TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 3, June 2020, pp. 1319~1330
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i3.14755

Web-app realization of Shor's quantum factoring algorithm and Grover's quantum search algorithm

Arya Wicaksana, Anthony, Adjie Wahyu Wicaksono
Department of Informatics, Universitas Multimedia Nusantara, Indonesia

Article Info

Article history:

Received Sep 7, 2019
Revised Jan 21, 2020
Accepted Feb 25, 2020

Keywords:

Grover
Quantum application
Shor
Web-app

ABSTRACT

Quantum algorithms are well-known for their quadratic if not exponential speedup over their classical counterparts. The two widely-known quantum algorithms are Shor's quantum factoring algorithm and Grover's quantum search algorithm. Shor's quantum factoring algorithm could perform integer factorization in O(logN). Grover's quantum search algorithm could solve the unsorted search problem in O(√N). However, both algorithms are introduced as theoretical concepts in the original papers due to the limitations of quantum technology at that time. In this paper, an improved way is presented to realize the two algorithms into a web application using state-of-the-art quantum technology. The web-app is designed and built considering the uses of a quantum simulator and libraries provided by ProjectQ and Rigetti Forest. The result shows that both algorithms are realizable into web-applications.

Corresponding Author:

Arya Wicaksana,
Department of Informatics,
Universitas Multimedia Nusantara,
Scientia Boulevard St., Gading Serpong, Tangerang-15810, Banten, Indonesia.
Email: arya.wicaksana@umn.ac.id

1. INTRODUCTION

Quantum computing could drive the progress of breakthroughs in science by leveraging quantum mechanical phenomena to employ information [1-5]. In 1994, MIT's Peter Shor shows that it's possible to factor a number into its primes on a quantum computer in polynomial time [6-9]. This is a problem that takes classical computers "an exponentially long time" to solve for large numbers [10, 11]. In 1996, Lov Grover introduces a fast quantum mechanical algorithm for unsorted database search. The quantum search algorithm takes O(√N) for the unsorted database search problem [12] and allows quadratic speedup over its classical counterpart by using amplitude amplification in quantum computing. The two quantum algorithms are introduced as theoretical concepts in their original papers with no detailed implementation.

Today's state-of-the-art quantum computing technologies [13] are delivered by Rigetti, IBM, ETH Zurich, Microsoft, Intel, and Google. The software platforms respectively are Forest, Qiskit, ProjectQ, and Quantum Development Kit. The Bristlecone is Google's latest quantum computer with the most number of qubits to-date (72 qubits) [13] along with Sycamore (53 qubits) [14, 15]. Rigetti Forest, on the other hand, is a quantum virtual machine (QVM) that is available for public use to do quantum programming and computational simulations on a classical computer [16]. ETH Zurich's ProjectQ is a publicly accessible Python library and framework that allows quantum computing using a classical computer [17]. These two quantum virtual machine (Forest and ProjectQ) are suitable for realizing the Shor's quantum factoring algorithm and Grover's quantum search algorithm into a web application due to their quantum library support for each algorithm.

A related study on the realization of a quantum algorithm into a web application could be found in quantum computing playground (QCP) [18]. The QCP demonstrates the work of Shor's quantum factoring algorithm and Grover's quantum search algorithm. It is a web-based WebGL Chrome experiment that is created by a group of Google engineers in 2014 using QScript. However, the QCP's web-app has a limitation of using only one quantum register with size up to 22 qubits and no possible portability nor connectivity to the gate level quantum hardware. It also does not support connectivity with the state-of-the-arts quantum technologies.

Thus, this paper proposes an improved way of design and implementation for the two quantum algorithms (Shor's and Grover's) into a web application. It is to address the lack of design and implementation details up to this date for both quantum algorithms using state-of-the-arts quantum technologies [19-22]. Web platform is chosen for the realization of the quantum algorithms to allow ease of use and access. The quantum technologies that are used for the web-app realization are Rigetti Forest for Grover's quantum search algorithm and ETH Zurich ProjectQ for Shor's quantum factoring algorithm. The two quantum computing technologies are chosen due to their support and availability for general users: documentation and examples. Both Forest and ProjectQ use Python as their programming language, hence Flask micro web framework could be used for the web-app development since it is also written in Python. The performance of the web-app realization in terms of the execution time is compared with the QCP's result under the same simulation scenario and parameters.

2. RESEARCH METHOD

The research methods used are literature reviews, design and implementation, and testing and evaluation. The literature reviews include quantum bit, universal quantum gate, Shor's quantum factoring algorithm [23], Grover's quantum search algorithm, Quantum Computing Playground, ETH Zurich ProjectQ framework, and Rigetti Forest SDK. The design of the implementation of the quantum algorithm is described using a flowchart. In this research, the quantum error correction and ancilla qubits are not taken into account.

The web-app realization of the Shor's quantum factoring algorithm is done using the ETH Zurich ProjectQ framework and Flask. Meanwhile for Grover's quantum search algorithm is done using Rigetti Forest SDK with pyQuil library and Flask. Two computers with different hardware specifications are used for the implementation part. The testing is done using the white-box testing methodology and the performance of the web-app is measured by the execution time and compared with the Quantum Computing Playground under the same simulation scenario and parameters as presented in [24, 25].

2.1. Shor's implementation design

Figure 1 shows Shor's quantum factoring algorithm flowchart. First, the user is asked to input a positive integer value i.e. N. The input is evaluated and returned with true if N modulo by 2 equals 0, and returns false otherwise. The next process is the inspection process if N is prime or not. Same as the previous process, this process will return true or false depending on the value of N entered. If the value of N can be used up modulated by a number smaller than itself, this process will return a false value. Furthermore, the value of the variable q is determined by finding a rank of 2 that is greater or equal to the value of N^2. This process will add one to the power variable until a displacement of two is greater than the target. This process flowchart can be seen in Figure 2 notated by off-page reference 1. After that, the value of the variable x is determined randomly. The sub-quantum routine of Shor's quantum factoring algorithm will be run if x is coprime with N and N is not prime nor even.

On-page reference C describes the initialization process of variables which will later be used in the quantum computation process which is notated by the on-page reference D in Figure 2. The n value in this process stores the bit size needed to represent integers up to the variable q minus one. Next, the quantum register is initialized to size n, the bit size previously calculated. This quantum register will then be passed through the Hadamard gate to enter the superposition state of all numbers from zero to q minus one. A qubit will also be initialized as a ctrl_qubit variable. The measurement variables are created as n-sized arrays to store the value of modular exponentiation at a later step.

The next process is modular exponentiation. This step is repeated for n number of times. The current_x value is filled with the value from the calculation of pow (x, 1 << (n-1-k, N)) where k is the iterator of the repetition of the process. Then, the ctrl_qubit variable is brought to a superposition state before the control statement is executed. If the conditions in the Control statement are satisfied, the modular multiplication calculation is continued. The modular multiplication process executes using the quantum register a as the base, current_x as the multiplier, and N as the value that will perform the modulo operation on the previous multiplication result. The last step in the quantum computation process is the measurement of the value of ctrl_qubit. The measured value is put into the measurement array at the k-index, where k is

the iterator of this process loop. The period of N could be found by adding up all values of the array measurements and look for the best rational approximation from that value.

After the period of N, represented by the variable r is found, r's value is checked whether or not it is odd. If it is odd, the value is multiplied by 2. The two factors of N could be determined by calculating the values of GCD $((x^{(r/2)}+1), N)$ and GCD $((x^{(r/2)}-1), N)$. Both values are stored in variables f1 and f2 respectively. If the product of f1 and f2 multiples do not produce N and is more than 1, the value of f1 becomes the product of f1 by f2 and f2 becomes the result of the division of N by f1. If the factor of N found, the program will issue both values. If not, the program notifies that the calculation is failed. This failure is part of the probabilistic nature of quantum superposition. The flowchart design for this last step is shown in Figure 3.

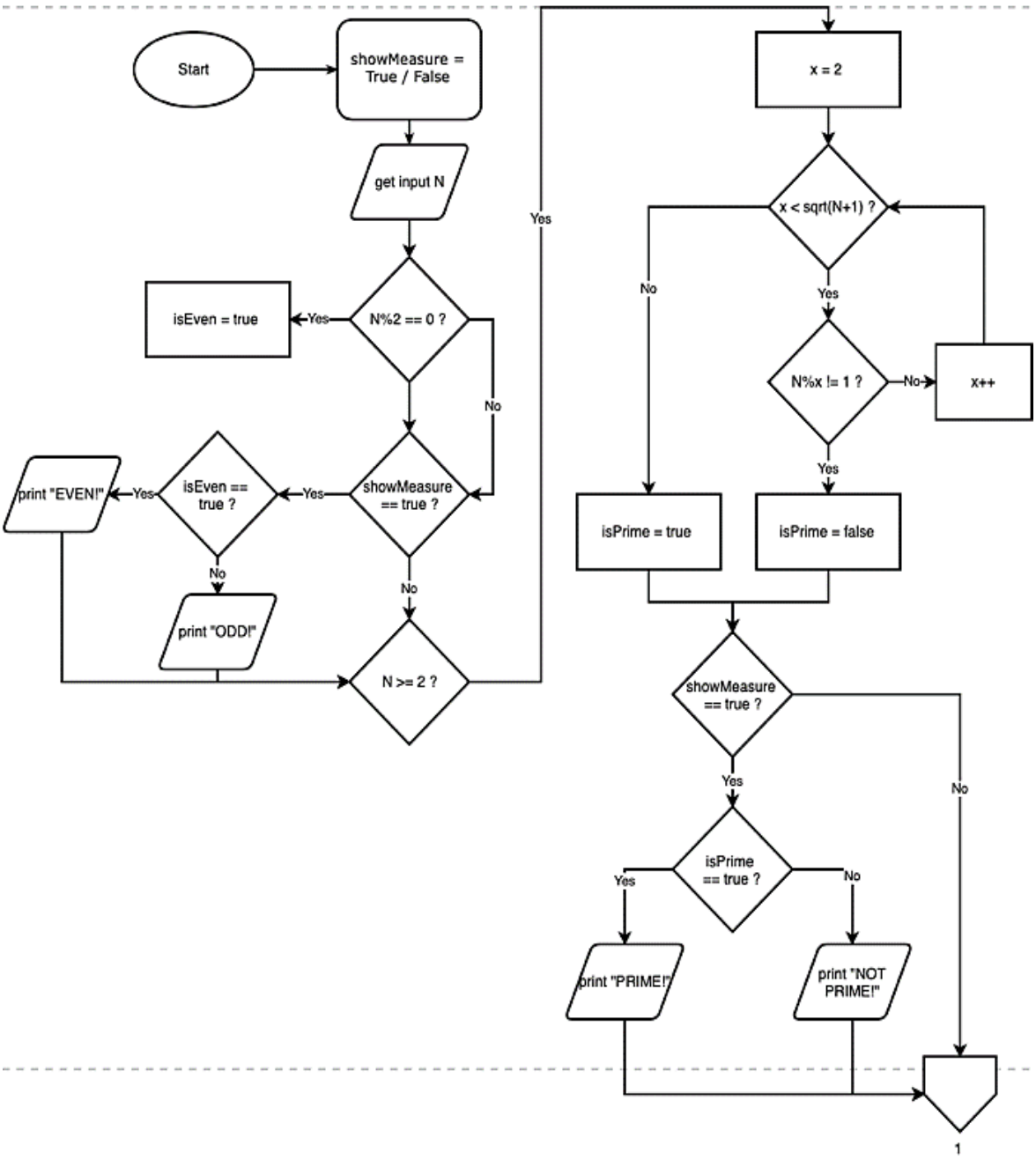

Figure 1. Shor's quantum factoring algorithm flowchart (part one)

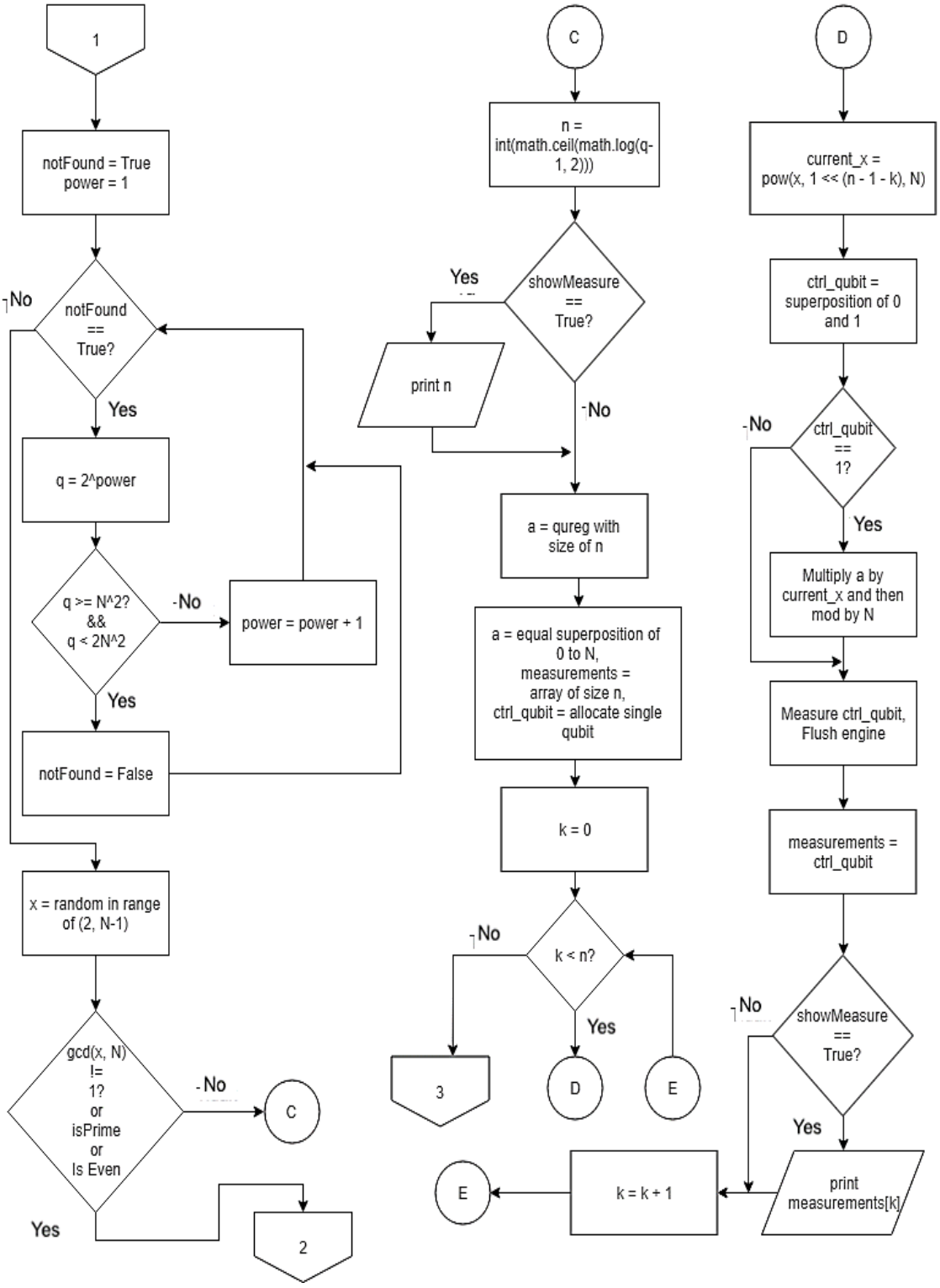

Figure 2. Shor's quantum factoring algorithm flowchart (part two)

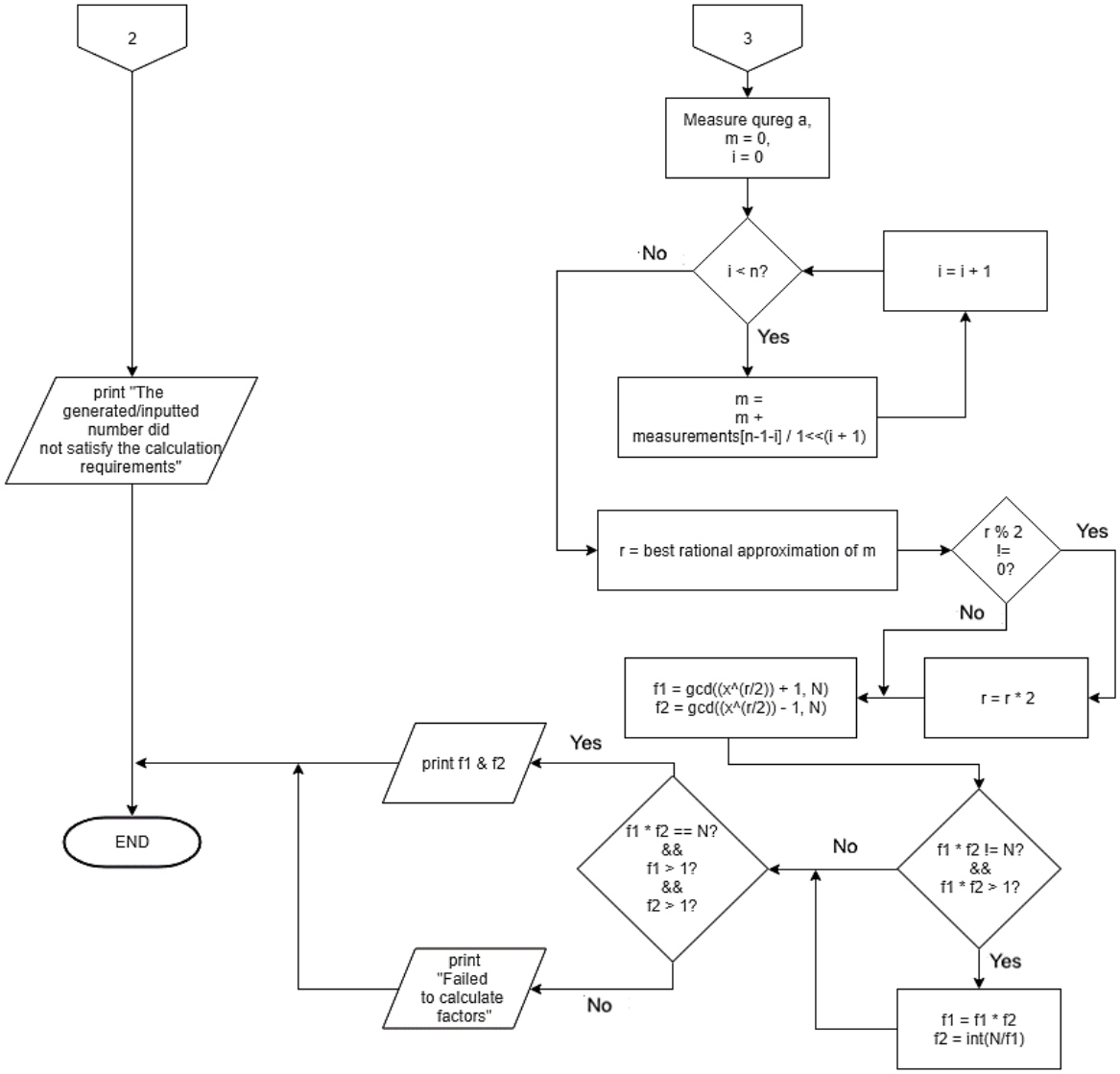

Figure 3. Shor's quantum factoring algorithm flowchart (part three)

2.2. Grover's implementation design

Figure 4 shows Grover's quantum search algorithm flowchart. First, the user enters input dataset and target to be searched. The program will convert target to binary representation and stored in variable dataTarget. Then, the program will search for max value in the dataset. The max value binary representation will be stored in variable bit string. The number of quantum bits that will be used depends on the length of the bit string. Quantum program is needed in Rigetti Forest SDK. The Oracle function and Diffusion matrix are created and added to the quantum program as a new gate. The first step in Grover's quantum search algorithm is to apply Hadamard transform on every qubit to make every state have the same amplitude. The next step is the amplitude amplification process, which is obtained by doing Oracle function and Matrix Diffusion with $\frac{\pi}{4}N$ loops. Oracle function will return 1 for the correct state and return 0 for the wrong state. Diffusion matrix will inverse the amplitude around the amplitudes mean.

Figure 5 shows the continuation of Grover's quantum search algorithm flowchart. The program will reserved memory space with the size of the number of qubit. Then, every qubits will be measured with measure function from pyQuil library. After that, the program will open a connection to the quantum virtual machine and run the quantum program. The result will be compared with the value inside variable dataTarget to check whether or not it is the same.

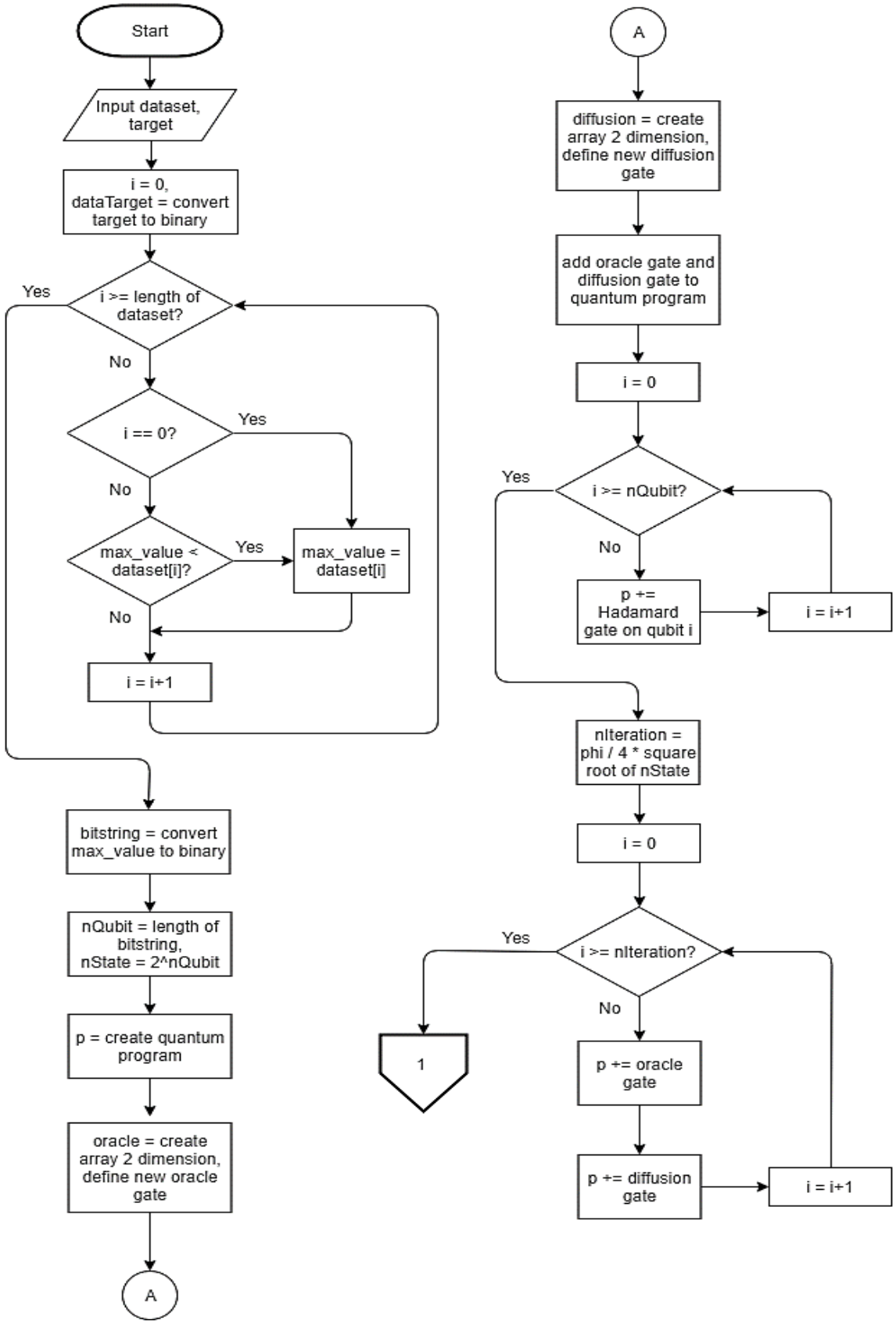

Figure 4. Grover's quantum search algorithm flowchart (part one)

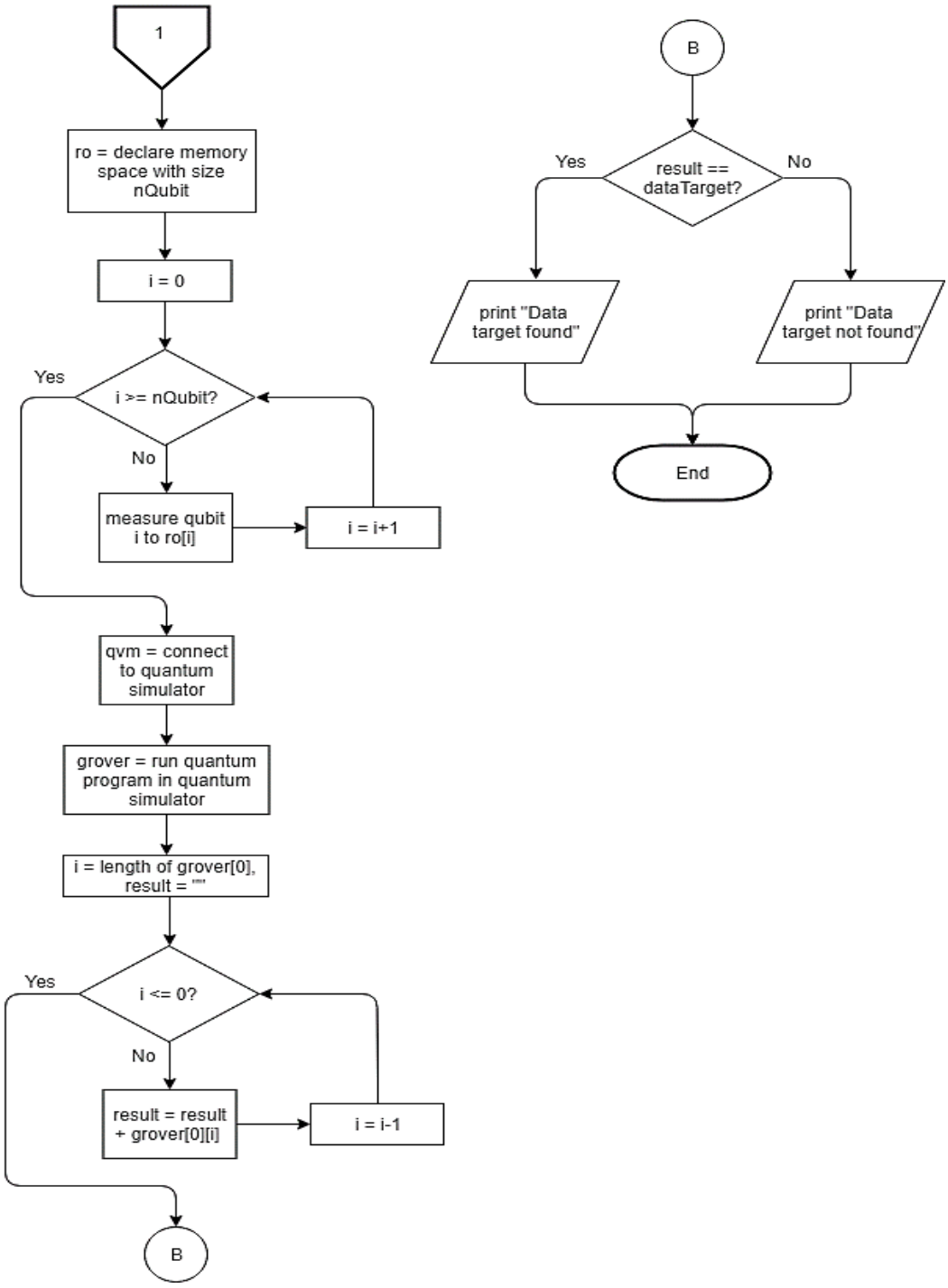

Figure 5. Grover's quantum search algorithm flowchart (part two)

3. RESULTS AND ANALYSIS

In this section, five test scenarios are used for testing the web-app implementations for both quantum algorithms respectively.

3.1. Shor's web-app realization

Here the implementation code in Python using ETH Zurich's ProjectQ framework is described. Figure 6 (a) shows the initialization phase of ProjectQ's quantum simulator. The quantum register n is allocated by the engine and put into superposition by using the Hadamard gate. The modular exponentiation process is implemented using function MultiplyByConstantModN as displayed in Figure 6 (b).

```
n = int(math.ceil(math.log(q-1, 2)))
if showMeasure:
    print("\n\tn: ", n)

a = eng.allocate_qureg(n)

All(H) | a

measurements = [0] * (n)

ctrl_qubit = eng.allocate_qubit()
```

(a)

```
for k in range(n):
    current_x = pow(x, 1 << (n - 1 - k), N)
    H | ctrl_qubit
    with Control(eng, ctrl_qubit):
        MultiplyByConstantModN(current_x, N) | a

    # and measure
    Measure | ctrl_qubit
    eng.flush()
    measurements[k] = int(ctrl_qubit)
```

(b)

Figure 6. (a) Quantum computing initialization code, and (b) modular exponentiation code

The period finding is the important step of Shor's quantum factoring algorithm. It is in fact the only part of the algorithm that requires a quantum computer. The implementation of the period finding is given in Figure 7. Figure 8 shows the final part of the algorithm which is finding the factors of the input. The web-app realization in Figure 9 shows successful design and implementation of the Shor's quantum factoring algorithm. The given input is 33 which is not even nor prime. The factors are 3 and 11 which are correctly obtained by the simulation given in Figure 9.

```
for i in range(n):
    m += measurements[n - 1 - i]*1. / (1 << (i + 1)) #Value of m/q in Hayward's

r = Fraction(m).limit_denominator(N-1).denominator
```

Figure 7. Period finding code

```
if r % 2 != 0:
    r *= 2
#STEP4-10

#STEP11
f1 = gcd((x**int(r/2)) + 1, N)
f2 = gcd((x**int(r/2)) - 1, N)
print("\tFactors:", f1, " x ", f2)
#STEP11

if ((not f1 * f2 == N) and f1 * f2 > 1 and int(1. * N / (f1 * f2)) * (f1 * f2) == N):
    f1 = f1 * f2
    f2 = int(N/(f1))

if f1 * f2 == N and f1 > 1 and f2 > 1:
    print("\n\tFactors found. F1: ", f1, "and F2: ", f2)
else:
    print("\n\tFailed to calculate factors")
```

Figure 8. Factors finding code

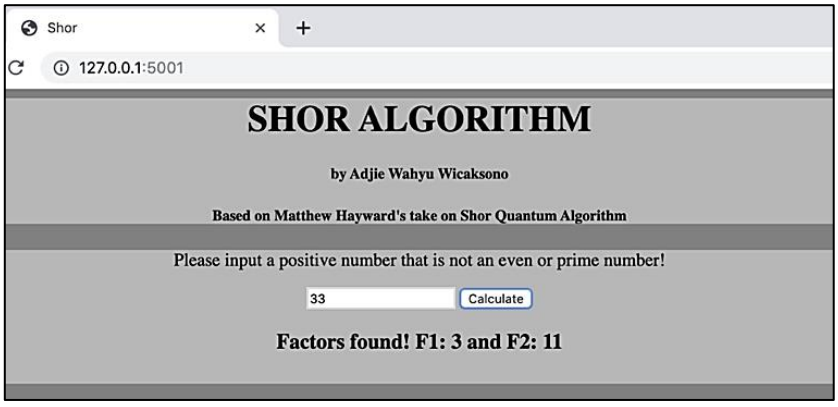

Figure 9. Shor's algorithm simulation for finding factors of 33

In Figure 10, the web-app realization of the Shor's quantum factoring algorithm is able to find the factors of 91, which are 7 and 13. This exhibits the success of the implementation of the algorithm in ProjectQ framework and also the development of the web-app for the implementation to be accessed and used using web browser conveniently.

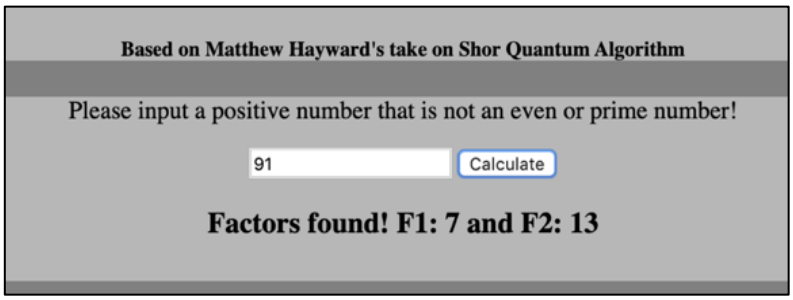

Figure 10. Shor's algorithm simulation for finding factors of 91

3.2. Grover's web-app realization

There are 3 steps in implementing the Grover's quantum search algorithm. These steps are initialization, amplitude amplification, and measure. Figure 11 shows the implementation code for initialization in Rigetti Forest. This initialization aims to make qubits have the same amplitude for each state. In this piece of code, initialization is obtained by applying the Hadamard gate to each qubit. Figure 12 shows the amplitude amplification implementation using Rigetti Forest. The number of repetitions needed is $\frac{n}{4}\sqrt{N}$. In this iteration process, the Oracle function and Diffusion matrix are applied.

```
# step i : apply hadamard gates to all qubits
for i in range(nQubit):
    p += H(i)
```

Figure 11. Superposition initialization for all qubits

```
for i in range(nIteration):
    # apply oracle function
    p += tuple(["Oracle_Function"] + list(qubit for qubit in range(nQubit)))
#     wavefunction(p)

    # apply diffusion matrix
    p += tuple(["Diffusion_Matrix"] + list(qubit for qubit in range(nQubit)))
#     wavefunction(p)
```

Figure 12. Amplitude amplification code

The Oracle function is not part of the Grover's quantum search algorithm. Figure 13 shows a piece of code that is developed originally in this research to create an Oracle function in the form of a 2-dimensional array. Diffusion matrix is part of Grover iteration and is implemented after Oracle functions. Figure 14 shows a piece of code to make a Diffusion matrix in the form of a 2-dimensional array. Figure 15 is the implementation of the agorithm to measure the result state. This measurement drives the qubits to collapse to one of its eigenstates. The measurement results are stored in the memory with size equal to the number of the qubits used. Figure 16 shows the simulation result for finding 5 from dataset containing 9, 5, 0, 11, 6 and 2. It also displays the number of qubit used for the search and the amplitudes for each of all possible states. Another simulation presented here is shown in Figure 17 where the dataset contains 1, 6, 2, 4, and 3 and the target value is 2.

```
# define oracle function
oracle = np.zeros((nState, nState))
for i in range(nState):
    for j in range(nState):
        if(i == j):
            if(bin(i)[2:].zfill(nQubit) == dataTarget.zfill(nQubit)[::-1]):
                oracle[i][j] = -1
            else:
                oracle[i][j] = 1
```

Figure 13. Oracle function code

```
### define new gate diffusion
diffusion = np.zeros((nState, nState))
for i in range(nState):
    for j in range(nState):
        if(i == j):
            diffusion[i][j] = -1+(2/nState)
        else:
            diffusion[i][j] = 2/nState
```

Figure 14. Diffusion matrix code

```
# step iii : measure
### declare classical bit to store result
ro = p.declare('ro', 'BIT', nQubit)
for i in range(nQubit):
    p.measure(i, ro[i])
```

Figure 15. Qubits measurement

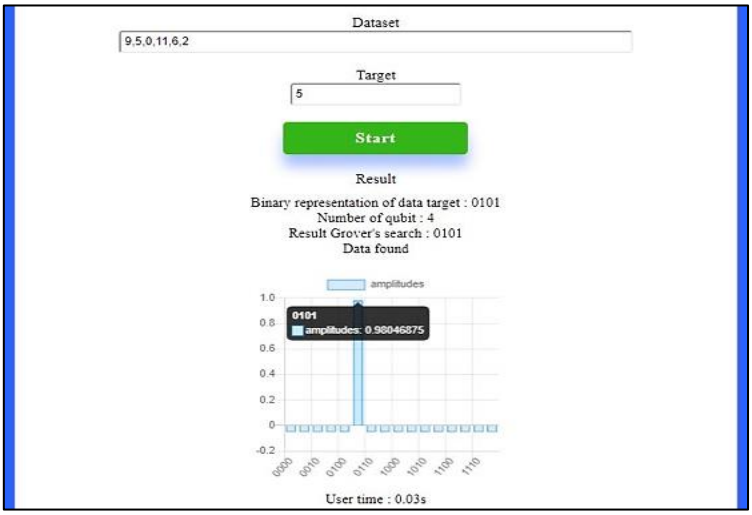

Figure 16. Grover's algorithm simulation for searching 5

Figure 17. Grover's algorithm simulation for searching 2

4. CONCLUSION

This work shows that today state-of-the-art quantum computing technologies allow the realization of quantum algorithm is possible. Here the Shor's quantum factoring algorithm and Grover's quantum search algorithm are chosen. The realization of the two quantum algorithms into a web application are using different quantum computing technology. The Shor's quantum factoring algorithm is realized using ETH Zurich's ProjectQ and the Grover's quantum search algorithm is realized using Rigetti Forest. The two quantum algorithms are successfully realized into a web application using the Flask framework. Simulations for each algorithm and problem are also given in this paper. The simulations rely on quantum simulator with only one time execution flow. The problem size that is able to be handled by the web-app follows the limitation from the respective quantum simulator.

REFERENCES

[1] M. Nielsen and I. Chuang, "Quantum Computation and Quantum Information," *American Journal of Physics*, vol. 70, no. 5, pp. 558-559, 2002.

[2] M. Ying, "Quantum computation, quantum theory and AI," *Artificial Intelligence*, vol. 174, no. 2, pp. 162-176, February 2010.

[3] D. Busvine and P. Dave, "Google unveils quantum computer breakthrough; critics say wait a qubit," U.S., 2019. [Online]. Available: https://www.reuters.com/article/us-alphabet-quantum/google-unveils-quantum-computer-breakthrough-critics-say-wait-a-qubit-idUSKBN1X21QW.

[4] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505-510, 2019.

[5] S. Kaplan, "Google scientists say they've achieved 'quantum supremacy' breakthrough over classical computers," Washingtonpost.com, 2019. [Online]. Available: https://www.washingtonpost.com/science/2019/10/23/google-scientists-say-theyve-achieved-quantum-supremacy/.

[6] N. Barde, D. Thakur, P. Bardapurkar and S. Dalvi, "Consequences and Limitations of Conventional Computers and their Solutions through Quantum Computers," *Leonardo Electronic Journal of Practices and Technologies*, vol. 10, no. 19, pp. 161-171, July 2011.

[7] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X. Zhou and J. O'Brien, "Experimental realization of Shor's quantum factoring algorithm using qubit recycling," *Nature Photonics*, vol. 6, no. 11, pp. 773-776, 2012.

[8] T. Monz et al., "Realization of a scalable Shor algorithm," *Science*, vol. 351, no. 6277, pp. 1068-1070, March 2016.

[9] A. Politi, J. Matthews and J. O'Brien, "Shor's Quantum Factoring Algorithm on a Photonic Chip," *Science*, vol. 325, no. 5945, pp. 1221-1221, September 2009.

[10] S. J. Lomonaco, "Lecture on Shor's Quantum Factoring Algorithm Version 1.1," Arxiv, 2000. [Online]. Available: https://arxiv.org/pdf/quant-ph/0010034.pdf.

[11] S. Nagaich, Y. C. Goswami, "Shor's Algorithm for Quantum Numbers Using MATLAB Simulator," *2015 Fifth International Conference on Advanced Computing & Communication Technologies*, Haryana, pp. 165-168, 2015.

[12] L. K. Grover, "From Schrodinger's equation to the quantum search algorithm," *American Journal of Physics*, vol. 69, no. 7, pp. 769-777, July 2001.

[13] R. LaRose, "Overview and Comparison of Gate Level Quantum Software Platforms," *Quantum*, vol. 3, no. 130, March 2019.

[14] E. Gibney, "Hello quantum world! Google publishes landmark quantum supremacy claim," *Nature*, vol. 574, no. 7779, pp. 461-462, 2019. Available: 10.1038/d41586-019-03213-z.

[15] S. Aaronson, "Opinion | Why Google's Quantum Supremacy Milestone Matters," Nytimes.com, 2019. [Online]. Available: https://www.nytimes.com/2019/10/30/opinion/google-quantum-computer-sycamore.html.

[16] Rigetti Computing, "Installation and Getting Started," 2019. [Online]. Available: http://docs.rigetti.com/en/stable/start.html.

[17] D. Steiger and T. Häner, "ProjectQ," 2019. [Online]. Available: http://projectq.ch/code-and-docs/.

[18] Quantum Computing Playground, "Help," Quantum Computing Playground, 2014. [Online]. Available: http://www.quantumplayground.net/#/about.

[19] "Quantum Computing: Progress and Prospects," *The National Academies Press*, April 2019.

[20] C. Boyer, "Duetsch-Jozsa's Algorithm in Quantum Computing | Topcoder," Topcoder, 2019. [Online]. Available: https://www.topcoder.com/duetsch-jozsas-algorithm-in-quantum-computing/.

[21] D. Knuth, "Simon's Algorithm in Quantum Computing | Topcoder," Topcoder, 2019. [Online]. Available: https://www.topcoder.com/simons-algorithm-in-quantum-computing/. [Accessed: 20- Jan- 2020].

[22] M. Grossi, "Build and deploy Quantum-based web Applications using Qiskit & Python Flask on IBM Cloud," Medium, 2019. [Online]. Available: https://medium.com/build-and-deploy-quantum-based-web-applications/build-and-deploy-quantum-based-web-applications-using-qiskit-python-flask-on-ibm-cloud-b0cb0a01e5f2.

[23] M. Hayward, "Quantum Computing and Shor's Algorithm," Imsa.edu, 2015. [Online]. Available: https://pdfs.semanticscholar.org/8072/dc7247460849b18abbb463429a09cfb2e3e6.pdf.

[24] A. W. Wicaksono and A. Wicaksana, "Implementation of Shor's Quantum Factoring Algorithm Using ProjectQ Framework," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 8, no. 6S3, pp. 52-56, September 2019.

[25] Anthony and A. Wicaksana, "Implementation of Grover's Quantum Search Algorithm using Rigetti Forest," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 8, no. 6S3, pp. 110-114, September 2019.

BIOGRAPHIES OF AUTHORS

Arya Wicaksana is graduated from Universiti Tunku Abdul Rahman in VLSI Engineering (M.Eng.Sc.) and Universitas Multimedia Nusantara in Computer Science (S.Kom.). Research interests and works are: quantum computing and computational intelligence.

Anthony received S. Kom., in Informatics from Universitas Multimedia Nusantara, Indonesia. His research interests and works are quantum computing and software engineering.

Adjie Wahyu Wicaksono received S. Kom., in Informatics from Universitas Multimedia Nusantara, Indonesia. Currently working as full-stack web developer. His research interests and works are quantum computing and software engineering.