QC

1. **A classical problem**

In 1993, Yao proposed [2] an alternative to Deutsch's original model of quantum computing, which seems to be inspired by Feynman's older idea. However, the success of the quantum circuit model has led to most practitioners abandoning all associations with classical computing; this appears to have been mainly prompted by

1) the formal differences between classical and quantum logic gates
2) the restraint of NISQ-era memory

This confusion is perhaps best indicated by the strange situation whereby we find ourselves using programming languages which contain separate statements for the application of a logic gate to a single qubit! In fact, the name "OpenQASM" given by IBM to its low-level interface, when seen in this light, is a misnomer. It is not an assembly language, and is, rather, a *hardware description language*.

However, we may mitigate this circumstance by considering the role of classical gates in classical computing. Theoretically, we may formulate our inferences in the following manner. We know that the combination of the gates and the qubits provided by IBM does not constitute even a linear bounded automaton. In fact, even in the context of probabilistic machines, these fail to be Turing complete. How, then, does classical computing rely on classical logic? The answer lies in the stored program paradigm, and, in general, the design of modern ALUs. A bridge between Turing machines and modern day ALUs is provided by Minsky [3] and Cook and Reckhow [4] whose "RASP instruction set", it can be seen, effects only finite changes in state, which can occur on a DFA. Their Turing-completeness is only assured by the ability of programs written with it to modify other programs, and thus access ever increasing (or decreasing) memory addresses.

But how do we discuss a "quantum stored program computer" rationally, without losing hold of at least a little of our senses? The answer to that too was provided by an artifact of another stage in our system's development. This was a stage in which we attempted to assess what circumstances would justify augmenting quantum processing with a little classical computing per qubit. It was decided that, given the cost of each qubit, we may easily associate a few bits more with each qubit, to help us save the phases of each projection $\langle 0| q \rangle$ and $\langle 1| q \rangle$. This seems absurd, but for the imposition of a few conditions:

1) No qubits may be entangled
2) Only the gates H, S, T, and X may be applied to a single qubit
3) A maximum of two Hadamard gates may be applied to any qubit

By restricting ourselves further, we have, indeed given ourselves more room to use quantum computers without being distracted by baroque instruction sets (logic gate selections, to be precise). Effectively, we separate concerns and increase productivity. Under the above conditions, the phase of each of the qubit projections are always integer multiples of $\pi/8$. This means that the phase (for each projection) can take upto $2\pi/(\pi/8) = 16$ values. This means that *only four bits* are required to encode the phase of each qubit projection, and therefore only 8 bits are needed for an entire qubit.

Meyer [5] has shown that entanglement can be bypassed, at the cost of greater memory use, but the above scheme allows for such extensive preprocessing, that these may no longer present such a serious challenge to the use of his suggestion.

Just on the off chance that it was not clear, we close this section by explicitly proposing that the phase be computed by a classical hardware circuit which can be easily drawn out by processing the specification of any given quantum circuit (in our implementation this is done in software) and the norm of each qubit is found by actually applying the circuit in the specification.

3. **A very interesting preprocessor**

The development of the aforementioned system also saw the implementation of a very interesting suggestion to deal with the memory restrictions of NISQ devices. Taking the cue from the designs of modern ALUs [6] as well as the ISA of Cook and Reckhow, we may imagine a mapping between the instruction set of some hypothetical, large quantum computer, and a large logic circuit- *only with quantum gates instead of classical ones*. However, the crux of the suggestion was in the implementation of linear bounded automata as the logic circuits of each

machine instruction- thus increasing the power of the ISA by a great amount. This is, of course, slightly impractical, given the size of extant machines. However, on making the memory of the LBA *circular*, we may not be far off.
But how, then, do we use the existing scheme (eg. IBM's) to make LBAs with circular memory, if all we have to deal with are static logic gates? There is the suggestion of an answer in Yao(1993).
The halting problem prevents us from determining the actual time it takes for a TM to terminate, but if there is already a well known asymptotic formula for the time complexity of the run of the LBA, then we may employ a formula of GH Hardy in "Orders of Infinity"- a text on the "Infintercalcul" of Paul Du Bois Reymond, in which Hardy gives an explicit construction of an analytic function that is always $\Omega (\varphi)$, given φ. Applying this to the time complexity of the algorithm, T(n), we may employ currying to obtain a polynomial approximation (see Fig 1.)
When we do this to argue a simple quantum- LBA, the implementation takes a surprisingly simple form:

```
import math as m

def G ( φ, N ):
    def  f (  x ):
        s ← 0
        for i in range( N):
            i ← i + 2
            ν ← m.floor ( m.log( φ( ( i + 1)**2 )**2 ) / m.log ( i ) ) + 1
            s ← s + ( x/i)**ν

        return s
    return  f
```

We can see that it works beautifully right away:

```
def a ( x ):
    return x **2
```

Trying it out...

```
>> G(a,1)(10)
1220703125.0
```

But, more impressive, is the expansion of gates that it accomplishes, when combined with an algorithm to "unroll" the loops of a (quantum-) Turing machine, whose definition has been modified to specify the action on the tape during a transition to correspond to the application of combinations of quantum logic gates. (Understandably, this machine is far more powerful than a quantum Turing machine.)

```
def expand ( st, M_s, M_a, T, ip_size, N  ):
    v ← []
    v.append (0)
    print ( st [0])

    for i in range ( m.floor ( G ( T, N )( ip_size )) ):
        res ← []
```

```
    res2 ← []
    for s in v:
        res.append ( M_s [ s ][0] )
        res.append ( M_s [ s ][1] )
        res2.append ( st [ M_s [ s ][0] ] )
        res2.append ( st [ M_s [ s ][1] ] )
    print (res2 )
    v ← res
```

We apply it for a three state Q-TM, with a 3-qubit input, and ask that the polynomial approximation in Hardy's scheme contains only three terms. We have assumed that it's run terminates in linear time. Even then, we get the branching out as below.
We cannot print even a quarter the tree over here, due to its size:

```
>>expand(['h','x','s'],[ [2,1],[0,2],[1,1] ],[ [1,-1],[1,1],[-1,1] ],a,3,4)


h
['s', 'x']
['x', 'x', 'h', 's']
['h', 's', 'h', 's', 's', 'x', 'x', 'x']
['s', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's', 'h', 's', 'h', 's']
['x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'h',
's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x']
['h', 's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h',
's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x',
'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h',
's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's']
['s', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x',
'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 's', 'x',
'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h',
's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's',
'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'h', 's', 's', 'x', 'x',
'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x',
's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x', 's',
'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x',
'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x',
'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x',
's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x', 's',
['x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'h',
's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's',
'h', 's', ', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h',
's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's',
'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'h', 's', 'h', 's', 's',
'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x',
'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x',
'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x',
's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'h', 's', 'h', 's', 's', 'x', 'x', 'x', 's',
```

```
'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x',
'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h',
's', 'h', 's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's',
'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h', 's', 'h', 's', 'x', 'x', 'h', 's', 'h',
's', 'h', 's', 's', 'x', 'x', 'x', 's', 'x', 'x', 'x', 'x', 'x', 'h', 's', 'h',
```

...and so on.
Clearly, for linear-time termination, we could have as easily made the branching happen in quadratic time, and stopped there- just another indication that a generalised algorithm is, by necessity inefficient; thus making a great case for the assertion that analysis of algorithms, both quantum and classical, remains a human being's job.