Qinyun Lin (NUID: 001582464)

Program Structures & Algorithms

Fall 2021

Assignment No. 3

- **Task**
  - o Implement height-weighted Quick Union with Path Compression
  - o Create a method count(), which take an integer value n and return the number of connecting operations that make all “sites” are connected
  - o Create a main program, which runs the count() with a set of n values, and print the returned values
  - o Deduce the relationship between the number of objects (n) and the number of pairs (m) generated to make all objects connected

- **Relationship Conclusion:**

$$m = 0.53 * n * \ln(n)$$

- **Evidence to support the conclusion:**

1. Output

I used 5000 different values of n, ranging from 10 to 50000. For each n, run 10 times and take the average value of m.The output is shown below:

```
Run:    UF_HWQUPC_Driver
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
======== Create file writer success! ========

======== Test on Union-Found started! ========
        components   using steps
            10           15
            20           38
            30           52
            40           87
            50           115
            60           143
            70           174
            80           206
```

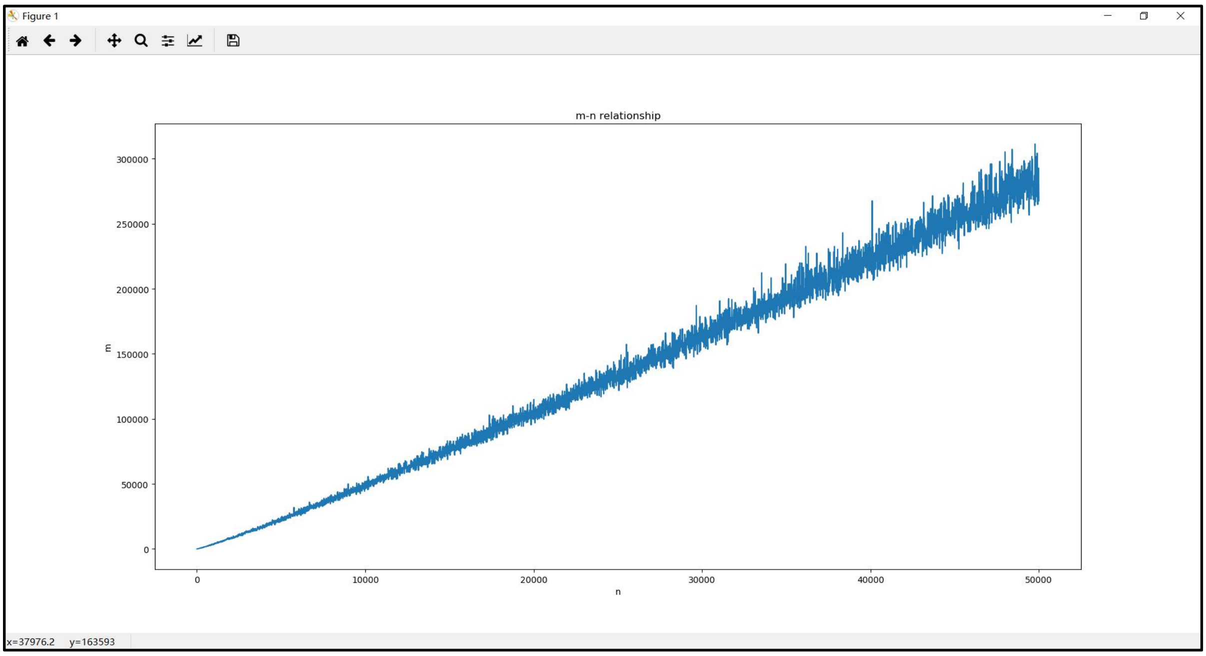
```
Run:    UF_HWQUPC_Driver
            49930        264825
            49940        280515
            49950        273152
            49960        275538
            49970        284935
            49980        292658
            49990        278631
            50000        267573

======== Results have been written to a file! =======

Process finished with exit code 0
```

2. **Graphical Representation**

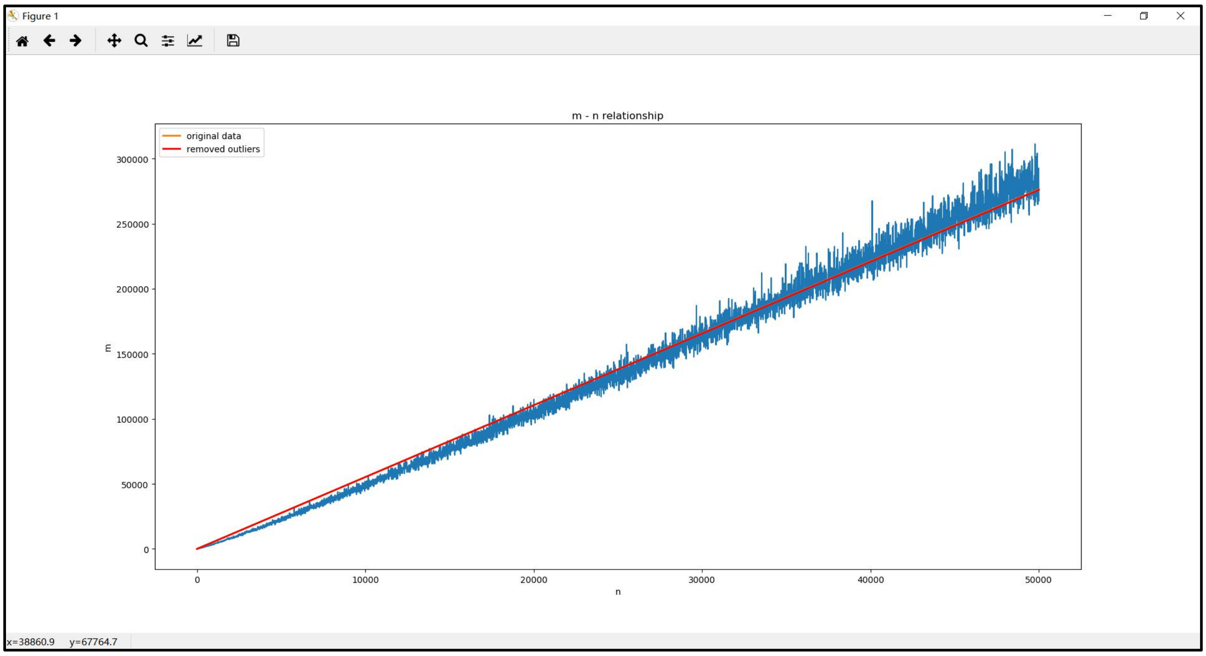
I saved the output above to a file, then used python to do visualization. The m-n relationship graph is shown below, it seems there is a linear relationship between m and n.



I tried to use "m = kn" and LSE method to fit it. Because of many outliers existing, I set a rule to deal with outlier and make the graph smother:

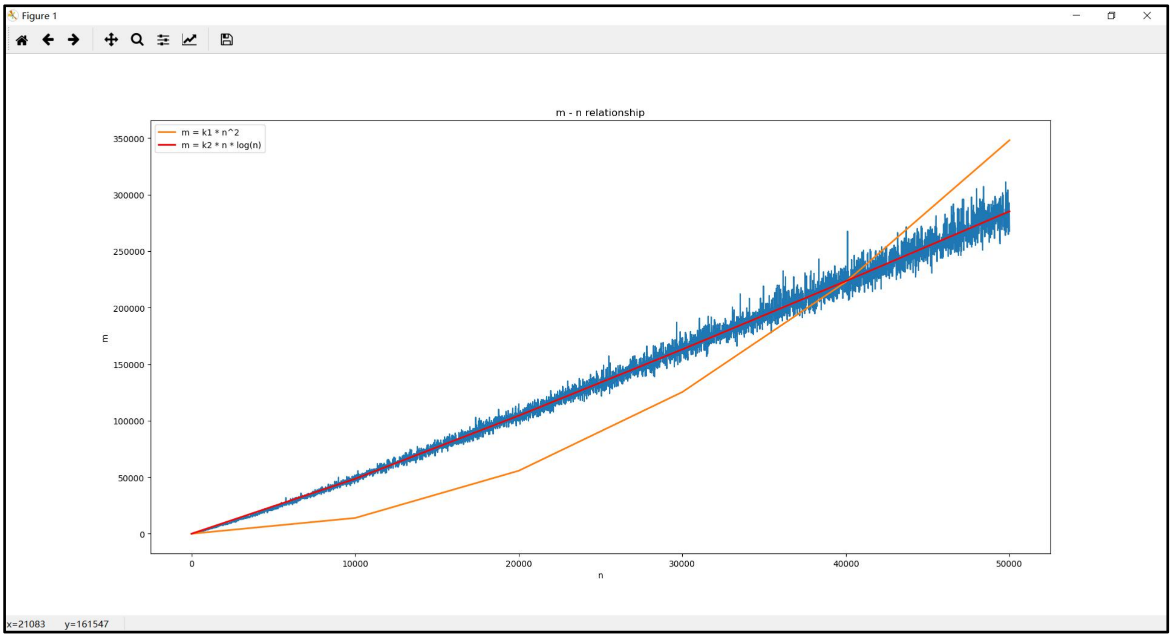
*For all the m value, if m[i] / m[i-1] > 110%, set m[i] = 0.8 * m[i-1] + 0.2 * m[i].*

Result is shown below:



According to the graph, I found that whether removing outliers has low influence to the result. Besides, linear function is obvious not good for the relationship. As n goes up, the fluctuation of m is sharper, and the slope is growing slightly.

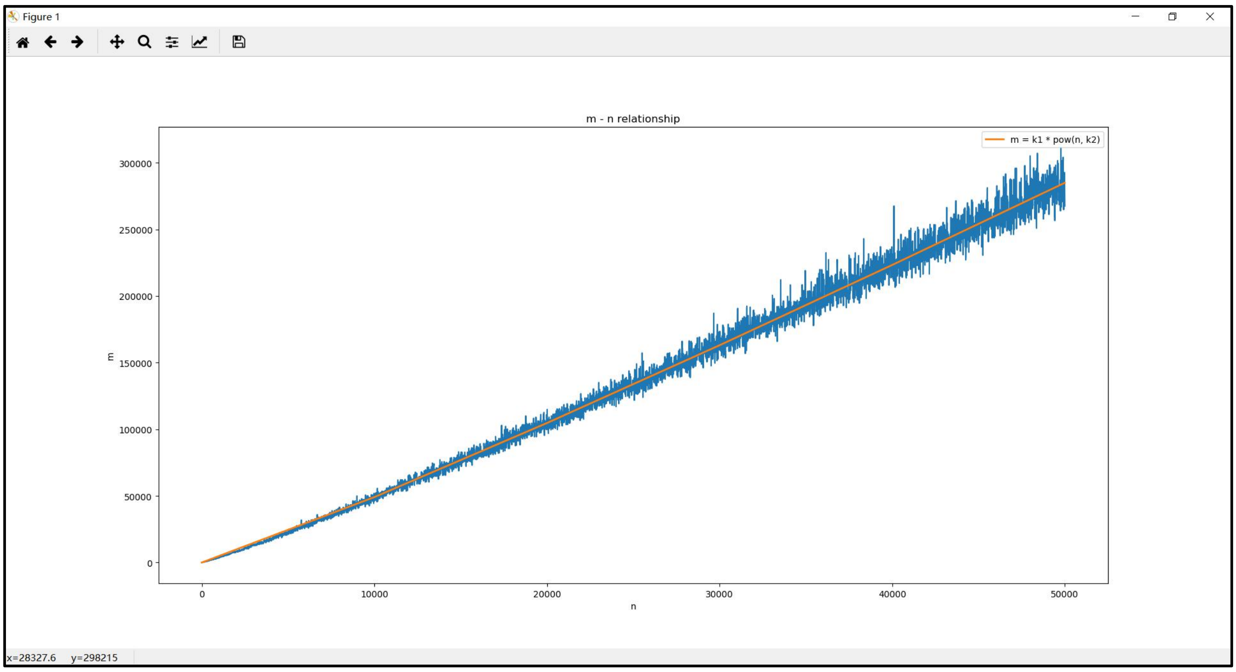
So I tried to use m = k * n^2 and m = k2 * n * log(n) to fit it. The result is shown below:



It is obvious that m = k2 * n * log(n) fits it better. The relationship is:

```
result expression: m = 0.527212458832 * n * log(n)
```

Besides, I also tried to fit another function, assume m = k * pow(n, p)). The result is shown below:



“m = k * pow(n, p))” seems also good. But when I change the range of n and recalculate the parameters of each functions, all the parameters changes a lot except “m = k * n * log(n) ”. The result is shown below:

Ranging n from 10 to 10000:

```
result expression: m = 4.71114801001 * n
result expression: m = 4.69122672807 * n
result expression: m = 0.000593756175609 * n ^ 2
result expression: m = 0.530634245334 * n * log(n)
result expression: m = 1.79258547526 * pow(n, 1.1087728962 )
```

Ranging n from 10 to 20000:

```
result expression: m = 5.52893731645 * n
result expression: m = 5.52105512809 * n
result expression: m = 0.000139247077911 * n ^ 2
result expression: m = 0.527212458832 * n * log(n)
result expression: m = 2.0879167785 * pow(n, 1.09281997234 )
```

Ranging n from 10 to 50000:

```
result expression: m = 5.07891864208 * n
result expression: m = 5.06814507938 * n
result expression: m = 0.000320106563214 * n ^ 2
result expression: m = 0.530683881949 * n * log(n)
result expression: m = 1.82145923286 * pow(n, 1.1070841526 )
```

It is obvious that “m = k * n * log(n) ” is the stablest and best. So I determined the relationship between m and n is:

$$m = 0.53 * n * \ln(n)$$

● **Unit tests result:**