Assignmment 4

Step 1:
(a) Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF_HWQUPC. All you have to do is to fill in the sections marked with // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION.

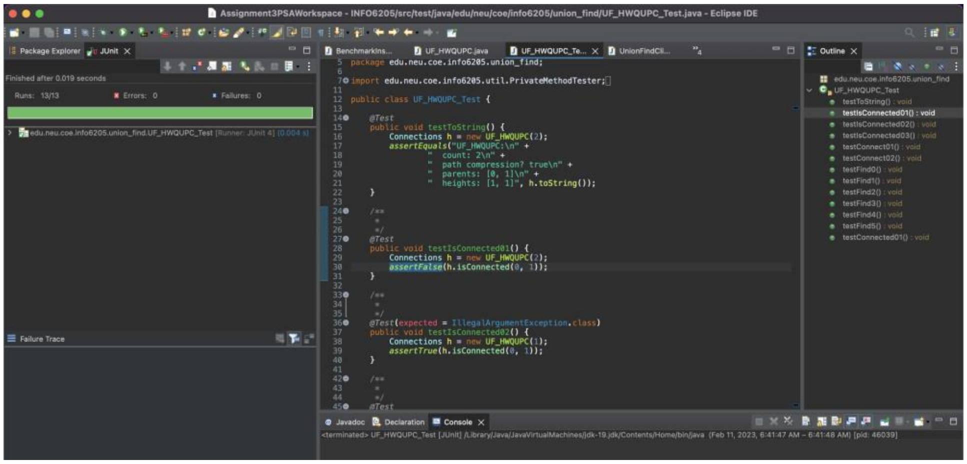
(b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

Step 2:
Using your implementation of UF_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop until all sites are connected then print the number of connections generated. Package your program as a static method count() that takes n as the argument and returns the number of connections; and a main() that takes n from the command line, calls count() and prints the returned value. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).

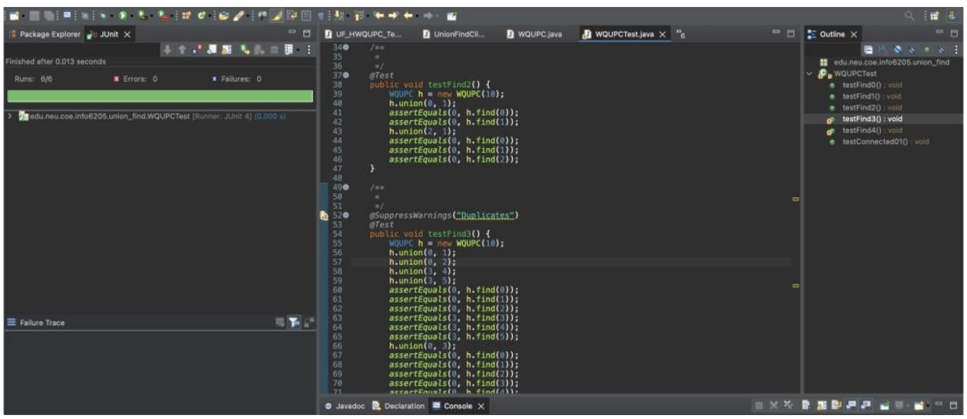
Step 3:
Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion in terms of your observations and what you think might be going on.

Unit Test Case for Step 1:

UF_HWQUPC_Test.java



WQUPCTest.java



Observation and Evidence:

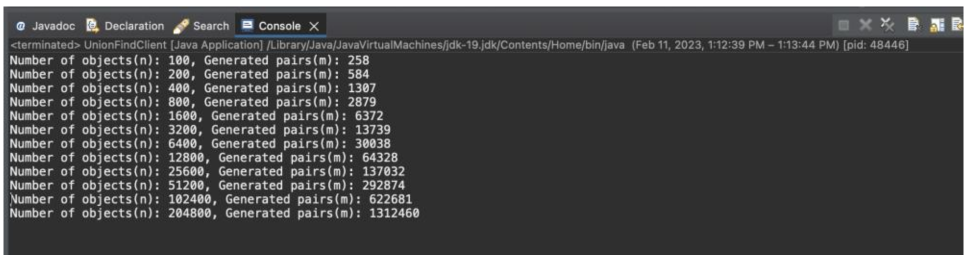
The observation for this relationship would likely be a correlation between the value of n and the number of pairs m needed to reduce the number of components from n to 1. For each sites count n, the weighted quick-union by height 1000 times is done.

Using the implementation of UF_HWQUPC.java, I created a client class named as UnionFindClient.java accepts an integer N as the number of sites. The client then generates random combinations of integers between 0 and N-1 and checks if they are connected by using the connected() function. If they are not connected, the union() function is called. This process continues until all sites are connected, and the result is the number of connections made. The program has been packaged as a static method called count() and can be invoked in the main() method.

In the primary function of the program, to ensure efficiency and precision, I established 12 values for the number of sites, “n”, with a range from 100 to 204800. For each n, I computed the results 1000 times and took the average to determine the number of connections required to connect all the sites. The results of my calculations are presented in the below image:

Screenshot of the observation:

The results are taken from the class UnionFindClient.java:

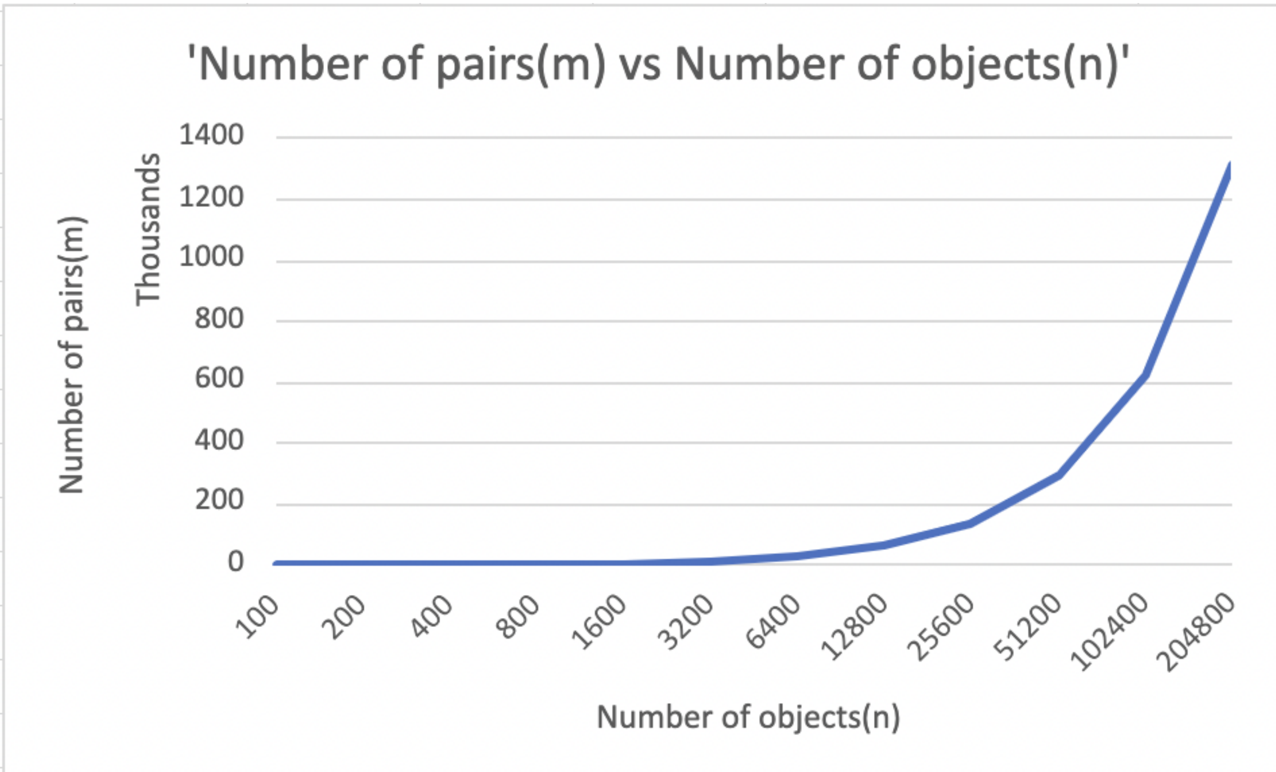


```
Javadoc  Declaration  Search  Console
<terminated> UnionFindClient [Java Application] /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java (Feb 11, 2023, 1:12:39 PM – 1:13:44 PM) [pid: 48446]
Number of objects(n): 100, Generated pairs(m): 258
Number of objects(n): 200, Generated pairs(m): 584
Number of objects(n): 400, Generated pairs(m): 1307
Number of objects(n): 800, Generated pairs(m): 2879
Number of objects(n): 1600, Generated pairs(m): 6372
Number of objects(n): 3200, Generated pairs(m): 13739
Number of objects(n): 6400, Generated pairs(m): 30038
Number of objects(n): 12800, Generated pairs(m): 64328
Number of objects(n): 25600, Generated pairs(m): 137032
Number of objects(n): 51200, Generated pairs(m): 292874
Number of objects(n): 102400, Generated pairs(m): 622681
Number of objects(n): 204800, Generated pairs(m): 1312460
```

The supposition was that it would take approximately 1/2 n log n pairings to execute the height-weighted quick union with path compression process, effectively reducing the number of components from n to just 1. To validate this hypothesis, I computed 1/2 n log n for various n values and organized the information, including n, the output from the program, and the calculated 1/2 n log n, into a data sheet displayed below:

Number of objects(n)	**Number of pairs(m)**	**fn = 0.5 * n * log(n)**
100	258	100
200	584	230.1
400	1307	520.41
800	2879	1161.2
1600	6372	2563.29
3200	13739	5608.23
6400	30038	12179.77
12800	64328	26286.14
25600	137032	56425.4
51200	292874	120557.31
102400	622681	256527.35
204800	1312460	543880.18

Plotted Graph from the raw timing observations of the above table:



Conclusion:

However, the general conclusion can be that as n increases, the number of pairs m needed to reduce the number of components from n to 1 will also increase. This is because as the number of objects increases, the chances of any two objects being connected become smaller, thus requiring more pairs to be generated and union operations to be performed to reduce the number of components to 1.

Slope of the line(m) =(y2-y1)/(x2-x1)=(1312460-258)/(204800-100) = 1107502

Also, the estimate for the number of pairs necessary to bring all initial sites together using the height-weighted quick union with path compression is approximately 0.5 n log(n), where n represents the original quantity of sites and log n represents the natural logarithm of n.