Python Application Engineer Assignment

Ahmet Özkul ozkulah@gmail.com

I will explain algorithms and how to use the program in this report.

1. Requirements

I used python3.6 and Pycharm to develop this project. For linux install;

```
sudo apt-get update
sudo apt-get install python3.6
```

I used virtual environment to separate libraries etc.

```
cd flask_project/
pip install virtualenv
virtualenv env
source env/bin/activate
```

I used quart framework for the web service and some other libraries. You can install them;

```
cd flask_project/
pip install -r requirements.txt
```

2. Algorithms

In the assignment, Fibonacci, Ackermann and Factorial algorithms are wanted in a web service. These algorithms are classic recursive algorithms but python have a recursive limit so I prefer to code them in a iterative way. I used quart framework to provide async operations. That will prevent locking in service but if the response takes more time than browser/caller client timeout limit, it can give error..

2.1 Fibonacci Calculation

The Fibonacci numbers, commonly denoted Fn form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F_0 = 0 \quad F_1 = 1$$

and

$$F_n = F_{n-1} + F_{n-2}$$

I used simple swap operation and summed values for next fibonacci calculation.

```
def calculate_fibonacci(self, n):
        fib1, fib2 = 0, 1
        for i in range(0, n):
```

```
            fib1, fib2 = fib2, fib1 + fib2
        return fib1
```

2.2 Ackermann Calculation

Ackermann function, is defined as follows for nonnegative integers m and n:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m >= 0 \text{ and} n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and} n > 0 \end{cases}$$

I used a stack and put all m values in it and used an iterative solution.

```
def calculate_ackermann(self, m, n):
        ackermann_vals = []
        ackermann_vals.append(m)
        while ackermann_vals:
            m = ackermann_vals.pop()
            if m == 0:
                n = n + 1
            elif n == 0:
                n = 1
                ackermann_vals.append(m - 1)
            else:
                n = n - 1
                ackermann_vals.append(m - 1)
                ackermann_vals.append(m)
        return n
```

2.3 Factorial Calculation

The factorial of a positive integer n, denoted by n!, is the product of all positive integers less than or equal to n:

$$n! = n \times (n-1) \times (n-2) \ldots \times 3 \times 2 \times 1$$

I used for loop and multiply operation to calculate it in a iterative way.

```
def calculate_factorial(self, fact):
        factorial = 1
        if fact >= 1:
            for i in range(1, fact + 1):
                factorial = factorial * i
            return factorial
        elif fact == 0:
            return 1
        else:
            return Utility.not_positive
```

3. Web Service

There are three service calls for this project. After running server part, you can use client request from a web browser or Postman or console. I used Postman to test web services.

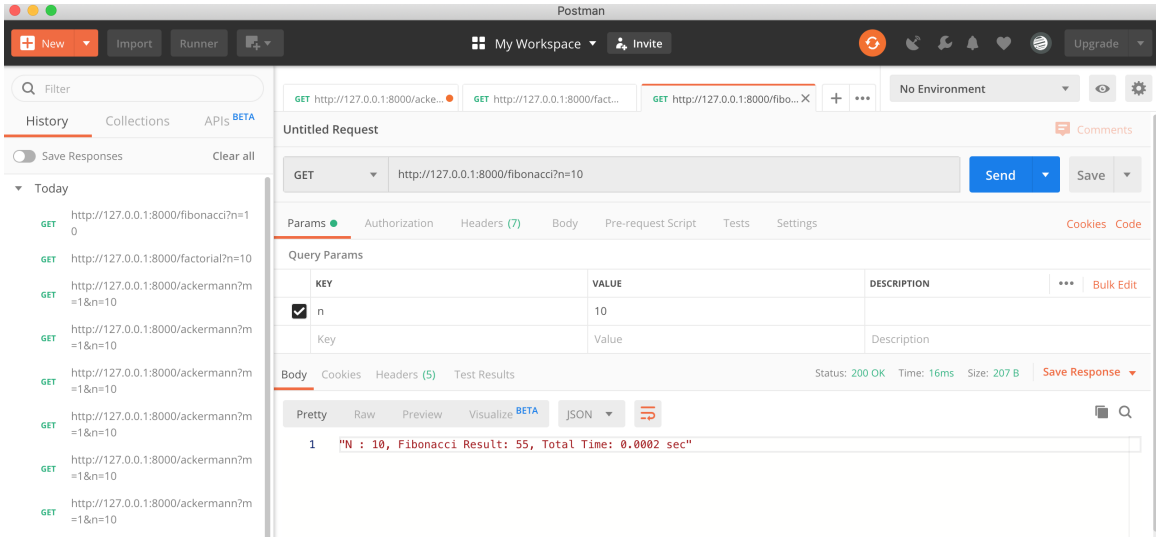

Figure 1: Fibonacci call for n=10

We first run the server which will service for the client request. I used async methods so we don't need "gunicorn" library or similar think, multiple client can ask simultaneously. You can run server with;

```
cd flask_project/src/
python main.py
```

"w" represent worker thread count for the service. I used 4 threadS for this project.

3.1 Fibonacci web service

You can use a web browser and type below address;

```
http://127.0.0.1:8000/fibonacci?n=10
```

In figure 1, you can see the fibonacci web service call for n = 10.

3.2 Ackermann web service

You can use a web browser and type below address;

```
http://127.0.0.1:8000/ackermann?m=2&n=10
```

In figure 2, you can see the ackermann web service call for m = 2 and n=10.

3.3 Factorial web service

You can use a web browser and type below address;

```
http://127.0.0.1:8000/factorial?n=10
```

In figure 3, you can see the factorial web service call for n = 10.

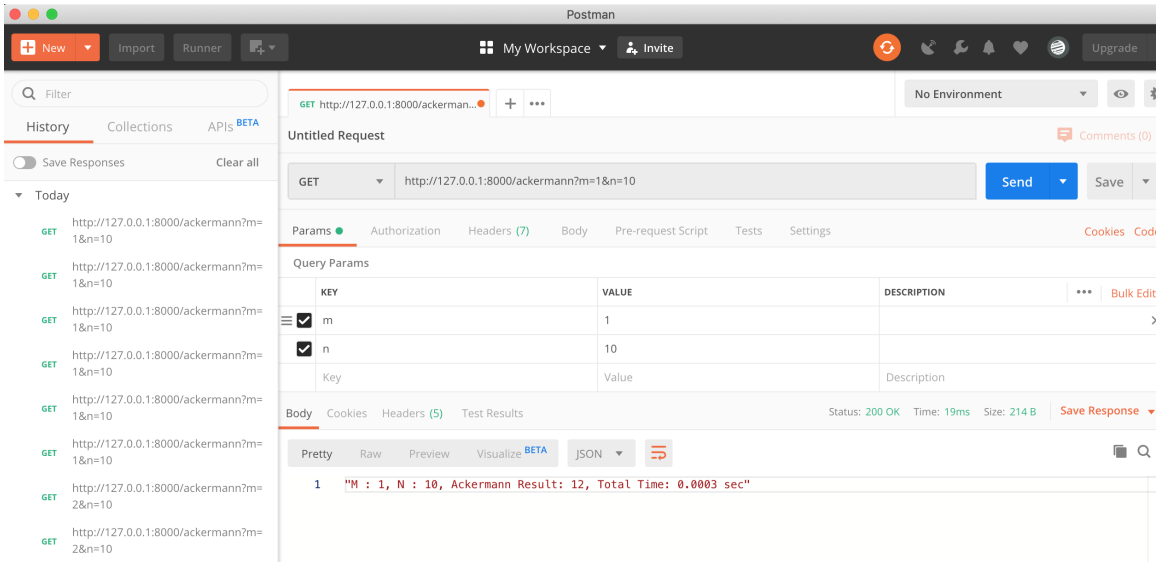

Figure 2: Ackermann call for m=2, n=10

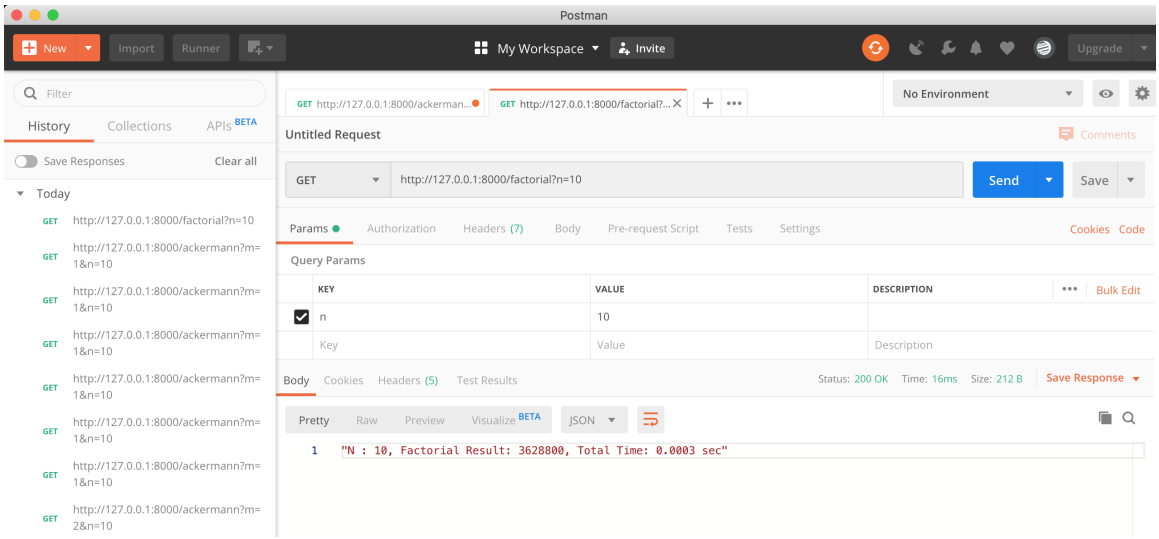

Figure 3: Factorial call for n=10

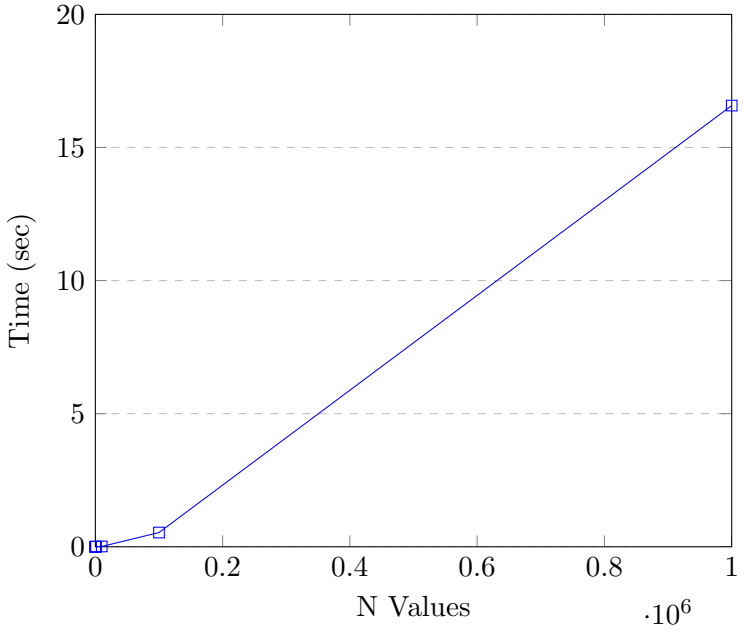

Figure 4: Fibonacci algorithm response times

4. Unit Testing

I used postman to test web services and used pytest to test factorial, ackermann and fibonacci algorithms. In the project, run below command to see the test result for these algorithms;

```
cd flask_project/
pytest
```

It will test each algorithm with a bunch of values and check responses. Each method checked with three standard values like n=0, n=1; n=10.

5. Algorithm Performances

I put upper limits in web service to prevent timeout and python can calculate very big numbers but they are not normally preferred, such as a number with more than 100 digit etc. You can see response times of algorithms in figures 4, 5, 6.

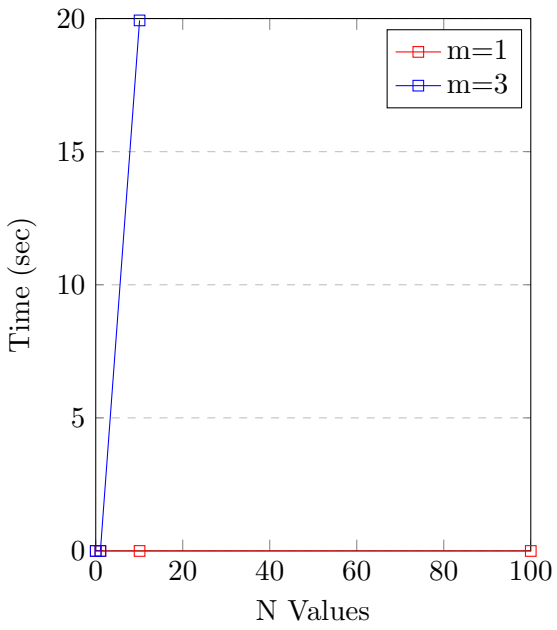

Figure 5: Ackermann algorithm response times

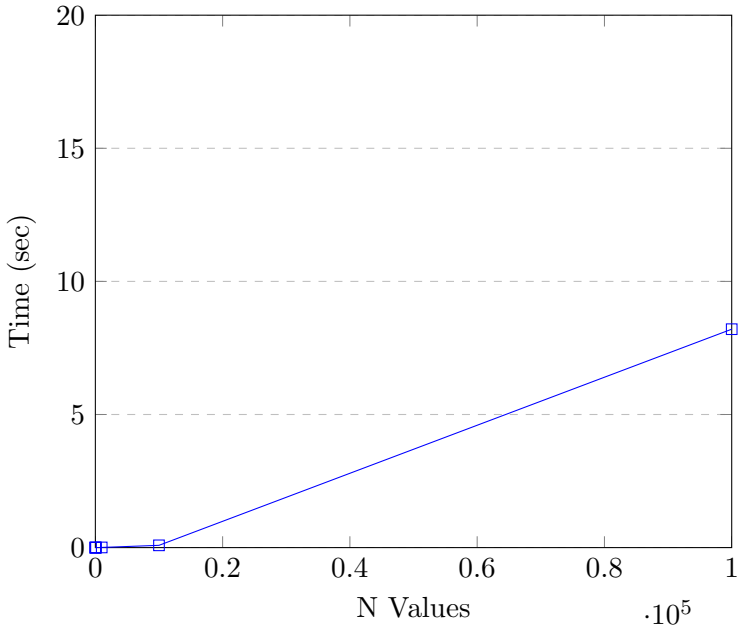

Figure 6: Factorial algorithm response times