Implementation of Shor's factorization algorithm

Quantum computer is different from classical computer. There are several quantum algorithms which solve some problems better than those algorithms in classical computer. Factoring is one of the problems proposed by Peter Shor who discovered the quantum algorithm to solve it. This project tried to simulate a quantum computer to implement Shor's factorization algorithm using two different open-source software development kits: IBM Qiskit and Microsoft Q#.

Project No.4
Member: Tanawat Deepo 59070503423
Advisor: Dr. Jaturon Harnsomburana

21 → Quantum algorithm → 3 x 7

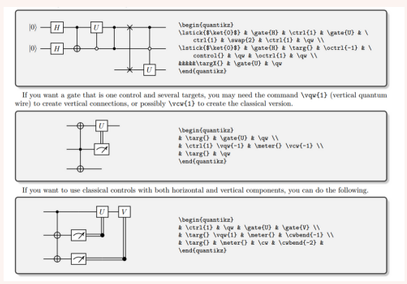
Objectives

- To simulate a quantum circuit of Shor's factorization algorithm.
- To factor composite numbers (represented in no more than 6 bits) of two small prime numbers.
- To break simple encryption involving prime factorization.
- To break RSA encryption.

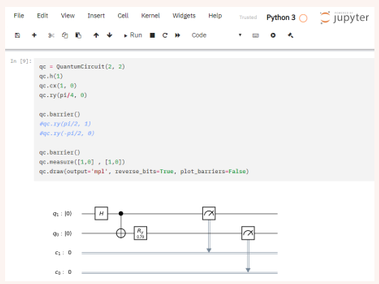
Scope

- Use IBM Qiskit and Microsoft Q# for simulation.
- The number to be factored is a composite number (represented in no more than 6 bits) of two small prime numbers.

Software tools



Quantum circuit design: LaTeX



Qiskit Notebooks (IBM)

```
>dotnet run
Hello quantum world!
```

```
1  import qsharp
2  from Chapter3 import HelloQ
3
4  HelloQ.simulate()
```

```
1   namespace Chapter3 {
2
3       open Microsoft.Quantum.Canon;
4       open Microsoft.Quantum.Intrinsic;
5
6
7       operation HelloQ() : Unit {
8           Message("Hello quantum world!");
9       }
10  }
```

Microsoft Q# and .NET SDKs

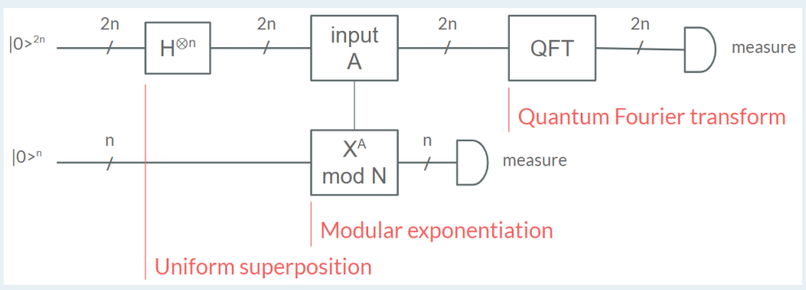
Order finding: a clever method to factor a number N

To factor $N = p*q$
Choose $X < N$
Find r that $X^r \equiv 1 \bmod N$
$X^r - 1 \equiv 0 \bmod N$
$(X^{r/2} - 1)(X^{r/2} + 1) \equiv 0 \bmod N$ (LHS is divisible by N)
$p = \gcd(X^{r/2} - 1, N)$ or $\gcd(X^{r/2} + 1, N)$
$q = N/p$

The method has polynomial complexity

$$O((\log N)^2 (\log \log N) (\log \log \log N))$$

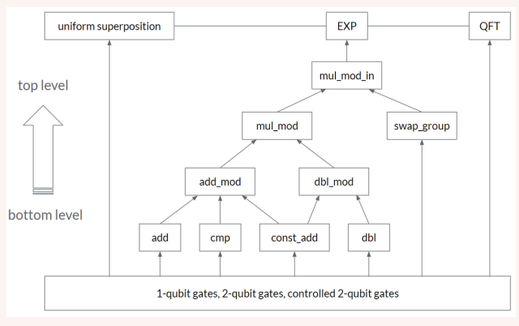
N is represented in n bits



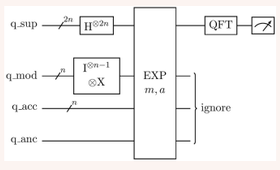
This project simulated quantum circuits and compared results between IBM Qiskit and Microsoft Q#.

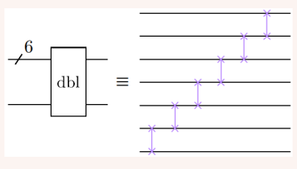
Circuit design: IBM Qiskit

Qiskit



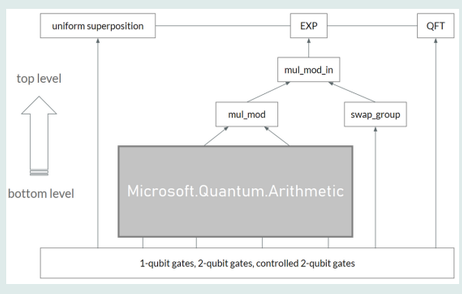
Implement everything from elementary gates to the top level.



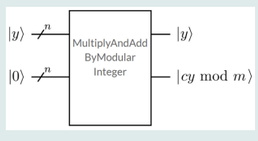


Circuit design: Microsoft Q#

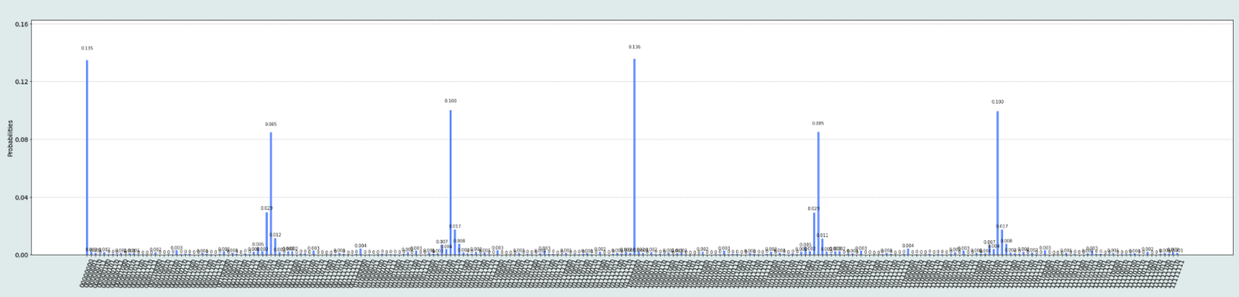
<Q#>



There are some built-in operations provided in Microsoft.Quantum library.



Results



Histogram of measurement results. 6 peaks mean the order is 6.

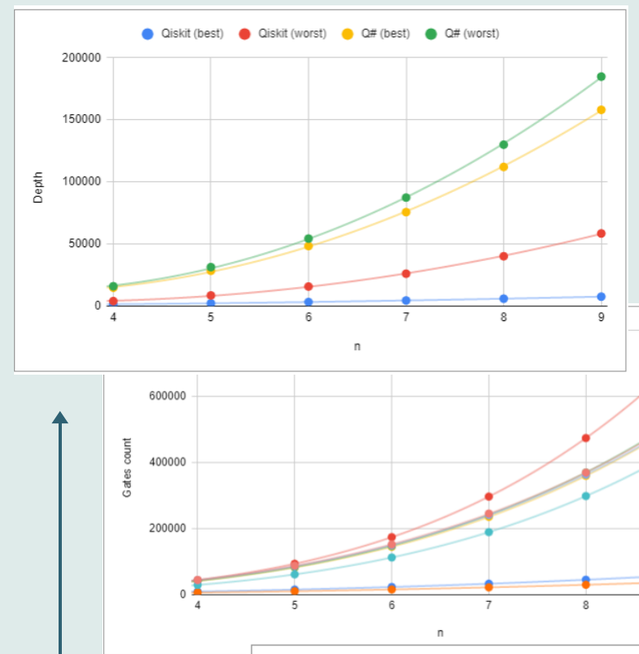
IBM Qiskit vs Microsoft Q#

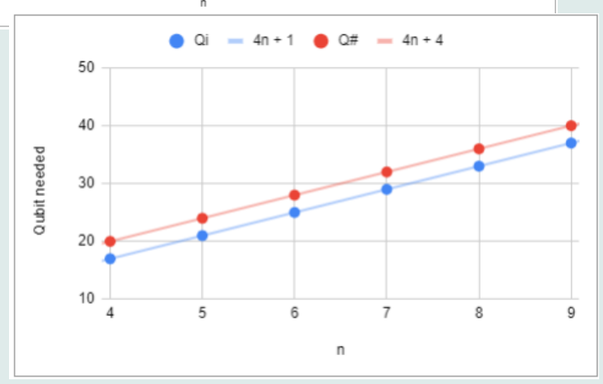
```
Round 1

----------------------------
choose X = 2

measured value: 320 (of 1024 basis states)
continued fraction: [0, 3, 5]
guessed k/r: 5 / 16
----------------------------
answer: 21 = 3 x 7
{
  "factor": [
    3,
    7
  ],
  "log": [
    {
      "type": "QFT",
      "rand": 2,
      "measure": [
        {
          "value": 320,
          "basis": 1024,
          "frac": [
            {
              "k": 5,
              "r": 16
            }
          ]
        }
      ]
    }
  ]
}
```

Output of factoring 21 = 3 x 7





Comparison of depth
Comparison of gates count
Comparison of qubits needed

RSA encryption demonstration

```
Hello quantum!

public key:  (15, 3)
secret key:  (15, 3)

encode:      2   2   0   6   2   5   5   4   3   3   0   6
7    4   4   0   3   4   2   7   2   5   4   1   3   3
4    7   2   1   6   5   3   3   2   2   0   4

encrypt:     8   8   0   6   8   5   5   4   12  12  0   6
13   4   4   0   12  4   8   13  8   5   4   1   12  12
4    13  8   1   6   5   12  12  8   8   0   4

decrypt:     2   2   0   6   2   5   5   4   3   3   0   6
7    4   4   0   3   4   2   7   2   5   4   1   3   3
4    7   2   1   6   5   3   3   2   2   0   4

Hello quantum!
```

The encryption/decryption of the text "Hello quantum!". Assume that the attacker is the man in the middle between encryption and decryption processes. He knows the public key, and he also knows the private key because RSA encryption involves prime factorization that is a critical part of the encryption.

Conclusion

Criteria	IBM Qiskit	Microsoft Q#
maximum length (bits)	**6**	5
times to random	indeterminate	indeterminate
times to do measurement	indeterminate	indeterminate
running time	**lower**	higher
depth	**lower**	higher
width	**lower**	higher
gates count	looser	**tighter**
demonstrate RSA breaking	yes	yes

Sumarized comparison between two SDKs

There is an unsolved problem of too much running time consumed in Microsoft Q#. This project also tried to break a simple encryption involving no-more-than-5-bit composite number.

Image Sources

- IBM Qiskit logo - https://pennylane.readthedocs.io/en/user-docs-refactor/introduction/plugins.html
- Microsoft Q# icon - https://www.microsoft.com/en-us/quantum/development-kit