Sort Wars: Rise of AI (Project 1)

CS 351: Analysis of Algorithms

Lucas P. Cordova, Ph.D.

This project involves building a performance analysis framework to empirically evaluate AI-generated sorting algorithms.

Table of contents

0.1 Project Overview

You will build a comprehensive performance analysis framework to empirically evaluate AI-generated sorting algorithms.

Key components:

- Prompt generative AI to create sorting algorithms
- Instrument algorithms for performance measurement

- Conduct rigorous empirical analysis
- Validate theoretical complexity predictions

0.2 Learning Objectives

By completing this assignment, you will:

- Perform empirical algorithm analysis through hands-on measurement
- Develop prompt engineering skills for AI-assisted code generation
- Apply statistical analysis to algorithm performance data
- Compare theoretical vs empirical complexity results
- Practice professional software development

0.3 GitHub Classroom Workflow

Setup process:

1. Accept the assignment using provided GitHub Classroom link in the assignment.
2. Clone your repository locally for development
3. Submit via tagged release when complete

Note: If you are unable to accept the assignment (open issue), you can use your own GitHub (and add me as a collaborator (GitHub handle: LucasCordova)).

Important: Final submission must be a GitHub release tagged (i.e., v1.0).

0.4 Phase 1: Build Instrumentation Framework

First, identify what performance metrics matter for sorting algorithms.

Required metrics to track:

- Number of comparisons between elements
- Number of swaps/element movements
- Array access operations
- Memory allocations
- Wall-clock runtime
- Peak memory usage (optional for higher grades)

0.5 Phase 1: Framework Structure

Example framework (suggestion only):

```
class SortingProfiler:
    def __init__(self) -> None
    def reset_counters(self) -> None
    def start_timing(self) -> None
    def end_timing(self) -> None
    def compare(self, a: Any, b: Any) -> bool
    def swap(self, arr: List, index_i: int, index_j: int) -> None
    def get_metrics(self) -> PerformanceMetrics
```

Note: Design your own interface that meets functional requirements

0.6 Phase 2: AI Algorithm Generation

Craft effective prompts that generate working, instrumentable sorting algorithms.

Required algorithms:

- **Standard QuickSort**: Basic implementation with simple partitioning
- **Optimized QuickSort**: Enhanced with multiple optimization techniques

0.7 Phase 2: Prompt Engineering Requirements

Document your process:

- Document your prompt iteration process
- Show how you refined prompts to get better results
- Include the final prompts that generated your algorithms
- Either get AI to include instrumentation hooks OR manually add them

0.8 Phase 3: Empirical Analysis

Run comprehensive tests, document the metrics you collected, and analyze performance data.

Analysis requirements:

- Test multiple input sizes (10, 50, 100, 500, 1000, 2000, 5000, 10000)
- Test different data distributions (random, sorted, reverse-sorted, nearly-sorted, duplicates)
- Fit complexity curves to empirical data
- Estimate best-fit complexity (O(n), O(n log n), $O(n^2)$, etc.)
- Extrapolate performance predictions for larger input sizes
- Compare empirical results to theoretical expectations

0.9 Grading: Specification-Based

This assignment uses specification-based grading

Your grade is determined by which functionality threshold you achieve, not by partial credit.

You must meet ALL requirements for a grade level to earn that grade.

0.10 D Level (60-69%): Basic Implementation

Required functionality:

- Functional instrumentation framework (comparisons and runtime)
- AI-generated standard QuickSort with basic instrumentation
- Test suite covering at least 3 input sizes and 2 distributions (how the data is arranged)
- Basic performance data collection and reporting
- GitHub repository with tagged release

0.11 C Level (70-79%): Enhanced Analysis

All D-level requirements PLUS:

- Instrumentation measures comparisons, swaps, array accesses, and memory
- Performance analysis across 5+ input sizes and 4+ distributions
- Basic curve fitting to estimate time complexity
- Comparison between different input distributions (best/average/worst case)
- Written analysis comparing empirical vs theoretical complexity

0.12 B Level (80-89%): Algorithm Comparison

All C-level requirements PLUS:

- Implementation and analysis of both standard AND optimized QuickSort
- Side-by-side performance comparison with statistical analysis
- Extrapolation analysis predicting performance for larger input sizes
- Comprehensive documentation of AI prompt engineering process
- Professional-quality performance visualization (graphs/charts)

0.13 A Level (90-100%): Advanced Optimization

All B-level requirements PLUS:

- Analysis of the **most optimized version** of ANY sorting algorithm (your choice!)
- Implementation includes advanced optimizations beyond basic AI generation
- Rigorous statistical analysis
  - Empirical validation of theoretical complexity predictions
  - Realistic extrapolation with awareness of limitations
  - Professional presentation of results with clear visualizations
  - Research-quality documentation

- Exploration of algorithm optimization beyond what AI generated

0.14 A-Level Challenge: Go Beyond QuickSort

Consider implementing and analyzing:

- **Introsort** (hybrid QuickSort/HeapSort/InsertionSort)
- **TimSort** (Python's built-in adaptive merge sort)
- **Dual-Pivot QuickSort** (Java's default sort)
- **Radix Sort variants** for specific data types
- Research and implement another cutting-edge sorting algorithm

0.15 Extra Credit: Software Design (+10 points max)

Demonstrate advanced coding practices for your language:

- **Interface Design** (+3 points): Clean, professional interfaces
- **Inheritance Hierarchy** (+3 points): Proper class hierarchy
- **Operator Overloading** (+2 points): Instrumented operators
- **Functional Programming** (+2 points): Higher-order functions

Research best practices for your chosen language and paradigm!

0.16 What I'm Looking For: Technical Excellence

Key technical aspects:

- Accurate instrumentation that doesn't interfere with algorithm correctness
- Comprehensive testing across multiple scenarios and edge cases
- Sound statistical analysis with proper curve fitting and extrapolation
- Professional code quality with type hints/comments, documentation, and testing

0.17 What I'm Looking For: AI Integration Mastery

AI-related expectations:

- Effective prompt engineering with documented iteration and refinement
- Critical evaluation of AI-generated code quality and correctness
- Manual optimization beyond what the AI provided (especially for A-level)
- Clear documentation of what came from AI vs. your own contributions

0.18 What I'm Looking For: Analytical Rigor

Analysis expectations:

- Empirical validation of theoretical complexity predictions
- Statistical significance in your performance comparisons
- Realistic extrapolation with awareness of limitations
- Professional presentation of results with clear visualizations

0.19 Repository Structure

Organize your project professionally. Here's a suggestion for a Python structure:

```
repo-[username]/
  README.md                  # Project overview
  requirements.txt           # Dependencies
  src/                       # Source code
  tests/                     # Unit tests
  docs/                      # Documentation
  results/                   # Performance data
  notebooks/                 # Analysis notebooks
```

0.20 Required Documentation

Essential documentation:

- **README.md**: Clear instructions for running your code
- **Prompt Engineering Log**: Document your AI interaction process
- **Performance Report**: Complete empirical analysis with conclusions
- **Algorithm Comparison**: Side-by-side analysis of your implementations

0.21 Milestone 1: Foundation

Set up foundation:

- Set up your GitHub repository and development environment
- Build a basic instrumentation framework
- Test your profiler with a simple sorting algorithm (like bubble sort)
- Verify that your measurements are accurate

0.22 Milestone 2: AI Generation

Generate algorithms:

- Experiment with different AI models and prompting strategies
- Generate your standard QuickSort implementation
- Integrate instrumentation (either from AI or manually)
- Test for correctness across various inputs

0.23 Milestone 3: Analysis

Analyze performance:

- Run comprehensive performance tests
- Implement complexity analysis and curve fitting
- Generate performance visualizations
- Write your analysis report

0.24 Milestone 4: Optimization (B/A Level)

Optimize and finalize:

- Generate or implement optimized algorithms
- Conduct comparative analysis
- Polish documentation and code quality
- Prepare final submission

0.25 Recommended AI Models

AI platforms to consider:

- **ChatGPT-4o or 5**: Generally good at generating correct algorithms
- **Claude**: Often produces well-documented code
- **GitHub Copilot**: Good for code completion and instrumentation
- **Gemini**: Alternative perspective on algorithm generation

Let me know if you have access issues!

0.26 Programming Language Options

Choose any language you prefer:

- **Python**: Great libraries for analysis and visualization
- **C#**: Excellent performance measurement tools
- **C++**: For low-level optimizations
- **Rust**: Safe and fast systems programming with optimized concurrency
- **JavaScript/TypeScript**: For web-based visualizations

Consider learning a new language to expand your toolkit!

0.27 Final Submission Process

Steps to complete:

1. Complete all required functionality for your target grade level
2. Run comprehensive tests to ensure correctness
3. Generate final performance report with all analysis
4. Create GitHub release tagged as v1.0
5. Submit your GitHub release URL through Canvas

0.28 Key Takeaways

Remember:

- **Specification-based grading**: Must meet ALL requirements for a grade level
- Focus on completely achieving your target level first
- Document your AI interaction process thoroughly
- Balance AI assistance with your own analytical contributions
- Professional code quality and documentation matter

Good luck combining AI tools with rigorous algorithmic analysis!