript is complete. All prompts are student's own original work (except for text copied directly from the assignment document). All aspects of the problem (basic sort, finding median of medians, utility functions to sort or find medians of sublists, etc) are addressed.
- Discussion document covers each significant block of code. Describe why you asked for or manually made the modifications you did, particularly for things that aren't bug fixes. DON'T JUST EXPLAIN WHAT THE CODE IS DOING. I can see what the code is doing. Comment on the quality of the code the LLM produced, how well it was able to modify the code in response to your prompts, and how difficult it was to integrate the various components.
- Code is correct:
  - Sorts list of numeric values correctly without calling library sort routines, using specified algorithms.
  - Test code for functions is included; code passes all relevant tests.
  - Pitfalls discussed in assignment document are addressed and bypassed.
  - 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.)
- Discussion document provides a clear linkage between the code produced by the LLM and the final code. Transformations from original LLM code through revisions to final version can be clearly traced for all code. Problems or shortcomings of LLM code are acknowledged & dealt with.

- In cases where other sources were used, the transcript and discussion show that the LLM's initial response was inadequate and 3+ attempts at a fix failed (new errors introduced, error not addressed, cycling between 2 or more errors, etc)

To be eligible for a B:

- All 3 documents specified are submitted.
- Transcript is mostly complete. All prompts are student's own original work (except for text copied directly from the assignment document). Most aspects of the problem (basic sort, finding median of medians, utility functions to sort or find medians of sublists, etc) are addressed.
- Discussion document covers most significant blocks of code; some utility functions may be omitted from discussion. Describe why you asked for or manually made the modifications you did, particularly for things that aren't bug fixes. DON'T JUST EXPLAIN WHAT THE CODE IS DOING. I can see what the code is doing. Comments on LLM code quality may be somewhat vague or superficial, but cover most significant issues, including initial quality and difficulty in making modifications or adjustments to code. It is possible to trace the progress from LLM code to final code in most cases.
- Code is correct:
  - Sorts list of numeric values correctly without calling library sort routines, using specified algorithms.
  - Test code for functions is included; code passes all relevant tests.
  - Pitfalls discussed in assignment document are addressed and mostly bypassed.
  - Performs fairly well for large (10M+ item) lists. (Sorts 10M numbers within a couple of minutes of wall-clock time.)
- Discussion document provides a mostly-clear linkage between the code produced by the LLM and the final code. Transformations from original LLM code through revisions to final version can be traced for most code. Problems or shortcomings of LLM code are acknowledged & dealt with.
- In cases where other sources were used, the transcript and discussion show that the LLM's initial response was inadequate and some attempt was made to fix the code via the LLM before turning to other sources.

To be eligible for a C:

- All 3 documents specified are submitted.
- Transcript is mostly complete but has some significant gaps. All prompts are student's own original work (except for text copied directly from the assignment document). Most aspects of the problem (basic sort, finding median of medians, utility functions to sort or find medians of sublists, etc) are addressed but there are significant gaps.
- Discussion document covers the few most significant blocks of code; most utility functions may be omitted from discussion. Describe why you asked for or manually made the modifications you did, particularly for things that aren't bug fixes. Some of the discussion (no more than about a third) is just describing what the code does. Comments on LLM code quality may be somewhat vague or superficial, but cover the most significant issues, including initial quality and difficulty in making modifications or adjustments to code. It is possible to trace the progress from LLM code to final code in most cases.
- Code is correct:
  - Sorts list of numeric values correctly without calling library sort routines.
  - Test code for functions is included; code passes most relevant tests.

    - Pitfalls discussed in assignment document are no more than partly addressed; significant performance issues may be present as a result.
    - Sorts small to moderate sized lists (1M items or fewer) correctly but may bog down on larger lists.
- Discussion document provides some reasonably-clear linkage between the code produced by the LLM and the final code. Transformations from original LLM code through revisions to final version can be traced for the most significant or complex code. Some problems or shortcomings of LLM code are acknowledged & dealt with.
- In cases where other sources were used, the transcript and discussion show that the LLM's initial response was inadequate.

To be eligible for a D:

- The discussion file is missing, but the transcript and code files are present. Transcript shows that the code submitted was generated by the LLM; modifications are no more than minor if discussion file is missing. If discussion file is present, some modifications are unexplained, more than a third of the 'discussion' is just description of what the code is doing, or description/discussion of the code is machine-generated.
- Code runs correctly (uses Quicksort algorithm) but does not do median-of-medians partitioning.
- Code appears to be LLM-produced version in the transcript, perhaps with some modifications, but the connections appear evident.
- Transcript file shows initial code generation for several aspects of the larger problem.
- Any outside sources used for code are cited via comments in the code file.