Complex Algorithm Simulator

Jack Brunswik

1/30/2026

1 Project Overview

1.1 Domain

Computational performance modeling and Monte Carlo simulation of algorithmic systems.

1.2 Problem Statement

Traditional algorithm analysis provides asymptotic bounds (e.g., $O(n \log n)$), but does not quantify variability in runtime behavior under stochastic input conditions. Real-world data typically follows probabilistic distributions, and randomized algorithms introduce inherent variability in execution behavior. This project investigates:

- How algorithm performance varies under probabilistically generated inputs
- How closely empirical results match theoretical expected-case models
- The variance and distribution of operation counts
- Sensitivity of algorithms to input distribution parameters

To answer these questions, this project implements a Monte Carlo simulation framework that repeatedly executes algorithms over randomly generated inputs and statistically analyzes performance outcomes.

1.3 Scope

This project simulates:

- Sorting algorithms (Merge Sort and Randomized Quicksort)
- Graph traversal algorithm (Breadth-First Search on random graphs $G(n, p)$)
- Operation-level performance metrics (comparisons, swaps, edge explorations)
- Statistical estimation of mean, variance, and confidence intervals

This project does **not** simulate:

- Hardware-level timing
- Cache or memory hierarchy effects
- Parallel execution or thread scheduling
- CPU benchmarking

The focus is mathematical modeling and stochastic performance behavior.

2 System Description

2.1 System Components

Input Generator

- Generates arrays or graphs from defined probability distributions
- Parameters: input size n, distribution type, distribution parameters

Algorithm Module

- Implements:
  - Merge Sort
  - Randomized Quicksort
  - Breadth-First Search (BFS)
- Counts primitive operations

Monte Carlo Engine

- Executes k independent trials
- Aggregates performance statistics

Statistical Analyzer

- Computes sample mean, sample variance, and confidence intervals
- Compares empirical results to theoretical models

2.2 System Dynamics

For each input size n:

1. Generate random input from distribution D.
2. Execute the algorithm and record operation count $C_i(n)$.
3. Repeat for k independent trials.
4. Compute statistical estimators:
   - Sample mean
   - Variance
   - Confidence interval
5. Compare empirical results to theoretical predictions.

Each trial is independent, forming a classical Monte Carlo simulation framework.

2.3 Core Mathematical Models

2.3.1 Model 1: Divide-and-Conquer Recurrence Model (Merge Sort)

Merge Sort satisfies the recurrence:

$$T(n) = 2T(n/2) + cn$$

Using the Master Theorem:

$$T(n) = \Theta(n \log n)$$

Empirical estimator of expected cost:

$$\hat{T}(n) = \frac{1}{k} \sum_{i=1}^{k} C_i(n)$$

Model validation error:

$$\epsilon(n) = \left| \hat{T}(n) - an \log n \right|$$

where a is a fitted constant.

2.3.2 Model 2: Expected Comparisons in Randomized Quicksort

Expected number of comparisons:

$$E[C_n] = 2(n+1)H_n - 4n$$

where the harmonic number is:

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

Asymptotically:

$$E[C_n] \approx 2n \ln n$$

Sample variance estimator:

$$S^2(n) = \frac{1}{k-1} \sum_{i=1}^{k} \left(C_i(n) - \hat{C}(n) \right)^2$$

2.3.3 Model 3: Probabilistic Input Models

(a) Continuous Distribution Model Uniform distribution:

$$X_i \sim U(0,1)$$

Normal distribution:

$$X_i \sim \mathcal{N}(\mu, \sigma^2)$$

Algorithm cost becomes a random variable induced by the input distribution.

(b) Random Graph Model Erdős–Rényi model:

$$G(n,p)$$

Each possible edge exists independently with probability p.
Expected number of edges:

$$E[E] = p\frac{n(n-1)}{2}$$

Since BFS runtime is $O(n+E)$, expected runtime becomes:

$$E[T(n)] = \Theta\left(n + p\frac{n(n-1)}{2}\right)$$

2.3.4 Statistical Output Analysis

Sample mean:

$$\hat{\mu}_n = \frac{1}{k}\sum_{i=1}^{k} C_i(n)$$

95% confidence interval:

$$\hat{\mu}_n \pm t_{\alpha/2,k-1}\frac{S_n}{\sqrt{k}}$$

2.3.5 Assumptions

- Inputs are independently generated.
- Primitive operations represent unit cost.

- Trials are independent and identically distributed.
- Large k ensures convergence (Law of Large Numbers).
- Constant factors are estimated empirically.

2.4 Implementation Approach

2.4.1 Programming Language

The simulation will be implemented in Python. Python is selected for the following reasons:

- Strong support for numerical computation and statistical analysis.
- Availability of scientific libraries such as `NumPy` and `SciPy`.
- Built-in pseudo-random number generation for Monte Carlo simulation.
- Rapid prototyping and readability for experimental modeling.

Python enables efficient implementation of algorithm logic while also supporting statistical post-processing and visualization.

2.4.2 Development Environment

The development environment includes:

- **IDE:** PyCharm.
- **Numerical Libraries:** NumPy for array operations and random sampling.
- **Statistical Tools:** SciPy for confidence interval computation.
- **Data Visualization:** Matplotlib for plotting empirical vs. theoretical performance curves.
- **Version Control:** Git for code management and experiment tracking.

These tools support reproducible experimentation and statistical analysis.

2.4.3 Simulation Type

The project uses a **Monte Carlo simulation** framework.

Each experiment consists of:

1. Generating stochastic input data from a specified probability distribution.
2. Executing the selected algorithm.
3. Recording operation-level performance metrics.
4. Repeating the process for k independent trials.
5. Performing statistical output analysis.

The simulation is **stochastic and discrete**, as outcomes vary due to probabilistic input generation and randomized algorithmic decisions.

2.4.4 Data Collection Plan

For each algorithm and input size n, the following metrics will be collected:

- Number of comparisons
- Number of swaps (for sorting algorithms)
- Number of edge explorations (for BFS)
- Total primitive operation count

Across k independent trials, the simulation will compute:

- Sample mean:
$$\hat{\mu}_n = \frac{1}{k}\sum_{i=1}^{k} C_i(n)$$
- Sample variance:
$$S_n^2 = \frac{1}{k-1}\sum_{i=1}^{k} (C_i(n) - \hat{\mu}_n)^2$$
- 95% confidence intervals:
$$\hat{\mu}_n \pm t_{\alpha/2,k-1}\frac{S_n}{\sqrt{k}}$$

Additionally, empirical results will be compared against theoretical models such as:

$$T(n) = \Theta(n \log n)$$

and

$$E[C_n] \approx 2n \ln n$$

to evaluate model accuracy and convergence behavior.

3 Literature Review

3.1 Core Models and Algorithms

Cormen, Leiserson, Rivest & Stein - Introduction to Algorithms
Provides:

- Master Theorem for solving recurrences
- Expected-case analysis of Randomized Quicksort
- Harmonic number approximations

The theoretical expressions used in this project are derived from these analyses and serve as baseline models for empirical validation.

Knuth - The Art of Computer Programming
Provides:

- Exact operation-count analysis
- Distributional behavior of algorithmic costs

This project follows Knuth's analytical framework but estimates cost distributions via Monte Carlo sampling.

Law & Kelton - Simulation Modeling and Analysis
Provides:

- Monte Carlo simulation methodology
- Output analysis

- Confidence interval estimation
- Experimental design principles

The simulation engine follows classical Monte Carlo methodology with statistical output analysis.

Motwani & Raghavan - Randomized Algorithms
Provides formal probabilistic analysis of randomized algorithms, including expectation and variance modeling, supporting the stochastic modeling of Randomized Quicksort.

Erdős & Rényi - Random Graph Theory
Introduced the $G(n, p)$ model used in this project to analyze expected traversal cost as a function of graph density.

3.2 Related Work

Existing algorithm visualization tools focus on deterministic step-by-step animation but do not incorporate statistical modeling or Monte Carlo performance estimation. This project extends beyond visualization by:

- Introducing probabilistic input generation
- Running repeated Monte Carlo trials
- Computing statistical estimators
- Validating theoretical models quantitatively

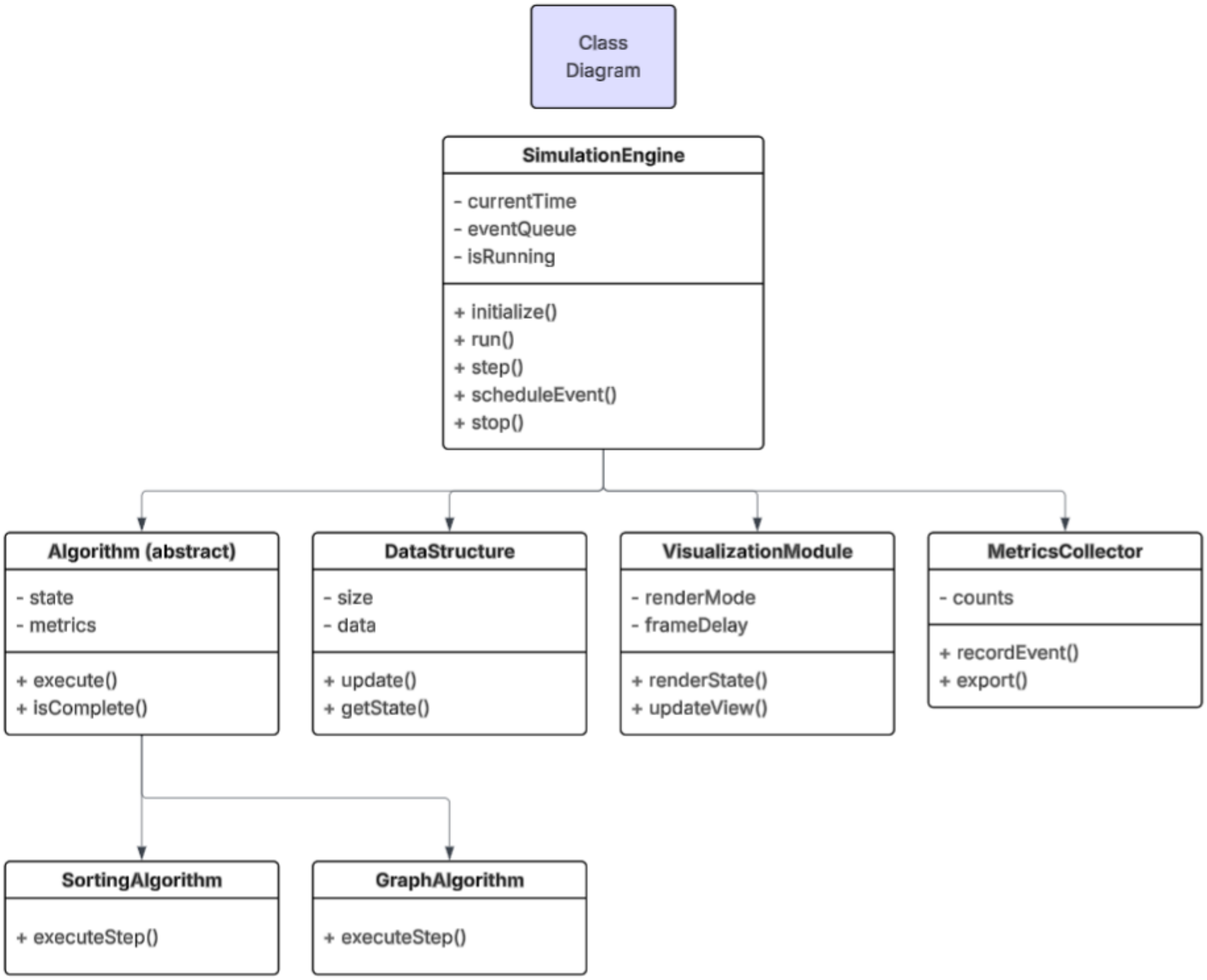

Figure 1: UML Class Diagram of the Algorithm Simulator

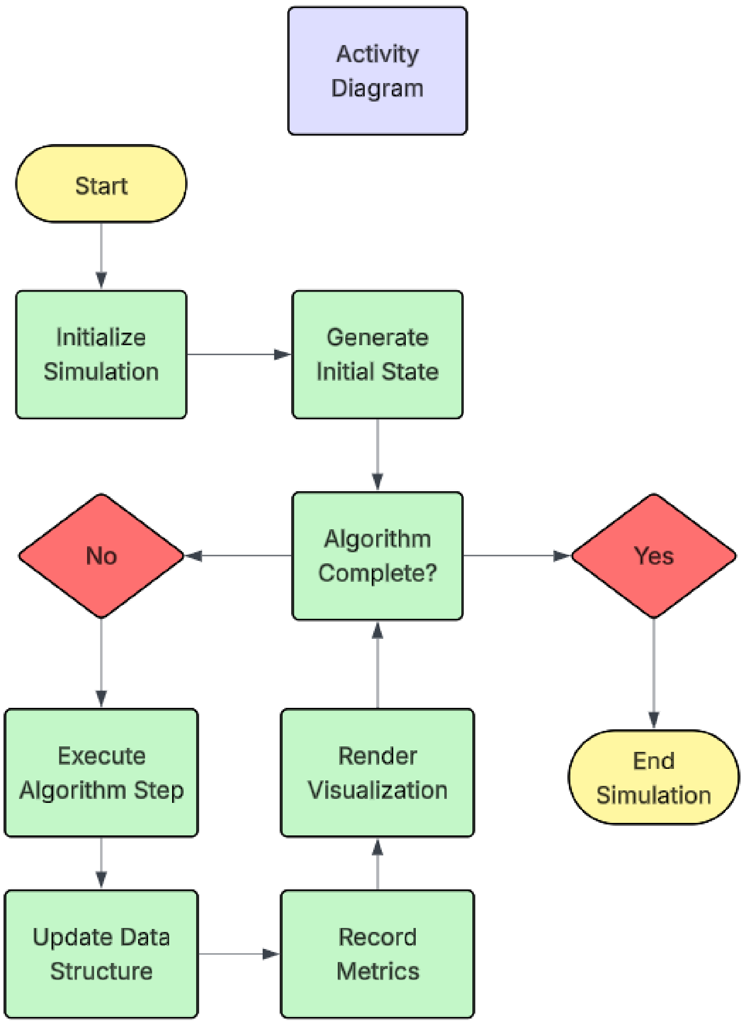

Figure 2: UML Activity Diagram of the Algorithm Simulator