Algorithm Selection: A Predictive Model for Optimal Sorting

Artem Kiselev

April 5, 2025

1 Introduction

Sorting is a fundamental task that appears across various applications in computer science, from database management to data analytics and real-time processing systems. Due to its critical importance, hundreds of sorting algorithms have been developed, each with unique performance characteristics optimized for particular scenarios. Some algorithms excel at sorting nearly-sorted data, others at handling large datasets, and others at optimizing memory usage. As a consequence, no single algorithm universally outperforms all others for all problem instances. This naturally leads to the following research question:

> *Can we design a model that dynamically and intelligently selects the optimal sorting algorithm based on the characteristics of a given dataset?*

Successfully addressing this question and constructing an effective predictive model would represent a meaningful advancement. Such a model, capable of analyzing a dataset's characteristics and predicting the optimal sorting strategy based on that analysis, could deliver substantial performance improvements over current static approaches. In practical terms, this could significantly enhance efficiency in real-world scenarios, including large-scale database operations, data-intensive computations, and latency-sensitive applications, providing tangible benefits over existing sorting implementations.

2 Literature Review

A critical component in developing a dynamic selection system is determining the relevant features of the input data. Mannila's work on *presortedness* [3] introduces a key concept that quantifies how ordered a dataset already is, which in turn helps to predict the optimal sorting strategy. Building on this, Petersson and Moffat [5] propose a refined measure of presortedness that effectively adapts to other known measures in the literature. Further analysis of adaptive sorting techniques is provided by Estvill-Castro and Wood [1], who explore a range of presortedness metrics and their implications for sorting efficiency.

In 2023, Google DeepMind et al. [2] published a work on using Deep Reinforcement Learning to improve existing sorting algorithms, with parts of their findings being incorporated into LLVM. Although this paper may not directly inform the project, it is motivating to see how machine learning methods can enhance sorting algorithm design and performance.

Complementing these theoretical approaches, practical benchmarks are crucial for understanding real-world performance. The benchmarking data compiled on the GitHub page by Morwenn [4] provides detailed empirical measurements of various sorting algorithms under different conditions. This resource will be greatly beneficial towards determining which algorithms should be included in the portfolio.

Finally, Rice's seminal framework for algorithm selection [6] formalizes the process of mapping input characteristics to algorithm performance. This framework will be instrumental when attempting to theoretically formulate and verify the problem, along with the findings and methods of the project.

3 Proposed Approach

The proposed system will incorporate the following components:

1. **Data Profiling Module:** Analyzes incoming data to extract features such as size, existing order (presortedness), and data distribution. Metrics like the number of inversions, runs, las, and others outlined in Mannila's [3] paper will be computed to quantify presortedness.

2. **Algorithm Repository:** Maintains a collection of sorting algorithms with annotated performance characteristics, including time complexity, space complexity, and adaptability to specific data patterns.

3. **Decision Engine:** Employs machine learning models trained on historical data to predict the most suitable sorting algorithm based on the profiled data characteristics. Features for the model may include data size, presortedness measures, and distribution patterns.

4. **Execution Monitor:** Oversees the sorting process, collecting performance metrics to refine the decision engine's predictions over time through reinforcement learning techniques.

My practical approach towards implementing the project will be as follows:

1. **Algorithm Selection:** Select sorting algorithms into the algorithm portfolio that have an advantage in a meaningful amount of scenarios. Achieve this by looking at papers mentioned in the literature review.

2. **Dataset Engineering:** Design a varied enough set of input data to test the algorithm portfolio on. Cross-reference this with the literature.

3. **Machine Learning and Feature Selection:** Apply ML techniques on various combinations of features to obtain the most optimal trained model. Will likely perform this using Jupyter Notebooks for convenience.
4. **Exporting Model:** Export the trained model coefficients or decision tree into C++.
5. **Feature Extraction Tooling:** Develop the tooling in C++ to efficiently extract features from the input data and use the trained model to select an algorithm at runtime.
6. **Empirical Testing and Formal Verification:** Expanded on below in the Evaluation section.

4 Evaluation

The system's performance will be benchmarked against traditional sorting algorithms using datasets with varying sizes and characteristics. Metrics such as execution time, memory usage, and scalability will be measured to evaluate efficiency gains.

The decision engine's predictions will be assessed by comparing the selected algorithm's performance against the optimal choice determined through exhaustive analysis. Metrics will include the percentage of correct selections and the performance gap between the chosen and optimal algorithms.

If time allows, formal theoretical verification of the problem will be attempted on to support or disprove the empirical findings.

References

[1] Vladmir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, December 1992.

[2] Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[3] Heikki Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Transactions on Computers*, C-34:318–325, 1985.

[4] Morwenn. cpp-sort: Benchmarks. `https://github.com/Morwenn/cpp-sort/wiki/Benchmarks`. Accessed: March 4, 2025.

[5] Ola Petersson and Alistair Moffat. A framework for adaptive sorting. *Discrete Applied Mathematics*, 59(2):153–179, 1995.

[6] John R. Rice. The algorithm selection problem. volume 15 of *Advances in Computers*, pages 65–118. Elsevier, 1976.