# Quantum Arithmetic Algorithms: Implementation, Resource Estimation, and Comparison

Dmytro Fedoriaka, Brian Goldsmith, Yingrong Chen

September 4, 2025

# Motivation & Background

## Why Quantum Arithmetic Algorithms?

Critical components of more complex algorithms (e.g., Shor's algorithm).

## Challenges

- Limited implementations and tests.
- Limited availability in quantum programming packages.
- Comparisons remain largely theoretical.
- Discrepancies between theoretical and real-world implementations.

# Project Outline

- Implement and test 23 quantum arithmetic algorithms.
  - Available as an external Q# library: https://github.com/fedimser/quant-arith-re.
- Perform resource estimation to:
  - Choose the best algorithm for each arithmetic operation.
  - Explore space-time trade-offs.
  - Select optimal subcomponents.
  - Fine-tune algorithm parameters.
  - Identify crossover points where one algorithm outperforms another.
  - Analyze asymptotic complexity.

**Aim**

This presentation aims to provide a codebase and knowledge base that researchers can leverage to build more complex quantum applications. It also demonstrates how resource estimation can be applied in quantum research and software engineering.

# Algorithms Overview

- In-place addition: $U\left|a\right\rangle\left|b\right\rangle = \left|a\right\rangle\left|(a+b) \bmod 2^n\right\rangle .$
- Out-of-place addition: $U\left|a\right\rangle\left|b\right\rangle\left|0\right\rangle = \left|a\right\rangle\left|b\right\rangle\left|(a+b) \bmod 2^n\right\rangle .$
- Quantum-classical addition: $U_a\left|b\right\rangle = \left|(a+b) \bmod 2^n\right\rangle .$
- Multiplication: $U\left|a\right\rangle\left|b\right\rangle\left|0\right\rangle = \left|a\right\rangle\left|b\right\rangle\left|a \cdot b\right\rangle .$
- Division: $U\left|a\right\rangle\left|b\right\rangle\left|c\right\rangle = \left|a \bmod b\right\rangle\left|b\right\rangle\left|a/c\right\rangle .$
- Modular exponentiation: $U_{a,N}\left|x\right\rangle\left|0\right\rangle = U_{a,N}\left|x\right\rangle\left|a^x \bmod N\right\rangle .$

In addition, we implemented incrementers, comparators, table lookups, a square root operator, and a greatest common divisor.

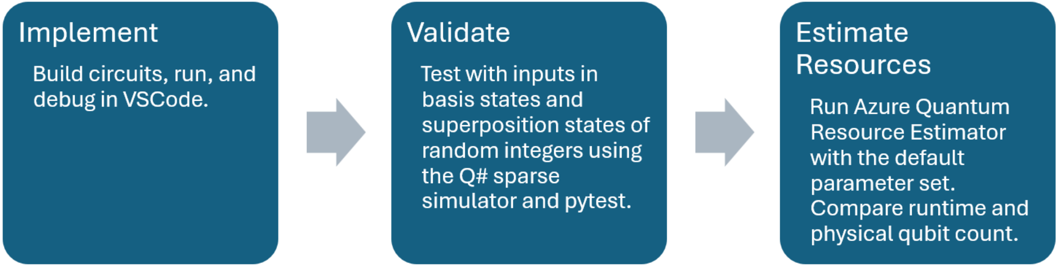
# In-Place, Out-of-Place, Quantum-Classical Adders & Subtractors

| **Year** | **Algorithm(s)** |
|---|---|
| 2000 | QFT-based Adder *[Draper, 2000]* |
| 2002 | Ripple-Carry Adder and Ripple-Borrow Subtractor *[Cheng, Tseng, 2002]* |
| 2004 | Ripple-Carry Adder with One Ancilla *[Cuccaro et al., 2004]* |
| | Carry-Lookahead Adder *[Draper et al., 2004]* |
| 2009 | Ripple-Carry Adder with Zero Ancilla *[Takahshi et al., 2009]* |
| | Ripple–Borrow Subtractor *[Thapliyal, Ranganathan, 2009]* |
| 2012 | QFT-based Quantum-Classical Adder *[Pavlidis, Gizopoulos, 2012]* |
| 2013 | Ripple-Carry Adder *[Thapliyal, Ranganathan, 2013]* |
| | Incrementer *[Li et al., 2013]* |
| 2016 | Ripple-Carry Adder *[Wang et al., 2016]* |
| 2018 | Ripple-Carry Adder with Logical-AND *[Gidney, 2018]* |
| 2021 | Ripple-Carry Adder *[Gayathri et al., 2021]* |
| 2023 | Carry-Lookahead Ling Base Adder *[Wang, Chattopadhyay, 2023]* |
| 2024 | Carry-Lookahead Higher Radix Adder *[Wang et al., 2024]* |
| 2025 | Quantum–Classical Adder *[Fedoriaka, 2025]* |

# Mulitpliers, Dividers, and Modular Exponentiation

| **Year** | **Algorithm(s)** |
| --- | --- |
| 2011 | Restoring Divider *[Khosropour et al., 2011]* |
| 2012 | Quantum-Classical QFT Mulitplier, Granlund–Montgomery Divider, and Modular Exponentiation *[Pavlidis, Gizopoulos, 2012]* |
| 2013 | Greatest Common Divisor *[Saeedi, 2013]* |
| 2016 | Shift–and–Add Mulitplier *[Jayashree et al., 2016]* |
| 2017 | Shift–and–Add Mulitplier *[Muñoz–Coreas et al., 2017]* |
| 2018 | Non-Restoring Square Root *[Muñoz–Coreas, Thapliyal, 2018]* |
| 2019 | Karatsuba Multiplier *[Gidney, 2019]*<br>Windowed Multiplier and Modular Exponentiation *[Gidney, 2019]*<br>Restoring and Non–Restoring Divider *[Thapliyal et al., 2019]* |
| 2021 | Modular Multiplier and Modular Exponentiation *[Liu et al., 2021]* |
| 2023 | Wallace Tree Multiplier*[Orts et al., 2023]* |

# Development Workflow




Workflow to ensure that the implementations are correct and optimal.

# Usage Example: $6 \times 7$

Main.qs

lib > src > Main.qs

```
import TestUtils.ApplyBigInt;
import TestUtils.MeasureBigInt;
import QuantumArithmetic.Utils;
import QuantumArithmetic.MCT2017.Multiply;

Run | Histogram | Estimate | Debug | Circuit
operation Main() : Unit {
    use X = Qubit[3];
    use Y = Qubit[3];
    use Z = Qubit[6];
    ApplyBigInt(6L, X);
    ApplyBigInt(7L, Y);
    QuantumArithmetic.MCT2017.Multiply(X, Y, Z);
    let result = MeasureBigInt(Z);
    Message($"{result}");
    ResetAll(X);
    ResetAll(Y);
}
```

QDK Circuit Main.Main

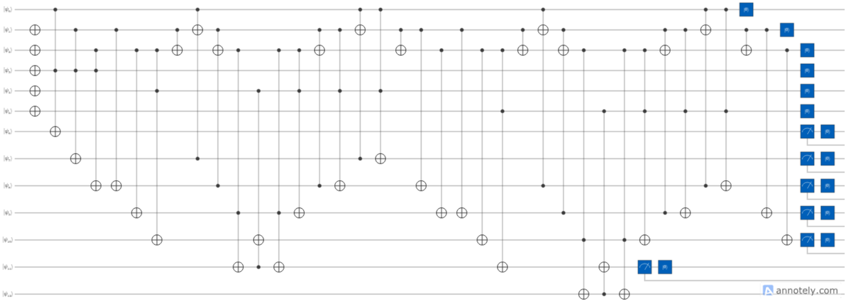
## Main.Main with 0 input qubits (Trace)

Target profile: QIR unrestricted

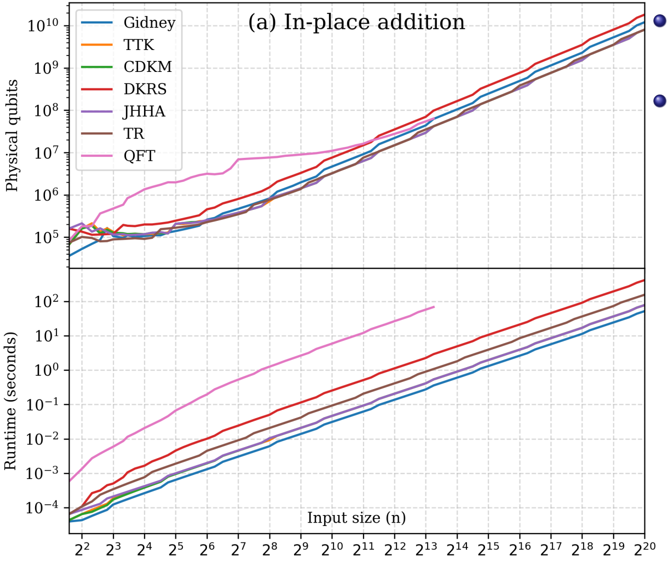
WARNING: This diagram shows the result of tracing a dynamic circuit, and may change from run to run.

Learn more at https://aka.ms/qdk.circuits

Zoom 50 %



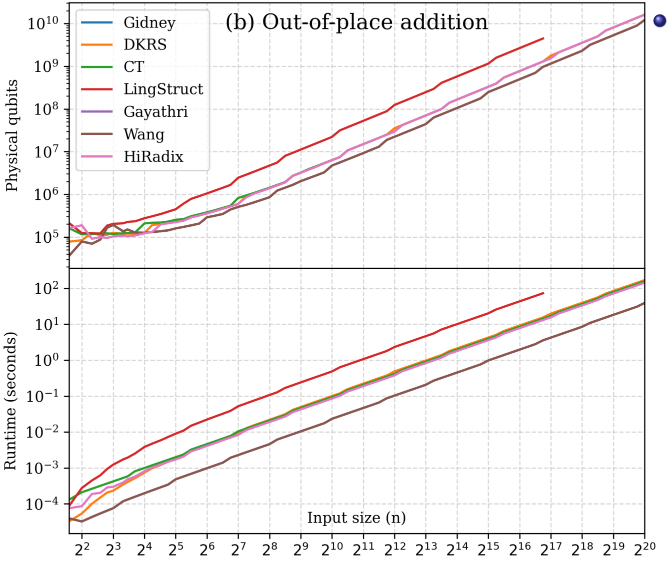
# Resource Estimation: In-Place Addition




- Fewest qubits: TTK, CDKM, JHHA, TR.
- Fastest: Gidney adder.


[Gidney] C. Gidney, "Halving the cost of quantum addition", 2018.
[TTK] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out," 2009.
[CDKM] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit", 2004.
[DKRS] T. G. Draper, S. A. Kutin, E. M. Rains, K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder, 2006.
[JHHA] H. Jayashree, H. Thapliyal, H. R. Arabnia, and V. K. Agrawal, "Ancilla-input and garbage-output optimized design of a reversible quantum integer multiplier", 2016.
[TR] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic-based binary and BCD adder circuits", 2013.
[QFT] T. G. Draper, "Addition on a quantum computer," 2000.

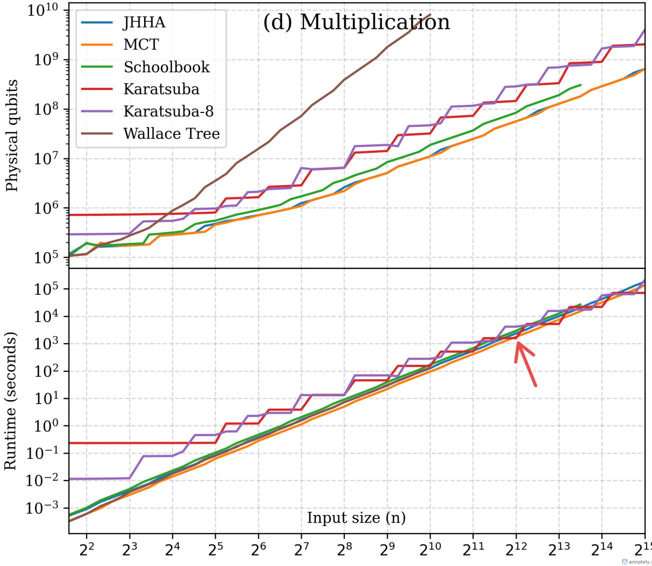
# Resource Estimation: Out-of-Place Addition




- Least qubits and fastest: Gidney, Gayarthi, Wang.


[Gidney] C. Gidney, "Halving the cost of quantum addition", 2018.
[DKRS] T. G. Draper, S. A. Kutin, E. M. Rains, K. M. Svore, "A logarithmic-depth quantum carry-lookahead adder", 2006.
[CT] K.-W. Cheng, C.-C. Tseng, "Quantum full adder and subtractor", 2002.
[LingStruct] S. Wang, A. Chattopadhyay, "Reducing depth of quantum adder using Ling structure", 2023.
[Gayathri] S. Gayathri, R. Kumar, S. Dhanalakshmi, B. K. Kaushik, and M. Haghparast, "T-count optimized wallace tree integer multiplier for quantum computing", 2021.
[Wang] F. Wang, M. Luo, H. Li, Z. Qu, and X. Wang, "Improved quantum ripple-carry addition circuit", 2016.
[HiRadix] S. Wang, A. Baksi, A. Chattopadhyay, "A higher radix architecture for quantum carry-lookahead adder", 2023.

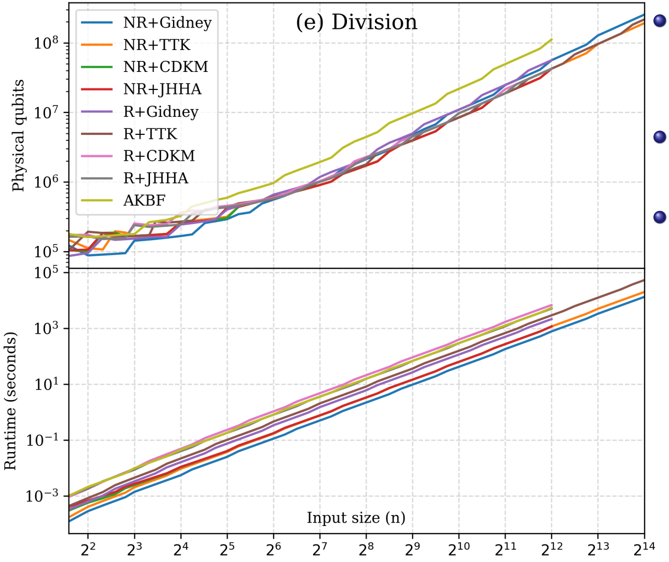
# Resource Estimation: Multiplication




- Simple Shift-And-Add ($O(n^2)$) faster for $n \leq 2^{12} \approx 4000$.
- The fastest is MCT [1].
- Karatsuba [2] ($O(n^{1.58})$) faster for some $n$ starting from $n \geq 2^{12}$, but slower for non-powers-of-2 because of padding.
- Karatsuba consistently faster starting with $n = 2^{18} \approx 260000$.
- Wallace Tree [3] has runtime comparable to Shift-And-Add but requires much more auxiliary qubits.


[1] E. Munoz-Coreas and H. Thapliyal, "T-count optimized design of quantum integer multiplication", 2017.
[2] C. Gidney, "Asymptotically Efficient Quantum Karatsuba Multiplication", 2019.
[3] F. Orts et al., "Improving the number of T gates and their spread in integer multipliers on quantum computing", 2023.

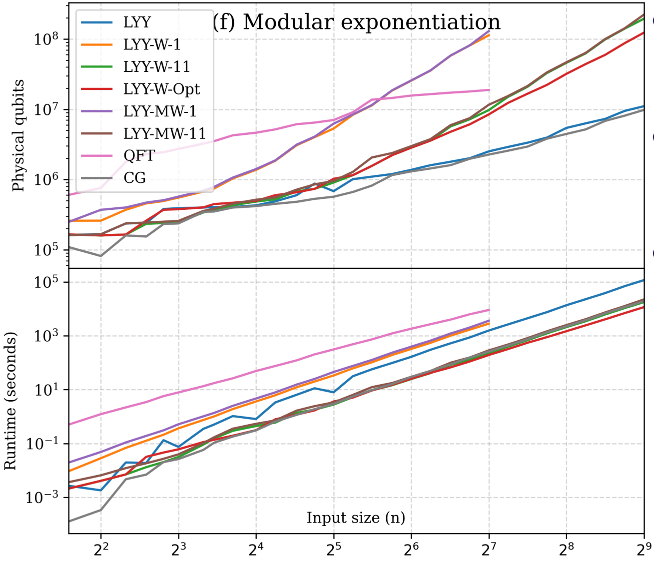
# Resource Estimation: Division




- Design Space Exploration: tried different division algorithms with different in-place adders as subcircuits.
- Fastest: restoring division from [1] using Gidney adder [2].
- Fewest qubits: non-restoring division [1] using TTK adder [3].


[1] H. Thapliyal, E. Munoz-Coreas, T. Varun, and T. S. Humble, "Quantum circuit designs of integer division optimizing T-count and T-depth" 2019.
[2] C. Gidney, "Halving the cost of quantum addition," 2018.
[3] Y. Takahashi, S. Tani, and N. Kunihiro, "Quantum addition circuits and unbounded fan-out", 2009.

# Resource Estimation: Modular Exponentiation




- Windowing technique optimizes resource usage. Optimal window size depends on $n$.
- Fastest: LYY-W-Opt, windowed algorithm from [1] with optimal window size $w^* \approx 2\log_2(n)$.
- Fewest qubits: CG [2], window size 2.

[1] X. Liu, H. Yang, and L. Yang, "CNOT-count optimized quantum circuit of the Shor's algorithm", 2021.
[2] C. Gidney. "Windowed quantum arithmetic", 2019.

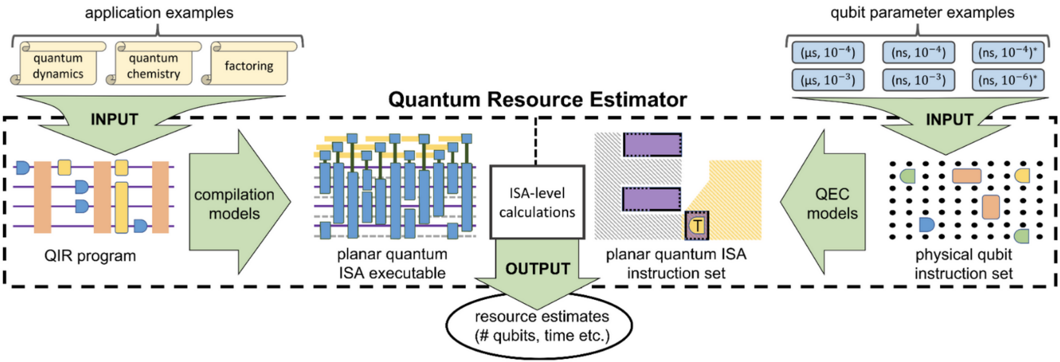
LYY-**W** - windowed.
LYY-**MW** - uses Montgomery modular multiplication.

# Results: Asymptotic Runtime Complexity Analysis

- Plot runtime as function of n in log-log scale.
- Use linear regression to fit to $T(n) = C \cdot n^a$.

| **Algorithm** | **a** | **Theoretical** |
|---|---|---|
| Adders | 1.07..1.12 | 1 |
| Shift–and–Add multipliers | 2.10 | 2 |
| Karatsuba multiplier | 1.76 | $\log_2 3 \approx 1.58$ |
| Dividers | 2.10..2.12 | 2 |
| ModExp (with optimal window size) | 2.97 | 3 |

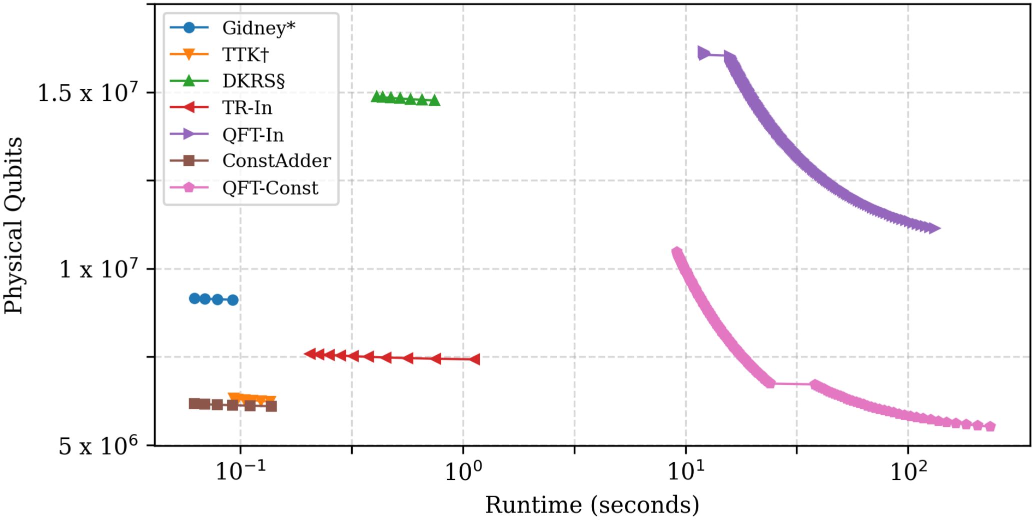
# Resource Estimation Methods



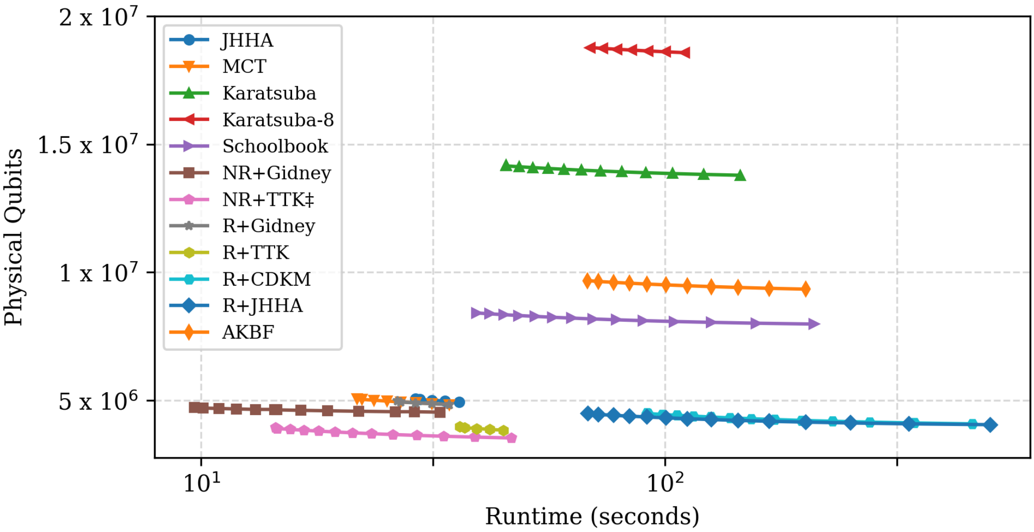

[1] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, and A. Vaschillo, "Assessing requirements to scale to practical quantum advantage," 2022.

- Physical qubits model: gate-based, superconducting qubit model.
- Error budget: 0.001.
- QEC: gate-based, surface code.

# Pareto Frontier Analysis for Adders



# Pareto Frontier Analysis for Dividers and Multipliers



# Software and Feedback

## QDK and Q# used to implement the algorithms

- Enforces clean auxiliary qubits.
- Automatically generates adjoint and controlled variances of operations.
- Runtime debugger is extremely helpful.
- AzureQRE used for resource estimation.

## Python for resource estimation and testing

- qsharp Python module used for generating and analyzing results.
- pytest Python module used for testing.

# Lessons from Implementations

### Things to remember when implementing algorithms

- Parallel Qubit Operations - Treated sequentially by AzureQRE.
- Reset and Measurement Operations - Disrupt entanglement.
- Uncomputation - Required to release auxiliary qubits.

# Acknowledgements

Quantum Open Source Foundation (qosf.org)

Mariia Mykhailova

Github Repo: