Quantum Bogosort

Chicago STEM Fair, Design

George Huebner (Grade 12), Michael Caines (Teacher Sponsor)

March 18, 2022

Walter Payton College Prep

Introduction

Abstract

Gag algorithms are a staple of computer science and provide many a good chuckle. Take, for instance, MiracleSort[1], an algorithm that relies on alpha particle emission to cause erroneous bit flips to sort the list:

```
while isSorted(unsorted_list) is False:
    time.sleep(1000)
```

Despite their apparent uselessness, joke algorithms provide valuable insight into algorithm design and complexity[2]. In this paper we propose an implementation to Quantum Bogosort, one such joke algorithm.

What is Quantum Bogosort?

Classical Bogosort is a sorting algorithm.

1. Randomly permute the list.
2. If the list is sorted, done! Otherwise go to step 1.

This has unbounded worst-case time complexity and an expected time complexity of $O(n \cdot n!)$.

Quantum Bogosort is very similar to classical Bogosort, except it uses a quantum computer to permute the list.

Attachments

Code + Paper: github.com/Borris-the-real-OG/Quantum_Bogosort

Acknowledgements

- Qiskit
- *We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.*[3]
- All code was written and tested with Qiskit 0.32.0 & Python 3.8 on WSL2 Ubuntu 20.04.1.
- Special thanks to Joe Clapis

Purpose

Just like its classical counterpart, Quantum Bogosort is pretty useless. In fact, classical sorting algorithms have been proven to have a lower bound time complexity of $\Omega(N \log_2 N)$: there's nothing to optimize!

However, QBS presents a good case study for quantum algorithms; we present an analysis of implementation and complexity of QBS in a similar vein to Gruber et al.'s[2] analysis of classic bogosort.

Algorithm Design

QRNG

Desired State:

$$\sum_{x=0}^{N-1} \frac{1}{\sqrt{N}} |x_{BE}\rangle$$

This is quite easy to do with powers of 2; we just need to use Hadamard gates with control qubits to chunk a big number into smaller subproblems.

$$|\psi\rangle = \frac{1}{\sqrt{2}} |0000\rangle + \frac{1}{\sqrt{2}} |1000\rangle$$

$$\mathbf{CH}(\psi_{\mathbf{0}}, \psi_{[\mathbf{1},\mathbf{3}]}),\ \mathbf{X}(\psi_{\mathbf{0}})$$

$$|\psi\rangle = \frac{1}{\sqrt{16}} (|0000\rangle + |0001\rangle + \ldots + |0111\rangle) + \frac{1}{\sqrt{2}} |1000\rangle$$

We mostly care about amplitude as opposed to register state

QRNG

Limitation: Hadamard gates create the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.
What about states that aren't powers of 2?

Solution: Arbitrary rotation gates allow for qubit superposition with arbitrary amplitudes.

Hadamard is just a rotation of $\frac{\pi}{2}$ around the Pauli Y-axis!

$$|\psi\rangle = |0\rangle$$

$$\mathrm{R_y}(\theta)\,|\psi\rangle = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} |\psi\rangle = -\sin\frac{\theta}{2}|0\rangle + \cos\frac{\theta}{2}|1\rangle$$

Everything in this project is measured in Pauli-Z basis, so we don't worry about phase or imaginary numbers.

Algorithm

1. If 'parts' $= 2^{\texttt{len(register)}}$, use the control register to apply multi-controlled Hadamard gates to the rest of the register. Done.
2. If 'parts' $\leq 2^{\texttt{len(register)}-1}$, add `register[0]` to the control register and skip to step 5.
3. Calculate θ, $\mathrm{R_y}(\theta)$ the first register qubit, and add it to the control register.
4. Using the control register, apply multi-controlled Hadamard gates to the rest of the register.
5. Recurse on the remaining qubits.

Worked Example

$$i = 77,\ |\psi\rangle = |0000000\rangle$$

$$\theta_0 = 2 \arccos \sqrt{\frac{77 - 64}{77}}$$

Most Significant Bit in register

$$\mathsf{R}_y(\theta_0, \psi_0),\ \mathsf{X}(\psi_0),\ \mathsf{CH}(\psi_0, \psi_{[1,6]}),\ \mathsf{X}(\psi_0) =$$

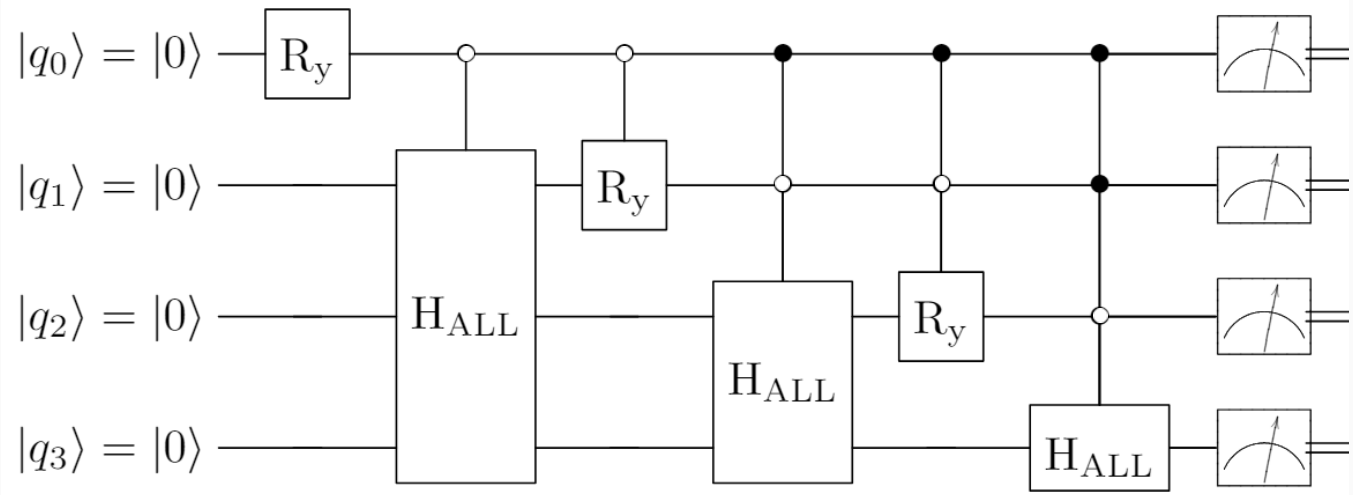
$$\frac{1}{\sqrt{77}}(|0000000\rangle + |0000001\rangle + ... + |0111111\rangle) + \sqrt{\frac{13}{77}}\,|1000000\rangle$$

Done with this part

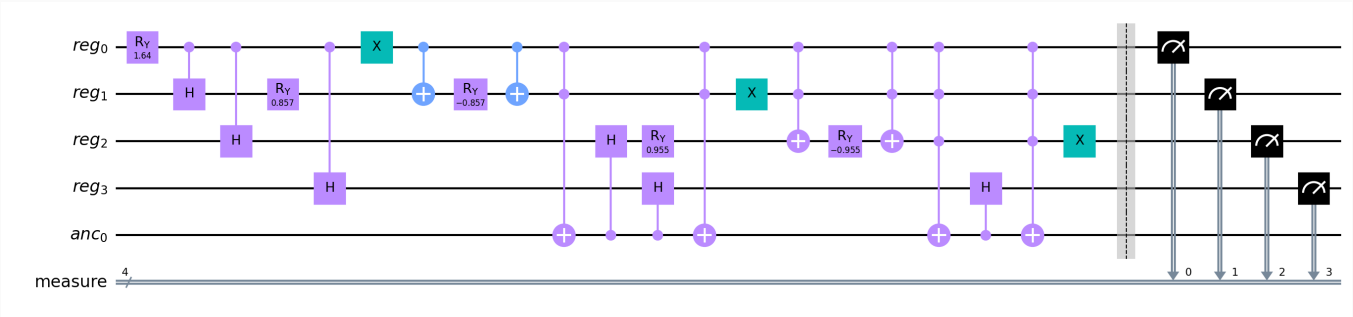
New subproblem

Practicality Assessment/Analysis

Circuit

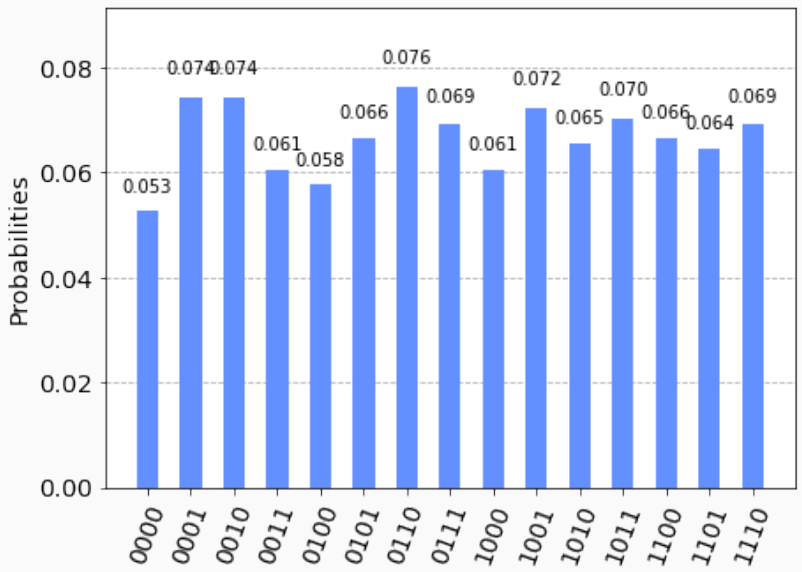


1 Conceptual Circuit[4]



2 Compiled to Aer Simulator

Results



Example measurement with `parts = 15`.

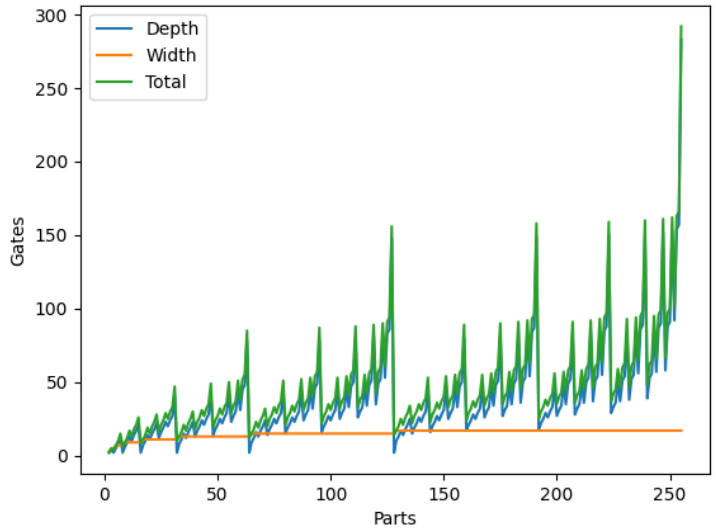
Complexity

$n = \#$ elements, $o =$ `bin(n).count('1')`

Many-worlds: $O(o)$

Copenhagen: $O(o \cdot n!)$

Gate count scales with the number of '1's in the binary representation, **not** the size of the number.



Circuit depth, width, and total gate count in relation to i.

Conclusion

Quantum Bogosort is not a useful algorithm, but it's helpful in teaching quantum computing fundamentals in a fun way.
Although infeasible due to its ballooning gate count, the QRNG algorithm could be repurposed for quickly preparing evenly balanced states among qubits.
By trying to optimize around the abysmal $O(\log_2 n!)$ space complexity via 'dividing and conquering', we end up implementing quicksort.

Bibliography

[1] K. Thompson, "Are there any worse sorting algorithms than bogosort (a.k.a monkey sort)?," Feb 2013.

[2] H. Gruber, M. Holzer, and O. Ruepp, "Sorting the slow way: An analysis of perversely awful randomized sorting algorithms," *Lecture Notes in Computer Science Fun with Algorithms*, p. 183–197, 2007.

[3] IBM Quantum, 2021.

[4] I. Chuang, "qasm2circ."