Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Perpustakaan Univeristas Multimedia Nusantara

BAB II

LANDASAN TEORI

2.1 Algoritma Kuantum Shor's

Sebuah *bit* merupakan sebuah konsep fundamental dalam merepresentasikan informasi di bidang komputasi dan informasi klasik. Komputasi dan informasi kuantum didirikan berdasarkan konsep yang mirip dalam bentuk *qubit* atau *quantum bit. Qubit* digunakan sebagai unit bilangan terkecil dalam komputasi kuantum dengan menggunakan *reversible gate* untuk memproses *qubit* dan menghasilkan keluaran yang diinginkan. Sebuah *qubit* memiliki kemungkinan untuk memiliki dua nilai yaitu 0 dan 1, seperti halnya bilangan biner. *Qubit* dapat diukur (*measure*) yang akan membuat *qubit* tersbut *collapse* dan bernilai 0 atau 1 sesuai dengan *state qubit* tersebut. Setiap nilai dari *qubit* memiliki suatu nilai koefisien yang merupakan suatu nilai probabilitas yang menjadi peluang *qubit* tersebut bernilai 0 atau 1 di saat dilakukan pengukuran (Nielsen dan Chuang, 2002).

Terdapat beberapa konsep yang digunakan dalam bidang mekanika kuantum yang dibawa ke dalam bidang komputasi kuantum, salah satunya yaitu *Quantum Superposition. Quantum Superposition* adalah suatu keadaan di suatu waktu di mana suatu *qubit* memiliki probabilitas untuk berada di *state* 0 dan 1 sehingga *qubit* tersebut berada di *state* 0 dan *state* 1 secara bersamaan (Lavor dkk., 2003). Sebuah algoritma kuantum biasanya mengandalkan sebuah sifat dasar dari konsep komputasi kuantum yaitu *Quantum Parallelism. Quantum Parallelism* datang dari kemampuan sebuah *Quantum Register* untuk berada dalam sebuah *superposition*. Sebuah *quantum register* dapat berada dalam *superposition* untuk

setiap *state*, setiap elemen dari *superposition* ini dapat merepresentasikan sebuah bagian pada suatu fungsi. Jika sebuah fungsi di terapkan kepada sebuah *register* yang berada dalam *superposition* maka fungsi tersebut akan diterapkan kepada setiap elemen dari *superposition* tersebut. Karena jumlah *state* yang dapat direpresentasikan oleh *qubit* adalah 2^n di mana *n* adalah jumlah *qubit*, maka suatu operasi komputasi kuantum dapat dijalankan dalam satu operasi di mana kalau dijalankan pada komputer klasik akan memakan jumlah operasi yang eksponensial jika berbanding dengan komputasi kuantum (Hayward, 2005).

Pada tahun 1994, Peter W Shor menemukan sebuah algoritma kuantum yang dapat melakukan pemfaktoran bilangan prima dalam waktu polynomial (Shor, 1994). Algoritma kuantum Shor memberikan solusi kepada permasalahan *Prime Factorization Problem* yang sebelumnya tidak dapat dilakukan dengan waktu *Polynomial* (Lomonaco, 2000). Algoritma Shor dilihat sebagai sebuah cara yang efisien dalam melakukan pemfaktoran angka yang besar yang mendasari banyak dari sistem kriptografi yang umumnya dipakai. Algoritma Shor mempunyai kompleksitas O($(log\ n)^2 * log\ log\ n$) jika dijalankan pada komputer kuantum yang merupakan *polynomial-time*. Hal ini menarik banyak perhatian untuk mengembangkan bidang komputasi kuantum (Hayward, 2015). Proses algoritma Shor untuk pemfaktoran sebuah bilangan N dapat dibagi menjadi beberapa langkah singkat berikut (Beauregard, 2003).

1. Tentukan apakah bilangan N bersifat genap, jika iya kembalikan faktor dari bilangan N adalah 2 dan $N/2$ (Beauregard, 2003).

2. Tentukan apakah bilangan N memenuhi persamaan $N = p^q$ di mana $p \geq 1$ dan $q \geq 2$, jika iya kembalikan faktor p (Beauregard, 2003).
3. Tentukan bilangan a secara acak sesuai dengan persamaan $1 < a \leq N - 1$ dan $\text{FPB}(a, N) = 1$ (Beauregard, 2003).
4. Menggunakan algoritma *order-finding* untuk mencari periode r dari persamaan $a^x\ Modulo\ N$ (Beauregard, 2003).
5. Jika r merupakan bilangan ganjil atau merupakan bilangan genap yang memenuhi persamaan $a^{r/2} = -1\ (mod\ N)$, kembali ke langkah 3. Jika tidak lakukan perhitungan $FPB\left(a^{\frac{r}{2}} - 1, N\right)\ dan\ FPB\left(a^{\frac{r}{2}} + 1, N\right)$. Kemudian jika hasil dari perhitungan tersebut menghasilkan faktor dari N kembalikan faktor tersebut (Beauregard, 2003).

Bagian proses yang menggunakan prinsip mekanika kuantum pada algoritma Shor terletak pada langkah ke-4 algoritma Shor yang telah dijelaskan, bagian ini adalah bagian pencarian periode r dari fungsi periodik $a^x\ Modulo\ N$. Bagian *Order-finding* ini terdiri dari fungsi *modular multiplication* yang akan menjabarkan hasil dari fungsi periodik $a^x\ Modulo\ N$ menjadi *superposition state* dan fungsi *inverse quantum fourier transform* untuk mendapatkan nilai r atau periode dari fungsi periodik setalah terjabar dalam *superposition state* (Qiskit Documentation, 2016), (Beauregard, 2003).

2.2 Sistem Kriptografi RSA

Sistem kriptografi RSA adalah sebuah *Public Key Algorithm* yang terkenal dibuat oleh Ron Rivest, Adi shamir, dan Len Adleman dan pertama kali dipublikasikan pada tahun 1977. RSA biasanya dipakai untuk menjaga privasi dan autentikasi data digital. RSA sendiri telah menjadi standar dalam komunikasi digital, RSA menjaga keamanan data dan membuat data tersebut tidak diakses oleh suatu entitas yang tidak terautentikasi (Kalpana, 2012).

RSA adalah algoritma kriptografi pertama yang cocok untuk *signing* dan enkripsi (Dhakar dkk., 2012). Algoritma kriptografi RSA teridir dari tiga bagian besar yaitu pembuatan *private* dan *public key*, enkripsi, dan dekripsi. Berikut adalah tahapan pembuatan *public-key* dan *private-key* RSA dan fungsi enkripsi dan dekripsi (Chhabra, 2011).

- Tentukan dua bilangan prima, p dan q. dan menghitung $n=pq$.
- Tentukan bilangan $\Phi(N) = (p-1)(q-1)$
- Tentukan bilangan e secara acak dan bersifat koprima $\Phi(N)$ dengan yang akan menjadi *public exponent* untuk *Public Key*.
- Menghitung d dari hasil bagi $d = e^{-1}\ Mod\ \Phi(N)$. Bilangan d menjadi *private exponent*.
- *Public Key* adalah pasangan bilangan (n,e).
- *Private Key* adalah pasangan bilangan (n,d)
- Mengenkripsi suatu *plain text* M dengan rumus $C = M^e Mod\ n$
- Mendekripsi suatu *cipher text* C dengan rumus $M = C^d Mod\ n$

2.3 Qiskit IBM Q Experience

Qiskit adalah sebuah *open-source framework* untuk *quantum computing*. Qiskit menyediakan berbagai *tools* untuk membuat dan memanipulasi *quantum programs* dan menjalankannya pada *prototype quantum device* yang dimiliki *IBM Q Experience* atau dari simulator pada komputer lokal. Qiskit didirikan oleh *IBM Research* untuk membuat sebuah *platform* untuk mengembangkan *cloud computing service IBM Q Experience* (Qiskit Documentation, 2016).

Qiskit terdiri dari empat bagian besar yaitu *Qiskit Terra, Qiskit Aer, Qiskit Ignis, dan Qiskit Aqua*. *Qiskit Terra* adalah model dasar pengembangan program kuantum yang terdiri dari enam modul yaitu *Circuit, Pulse, Transpiler, Providers, Quantum Information, Visualization*. *Qiskit Aer* mencakup tiga *high performance simulator backends* yaitu *Qasm Simulator, Statevector Simulator, Unitary. Qiskit Ignis* menyediakan *code* untuk membantu *user* membuat sirkuit kuantum untuk eksperimen yang spesifik. *Qiskit Ignis* mencakup tiga dasar yaitu *Circuits, Fitters, dan Filters. Qiskit Aqua* adalah di mana *Quantum Algorithm* dirancang dan dibuat, dan biasanya dipakai oleh orang yang berpengetahuan dalam tentang *quantum computing* (Qiskit Documentation, 2016).

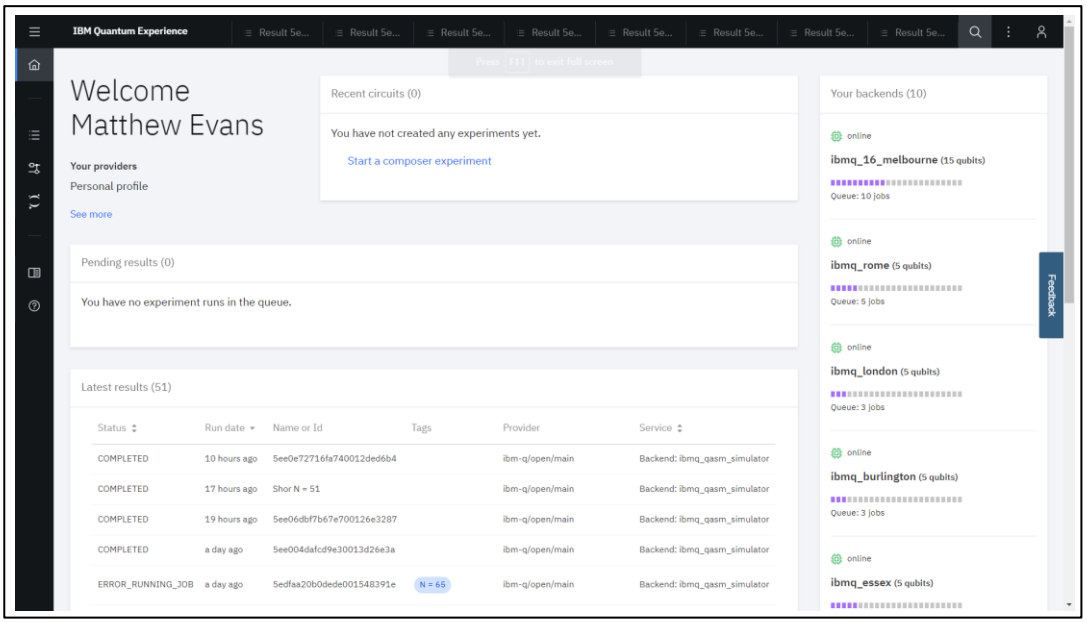
IBM Q Experience menyediakan sebuah *website* yang menyediakan beberapa fitur yang dapat membantu penelitian kuantum menggunakan Qiskit. Pada *website* ini terdapat daftar komputer kuantum dan simulator kuantum yang dapat dipakai untuk umum dalam pelaksanaan penelitian kuantum. Setiap kali menjalankan suatu sirkuit kuantum terhadap suatu *backend* komputer kuantum atau simulator kuantum akan terbuat sebuah *Job* yang akan masuk ke dalam *queue* yang dimiliki oleh setiap *backend* kuantum komputer atau kuantum simulator. Pada

halaman *web* ini juga pengguna dapat mengatur dan memonitor semua daftar *job* yang telah dilaksanakan (Qiskit Documentation, 2016).

Qiskit menyediakan Implementasi sirkuit kuantum berdasarkan algoritma Shor yang bersifat *open source* dan dapat digunakan untuk mempelajari dan mengembangkan sirkuit kuantum tersebut. Qiskit menyediakan beberapa fungsi yang dapat digunakan untuk pembuatan sirkuit kuantum Shor (Qiskit Documentation, 2016).

1. Fungsi *check_if_power(),* mengembalikan hasil pengecekan *perfect power* (Qiskit Documentation, 2016).
2. Fungsi *get_value_a(),* mengembalikan variable *a* (Qiskit Documentation, 2016).
3. Fungsi *get_factors(),* mengembalikan hasil dari proses pencarian faktor menggunakan *continued fractions* (Qiskit Documentation, 2016).
4. Fungsi *mod_inv(),* mengembalikan hasil *modular inverse* (Qiskit Documentation, 2016).
5. Fungsi *create_inverse_QFT(),* pembuatan sirkuit kuantum *inverse QFT* (Qiskit Documentation, 2016).
6. Fungsi *cMULTmodN(),* pembuatan sirkuit untuk *modular multiplication* (Qiskit Documentation, 2016).

Backend komputer atau simulator kuantum akan menjalankan sirkuit sesuai dengan urutan pada *queue.* Setelah *job* selesai dijalankan, hasilnya dapat dilihat pada bagian *results*, yang akan menampilkan semua hasil *job* milik seorang *user*. Pada halaman utama dari *website IBM Q Experience* terdapat daftar *job* yang sedang berjalan, daftar *job* yang sudah selesai, dan daftar *backend* yang dapat digunakan untuk menjalankan sirkuit kuantum (Qiskit Documentation, 2016).



Gambar 2.1 Website *IBM Q Experience* halaman *results* (*IBM Q Experience Website*, 2020)