QUANTUM vs CLASSICAL ALGORITHMS

Comparative Analysis and Simulation Framework

Project Documentation

Date: February 12, 2026
Version: 1.0

Table of Contents

1. Problem Statement

1.1 Background

Classical computing has dominated the computational landscape for decades, but faces fundamental limitations in solving certain classes of problems efficiently. Quantum computing promises exponential or quadratic speedups for specific algorithms through quantum mechanical phenomena such as superposition and entanglement.

1.2 Core Problem

There is a significant gap in accessible, practical demonstrations that allow students and researchers to:

- Understand the practical differences between quantum and classical algorithms
- Visualize quantum speedup through concrete implementations
- Compare performance metrics across different problem domains
- Simulate quantum algorithms without requiring quantum hardware
- Analyze scalability and complexity differences empirically

1.3 Research Questions

- What are the measurable performance differences between classical and quantum algorithms for standard problems?
- How do quantum algorithms scale compared to their classical counterparts?
- Can quantum advantage be demonstrated using simulation on classical hardware?
- What are the practical limitations of current quantum simulation frameworks?

2. Project Synopsis

2.1 Overview

This project develops a comprehensive comparison framework that implements and benchmarks five fundamental quantum algorithms against their classical counterparts. The framework provides visual analysis, performance metrics, and educational insights into quantum computing advantages.

2.2 Algorithms Implemented

Algorithm	Classical Complexity	Quantum Complexity	Speedup Type
Factorization (Shor's)	$O(\exp(\sqrt[3]{(\log N)}))$	$O((\log N)^3)$	Exponential
Search (Grover's)	$O(N)$	$O(\sqrt{N})$	Quadratic
Deutsch-Jozsa	$O(N/2)$	$O(1)$	Exponential
Min/Max Finding	$O(N)$	$O(\sqrt{N})$	Quadratic
Phase Estimation	$O(2^n)$	$O(n^2)$	Exponential

2.3 Key Features

- Side-by-side implementation of classical and quantum algorithms
- Performance benchmarking with statistical analysis
- Interactive visualizations and circuit diagrams
- Scalability analysis across different input sizes
- Comprehensive documentation and educational materials

2.4 Target Audience

- Computer Science students learning quantum computing
- Researchers comparing algorithmic approaches
- Educators teaching quantum computing concepts
- Industry professionals evaluating quantum computing potential

3. Scope of the Project

3.1 In Scope

- Implementation of 5 quantum algorithms with classical equivalents
- Simulation using Qiskit quantum computing framework
- Performance metrics: time complexity, query counts, success rates
- Visualization tools: circuit diagrams, probability distributions, comparison charts
- Comprehensive documentation and user guide
- Analysis of scalability from small to medium input sizes
- Educational materials explaining quantum concepts

3.2 Out of Scope

- Implementation on actual quantum hardware
- Algorithms requiring more than 20 qubits
- Error correction and fault-tolerant quantum computing
- Quantum cryptography protocols
- Commercial-scale optimization problems
- Real-time quantum computing applications
- Advanced quantum algorithms (VQE, QAOA with large circuits)

3.3 Deliverables

- Complete source code repository with documentation
- Executable simulation framework
- Performance analysis reports with visualizations
- User manual and installation guide
- Technical documentation explaining implementation
- Presentation slides for demonstration
- Test cases and validation results

3.4 Constraints

- Limited to simulation on classical hardware
- Computational resources: 16GB RAM, modern multi-core processor
- Input sizes limited by simulation overhead
- Development timeframe: 4-6 weeks
- Technologies: Python 3.8+, Qiskit, Matplotlib, NumPy

4. Feasibility Analysis

4.1 Technical Feasibility

Technical Assessment: HIGHLY FEASIBLE

Strengths:

- Qiskit is mature, well-documented, and actively maintained by IBM
- All algorithms are well-established with known implementations
- Simulation requires only classical computing resources
- Python ecosystem provides excellent visualization libraries
- No specialized hardware requirements

Challenges:

- Simulation complexity grows exponentially with qubit count
- Limited to ~20 qubits on standard hardware
- Shor's algorithm requires careful implementation for stability
- Performance benchmarking requires statistical rigor

4.2 Operational Feasibility

Operational Assessment: FEASIBLE

- Development team requires knowledge of quantum computing basics and Python
- Qiskit learning curve is manageable (1-2 weeks for basics)
- Project can be completed in 4-6 weeks with dedicated effort
- Testing and validation are straightforward with known results
- Documentation can be generated alongside development

4.3 Economic Feasibility

Economic Assessment: HIGHLY FEASIBLE

Resource	Cost	Notes
Development Tools	$0	Python, Qiskit, VS Code - all free
Computing Resources	$0	Standard laptop/desktop sufficient
Cloud Services	$0	Optional; free tiers available
Learning Resources	$0	Free documentation and tutorials
Total Investment	$0	Zero-cost implementation

4.4 Schedule Feasibility

Phase	Duration	Deliverables

Learning & Setup	1 week	Environment setup, Qiskit basics
Algorithm Implementation	2 weeks	All 5 algorithms working
Visualization & Analysis	1 week	Charts, metrics, comparisons
Testing & Validation	1 week	Test cases, bug fixes
Documentation	1 week	User guide, technical docs

4.5 Risk Analysis

Risk	Probability	Impact	Mitigation
Performance issues	Medium	Low	Limit input sizes
Implementation bugs	Medium	Medium	Extensive testing
Learning curve	Low	Low	Use tutorials
Scope creep	Medium	High	Strict scope control

Conclusion: **The project is HIGHLY FEASIBLE with minimal risks and zero financial investment.**

5. System Design

5.1 Architecture Overview

The system follows a modular architecture with clear separation of concerns:

- Algorithm Module: Contains implementations of all quantum and classical algorithms
- Simulation Engine: Handles quantum circuit execution using Qiskit
- Benchmarking Module: Measures performance metrics and statistical analysis
- Visualization Module: Generates charts, graphs, and circuit diagrams
- Comparison Engine: Aggregates results and performs comparative analysis
- User Interface: Command-line and web-based interfaces for interaction

5.2 Technology Stack

Layer	Technology	Purpose
Core Language	Python 3.8+	Primary development language
Quantum Framework	Qiskit	Quantum circuit simulation
Classical Computing	NumPy, SciPy	Classical algorithm implementation
Visualization	Matplotlib, Seaborn	Data visualization
Testing	pytest, unittest	Unit and integration testing
Documentation	Sphinx, Markdown	Code and user documentation

5.3 Data Flow

The system processes data through the following pipeline:

1. Input: User specifies algorithm, problem size, and parameters

2. Classical Execution: Classical algorithm runs and metrics are recorded

3. Quantum Circuit Construction: Quantum circuit is built based on problem

4. Quantum Simulation: Qiskit simulator executes the quantum circuit

5. Results Collection: Both classical and quantum results are gathered

6. Analysis: Performance metrics are calculated and compared

7. Visualization: Charts and graphs are generated

8. Output: Results displayed to user with comparative analysis

5.4 Design Patterns

- Strategy Pattern: Algorithms implement a common interface for easy switching
- Factory Pattern: Algorithm instances created based on user selection

- Observer Pattern: Performance metrics updated during execution
- Template Method: Common benchmarking workflow with algorithm-specific steps
- Singleton Pattern: Configuration and logging managers

6. Implementation Details

6.1 Project Structure

quantum-vs-classical/

```
│
├── algorithms/
│   ├── classical/
│   │   ├── factorization.py
│   │   ├── search.py
│   │   ├── deutsch_jozsa.py
│   │   ├── min_max.py
│   │   └── phase_estimation.py
│   │
│   ├── quantum/
│   │   ├── shor.py
│   │   ├── grover.py
│   │   ├── deutsch_jozsa.py
│   │   ├── quantum_search.py
│   │   └── qpe.py
│   │
│   └── base.py
│
├── simulation/
│   ├── quantum_simulator.py
│   ├── benchmarking.py
│   └── metrics.py
│
├── visualization/
│   ├── plots.py
│   ├── circuits.py
│   └── dashboard.py
│
├── tests/
│   ├── test_classical.py
│   ├── test_quantum.py
│   └── test_comparison.py
│
├── results/
│   ├── metrics.csv
│   └── plots/
│
├── docs/
│   ├── user_guide.md
│   └── technical_docs.md
│
├── main.py
├── config.py
```

```
├── requirements.txt
└── README.md
```

6.2 Key Implementation Details

Shor's Algorithm:

- Implements period finding using Quantum Fourier Transform
- Classical post-processing for factor extraction
- Optimized for numbers up to 21 for simulation feasibility

Grover's Algorithm:

- Oracle construction for marked item detection
- Amplitude amplification iterations: ceil(π/4 * √N)
- Success probability > 99% with optimal iterations

Deutsch-Jozsa:

- Determines if function is constant or balanced
- Single query regardless of input size
- Demonstrates exponential quantum advantage

6.3 Performance Optimization

- Use of Aer simulator for efficient quantum simulation
- Parallel execution of independent test cases
- Caching of frequently used quantum circuits
- Efficient NumPy operations for classical algorithms
- Memory management for large-scale simulations

7. Expected Outcomes

7.1 Quantitative Results

Algorithm	Input Size	Expected Speedup	Success Rate
Shor's (N=15)	4 qubits	1.5-2x	~100%
Grover (N=16)	4 qubits	4x	~99%
Deutsch-Jozsa	5 qubits	16x	100%
Min Finding (N=16)	4 qubits	4x	~85%
Phase Estimation	8 qubits	Exponential	~95%

7.2 Qualitative Outcomes

- Clear demonstration of quantum advantage for specific problem classes
- Visual understanding of quantum circuit complexity
- Educational tool for teaching quantum computing concepts
- Baseline for future quantum algorithm research
- Framework extensible for additional algorithms

7.3 Learning Outcomes

- Deep understanding of quantum algorithm design principles
- Practical experience with quantum simulation frameworks
- Skills in performance benchmarking and statistical analysis
- Knowledge of when quantum computing provides advantages
- Ability to implement and optimize quantum circuits

7.4 Future Extensions

- Integration with real quantum hardware (IBM Quantum, AWS Braket)
- Additional algorithms (VQE, QAOA, Quantum Annealing)
- Web-based interactive dashboard
- Machine learning classification using quantum circuits
- Comparative analysis with other quantum frameworks (Cirq, PennyLane)

8. Conclusion

This project provides a comprehensive framework for understanding and demonstrating the advantages of quantum computing through practical implementation and simulation. By comparing five fundamental quantum algorithms with their classical counterparts, the project delivers measurable insights into quantum speedup, algorithm efficiency, and the potential of quantum computing.

The feasibility analysis confirms that the project is highly achievable with zero financial investment, manageable technical complexity, and clear deliverables. The modular design ensures extensibility for future research while providing immediate educational and analytical value.

Expected outcomes include quantitative performance comparisons, visual demonstrations of quantum advantage, and a robust framework that can serve as both an educational tool and a research platform for quantum algorithm development.