Classical computation on a quantum computer
(Nielsen and Chuang 3.2.5)

Last week we considered the order finding algorithm, which requires controlled U's for

$$U = \sum_{j=0}^{2^L-1} |xj \bmod N\rangle\langle j|$$

This unitary is just doing modular multiplication, which is efficient on a classical computer

We've seen before that a quantum computer can efficiently simulate a quantum computer

quantum computer $\longrightarrow$ reversible classical computer $\longrightarrow$ general classical computer

So can we just use classical methods (on a quantum computer) to perform these operations?

More generally, can we run classical subroutines within our quantum computations?

Yes! But it is not completely straightforward

Consider only reversible classical circuits
(since unitary circuits are of this form)

Consider a circuit that maps an input z to an output f(z),
and also ouputs z to ensure reversibility

In general it will also output 'garbage' g(z)

$$(z, 0) \longrightarrow \big(z,\ f(z),\ g(z)\big)$$

If the input was not included in the output, the garbage
might be required for unitarity. But this is not the case here

So g(z) is just some undeleted remnants of the computation

For a classical computation, this garbage can be ignored or deleted

For a quantum computation, we have superpositions to worry about

$$U|z,0,0\rangle = |z, f(z), 0\rangle \quad \Rightarrow \quad U\sum_z c_z |z,0,0\rangle = \sum_z c_z |z, f(z), 0\rangle$$

$$U'|z,0,0\rangle = |z, f(z), g(z)\rangle \quad \Rightarrow \quad U'\sum_z c_z |z,0,0\rangle = \sum_z c_z |z, f(z), g(z)\rangle$$

The operations U and U' are fundamentally different. U' entangles the computation to garbage in an ancilla, which could mess up required interference effects

$$H_1 U^\dagger U H_1 |0,0,0\rangle = |0,0,0\rangle$$

$$H_1 U^\dagger U' H_1 |0,0,0\rangle = \frac{1}{\sqrt{2}}\left(|0,0,0\rangle + |1,0,1\rangle\right) \quad \text{if } g(z) = z$$

So, can a function f(z) that can be computed efficiently with a classical computer also be computed efficiently with a reversible classical computer that produces no garbage?

Yes! By means of 'uncomputation'

$(z, 0, 0, 0)$ We use 4 registers, one with input, rest set to 0

$\rightarrow (z, f(z), g(z), 0)$ Then do the computation, including garbage output

Then copy f(x) to the fourth register $\rightarrow (z, f(z), g(z), f(z))$

$\rightarrow (z, 0, 0, f(z))$

Finally invert the computation (but not the copy) to reset the second and third registers

Same complexity, but no garbage

Note: we assumed above that the additional registers were initialized in the zero state

$$(z, 0, 0, 0) \longrightarrow (z, 0, 0, f(z))$$

This need not be true in general, they can be initialized in any state

$$(z, a, b, c) \longrightarrow (z, 0, 0, f(z))$$

This allows us to have nontrivial a, b and c if we want (pre-stored constants, mathematical convenience)

Coming back to the quantum realm (and ignoring any systems that start and finish in the same state) an efficient classical computation

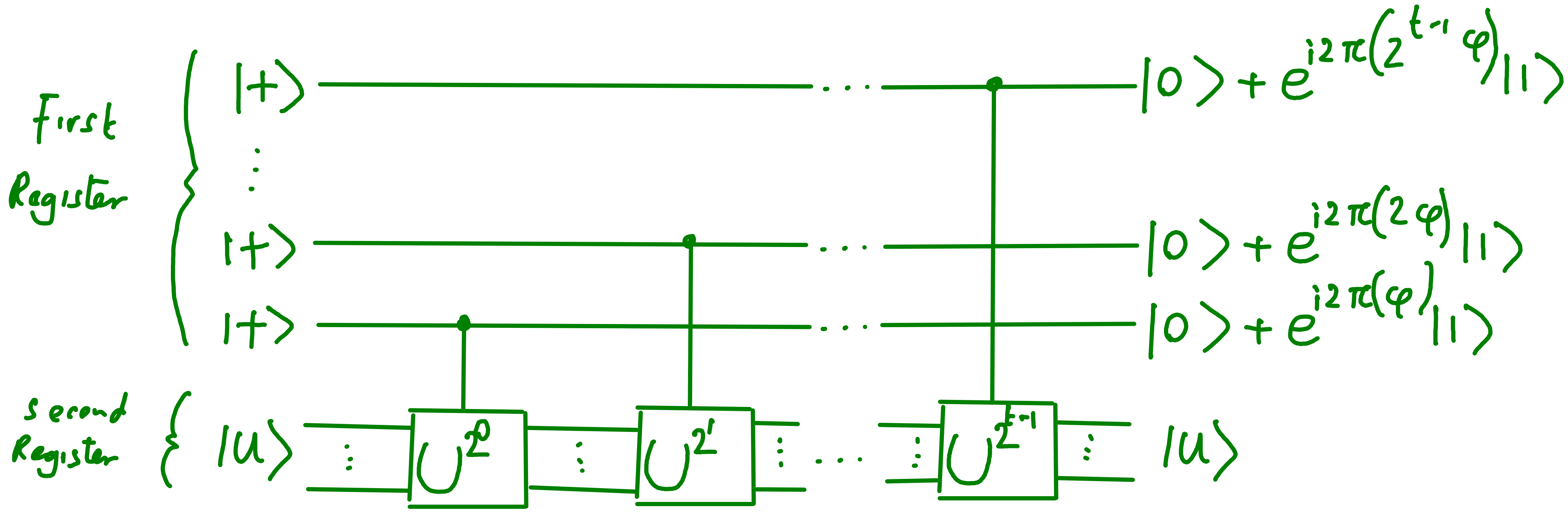
$$z \rightarrow (f(z), g(z)) \quad [\text{garbage present in general}]$$

Implies an efficient quantum computation

$$|z, 0\rangle \rightarrow |z, f(z)\rangle \quad [\text{no garbage}]$$

Modular Exponentiation

The method we considered last week requires us to implement the circuit



For the unitary

$$U = \sum_{j=0}^{2^L-1} |xj \bmod N\rangle\langle j|$$

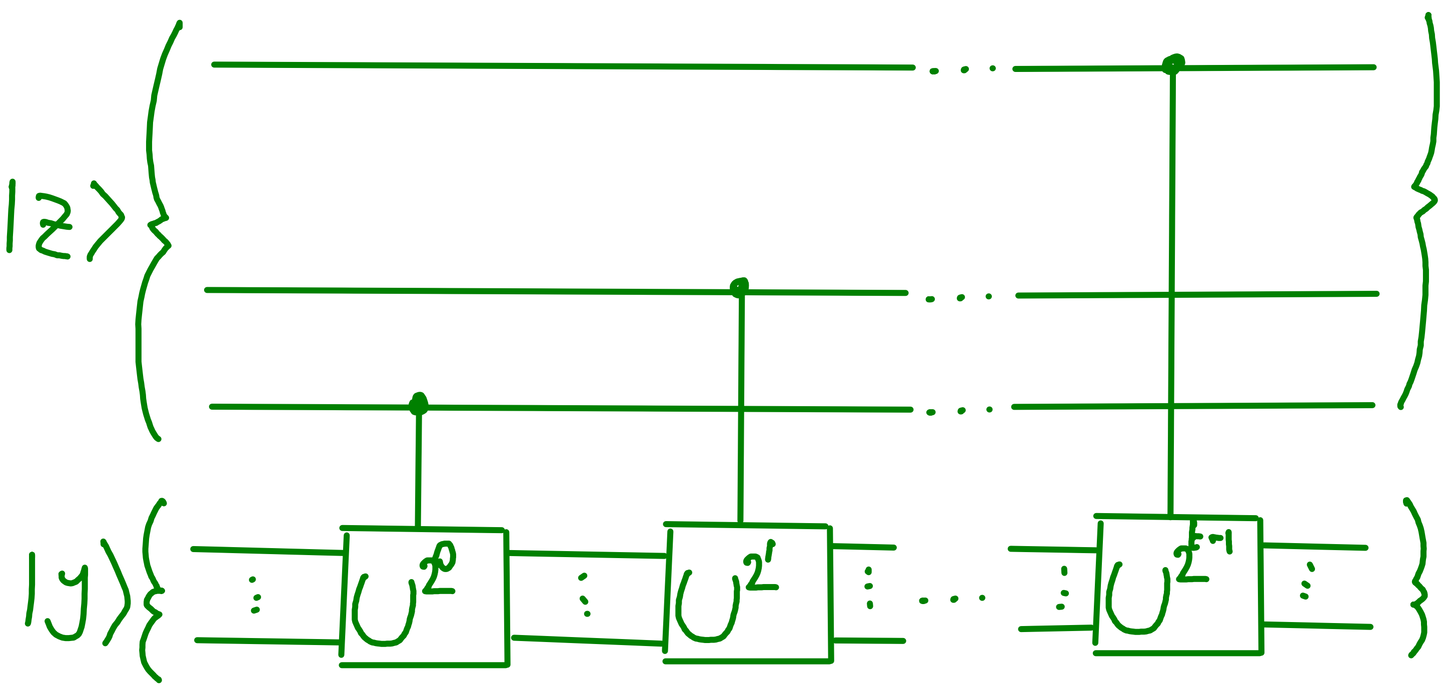
Clearly this is of size $2^L \times 2^L$ and the second register holds L qubits

How do we perform this efficiently?

The state of the first register can be expressed

$$|+\rangle^{\otimes t} = \sum_{z} |z\rangle \qquad |z\rangle = |z_{t-1}, \dots, z\rangle, \quad z = \sum_{j=0}^{t-1} z_j 2^j$$

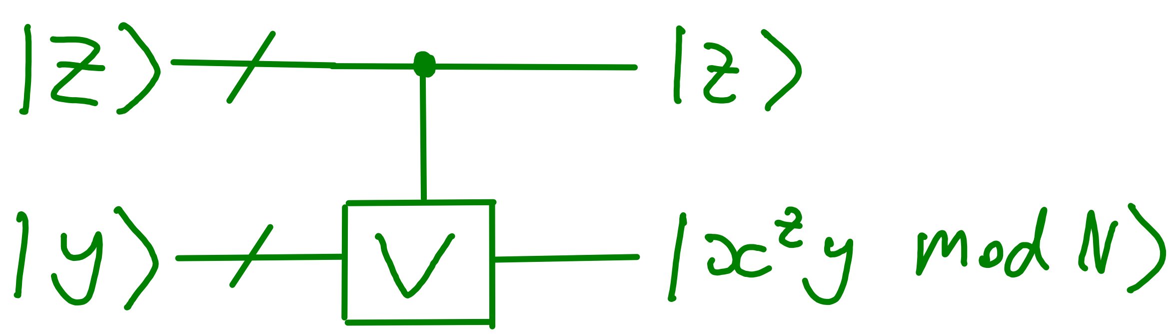
Let's consider application of the circuit for a given z on the first register, and a basis state y for the second

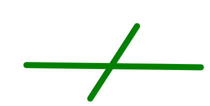


$$|x^{z_{t-1}2^{t-1}} \times \dots \times x^{z_0 2^0} \times y \mod N\rangle$$
$$= |x^z y \mod N\rangle$$

Nice thing about this basis: state on first register does not change

So the circuit can be simply understood as



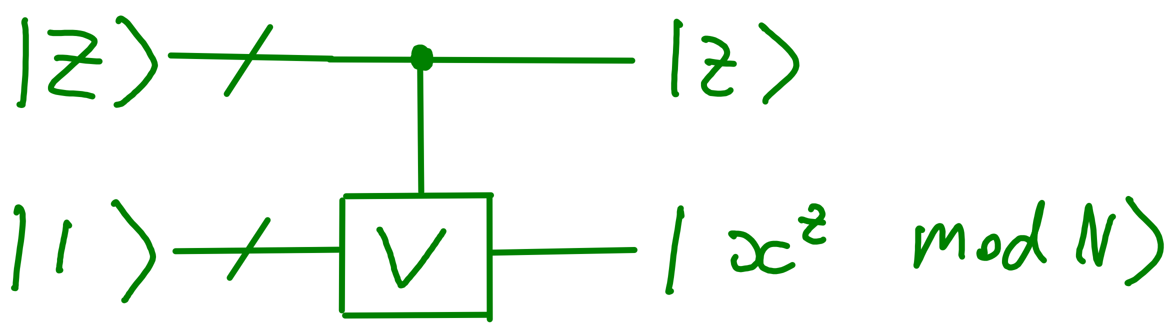
 = multiple qubits

Also remember from last week that we cannot prepare the eigenstates $|u_s\rangle$ needed for the input of the 2nd register

However, it is fine to use the superposition state

$$|1\rangle = \sum_{s=0}^{r-1} |u_s\rangle \qquad \left(|1\rangle = |00\ldots01\rangle \right)$$

So the circuit we need to implement is



We only need to run the circuit for y=1, not abitrary values of y. This will make life easier, since we don't need to fully implement the unitary

$$U = \sum_{j=0}^{2^L-1} |x^j \bmod N\rangle\langle j|$$

But instead we can implement any unitary such that

$$|z\rangle \otimes |1\rangle \rightarrow |z\rangle \otimes |f(z)\rangle, \qquad f(z) = x^z \bmod N$$

$$|z\rangle \otimes |1\rangle \to |z\rangle \otimes |f(z)\rangle, \qquad f(z) = x^z \bmod N$$

This can be done efficiently on a classical computer, and so also on a quantum one without garbage

Total complexity upper bounded by $O(L^3)$

Note: The full U can also be efficiently implemented

However, this is more complex, so we just consider the simpler case here

Factoring

We now have an efficient algorithm for calculating r, the smallest integer such that

$$x^r = 1 \mod N, \quad x < N$$

This might make number theorists happy, but we want to rob banks and spy on people by breaking RSA. We want to factor!

To find all prime factors it is sufficient to have an algorithm that can find a non-trivial factor for a given number N

Repeated calls to this can then be used to find all prime factors

Primality tests can be used to determine whether outputs are prime, these are classically efficient

There are two theorems that help us to build a factor finding algorithm

Theorem 1: If N is an L bit number and y satisfies

$y^2 = 1 \mod N, \quad 1 < y < N-1$ (trivial solutions $y=1$ and $y=N-1$ are excluded)

Then $\gcd(y-1, N)$ and/or $\gcd(y+1, N)$ is a non-trivial factor of N computable with complexity $O(L^3)$

Proof:

$y^2 = 1 \mod N \;\therefore\; y^2 - 1 = 0 \mod N \;\therefore\; N$ is a factor of y^2-1

$y^2 - 1 = (y+1)(y-1)$ $\therefore$ factors of N must be split between the two

$\therefore N$ must have a common factor with at least one

but $1 < y < N-1$ $\therefore$ the factor cannot be N itself

So computing $\gcd(y-1, N)$ and $\gcd(y+1, N)$ will yield at least one non-trivial factor of N

The 'Euclidean algorithm' can compute the gcd in poly L time on a classical computer

From this theorem, it is clear that an efficient means to calculate y could be used to efficiently factor numbers

To find a factor of N, just find the corresponding y

$$y^2 = 1 \mod N$$

Then compute $\gcd(y-1, N)$ and $\gcd(y+1, N)$

This gives at least one nontrivial factor, N', of N

Then calculate $N'' = N/N'$ and apply the same method again for both N' and N''

Continue to break down each factor into further factors until only primes remain

This will yield all primes in time

$$O\left(\#\text{ prime factors} \times \left[\text{poly}(L) + \text{time required for } y\right]\right)$$

Classical computation of y is not efficient. What about quantum?

Theorem 2: For an odd positive integer N expressed in terms of m distinct prime factors as

$$N = \prod_{j=1}^{m} p_j^{\alpha_j}$$

And for a randomly chosen integer x that satisfies

$$2 \leq x \leq N-1 \qquad \gcd(x, N) = 1$$

Consider r, the order of x mod N $\quad x^r = 1 \mod N$

$$\text{Prob}\left(r \text{ is even and } x^{r/2} \neq -1 \bmod N\right) \geq 1 - \frac{1}{2^m}$$

Proof: Nielsen and Chuang A4.3

If r is even then $x^{r/2}$ is an integer such that $\left(x^{r/2}\right)^2 = 1 \mod N$

And since $x^{r/2} \neq -1 \mod N$, then $1 < x^{r/2} \mod N < N-1$

So, with O(1) probability, $y = x^{r/2} \mod N$

Shor's Algoritm

Using order finding, a quantum computer can efficiently find y, and so efficiently factor

The whole algorithm to find a factor for an L bit number N is then as follows

Grey steps use a classical computer, black use a quantum computer

Step 1: Determine whether N is prime or composite

Can be done efficiently with AKS primality test

If prime, output N;
Else, continue

Step 2: Determine whether N is even

If even, output 2;
Else, continue

Step 3: Determine whether N is of the form $N = a^b$, $a \geq 1$, $b \geq 2$ where a and b are integers

Can be done efficiently (N+C exercise 5.17)

If so, output a;
Else, continue

Step 4: Randomly choose x in the range 2 to N-1 and calculate $\gcd(x, N)$

Sampling can be done in O(L) time and gcd is efficient using the Euclidean algorithm

If $\gcd(x, N) > 1$ output this;
Else, continue

Step 5: Take the x from step 3 for which $\gcd(x, N) = 1$ and find r, the order of x mod N. This can be done efficiently using quantum order finding

If r is odd or $x^{r/2} \bmod N = -1$ output 1 (fail);
Else, set $y = x^{r/2} \bmod N$

Step 6: Calculate $\gcd(y-1, N)$ and $\gcd(y+1, N)$

Efficient using Euclidean algorithm

Output $\gcd(y-1, N)$ and $\gcd(y+1, N)$

This algorithm succeeds (outputs only trivial factor 1) only when

1) The number does not require step 4 or later

2) If the number requires up to step 4, step 5 outputs an r such that

$$r \text{ is even and } x^{r/2} \not\equiv -1 \bmod N$$

and theorem 2 guarantees that

$$\mathrm{Prob}\left(r \text{ is even and } x^{r/2} \not\equiv -1 \bmod N\right) \geq 1 - \frac{1}{2^m} \geq \frac{1}{2} = O(1)$$

where m>2 is the number of distinct prime factors

So it succeeds with O(1) probability

Factorizing 15

Step 1: Not prime, so continue
Step 2: Not even, so continue
Step 3: Not of this form, so continue

Step 4: Let's consider the case that x=4 is the randomly chosen value. gcd(4,15)=1, so continue

Step 5:

$4^2 = 16 = 15+1 = 1 \bmod 15$ $\quad \therefore\ r = 2$

$\therefore\ y = x^{r/2} = 4$

Step 6:

$y+1 = 5$ $\quad gcd(5,15) = 5$

$y-1 = 3$ $\quad gcd(3,15) = 3$

Algorithm outputs both factors