Program Structures and Algorithms
Spring 2023(SEC – 1)

NAME: Foram Kamani
NUID: 002732551

Task:

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 (the 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.

Relationship Conclusion:

The relationship between the number of objects (n) and the number of randomly generated pairs (m) needed to reduce the number of components/objects from n to 1 can be described as follows:

m is proportional to n times the logarithm of n, with the logarithm taken to the base of 2. This relationship was determined by taking the average of m over 100 runs of a program, with the value of c, which is approximately equal to 1.22, being determined from the results of runs with n ranging from 1000 to 512000 (doubling in each iteration).

m = c * n * log(n)
where c = m/n * log(n) which is approximately equal to 1.22 as observed over n ranging from 1000 to 512000(doubling)

Therefore we can summarize the relationship as
m ∝ n * log(n)

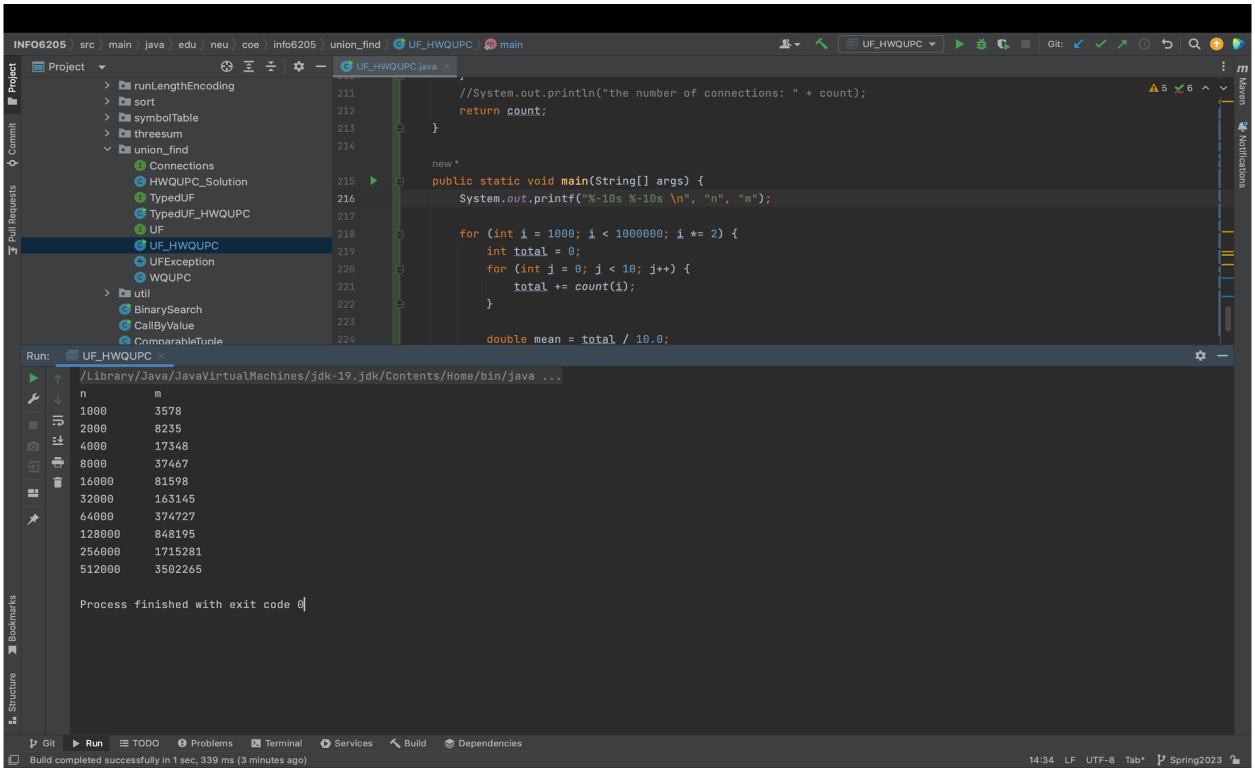
Evidence to support that conclusion:

1. Values of n are ranging from 1000 to 2048000(doubling each time).

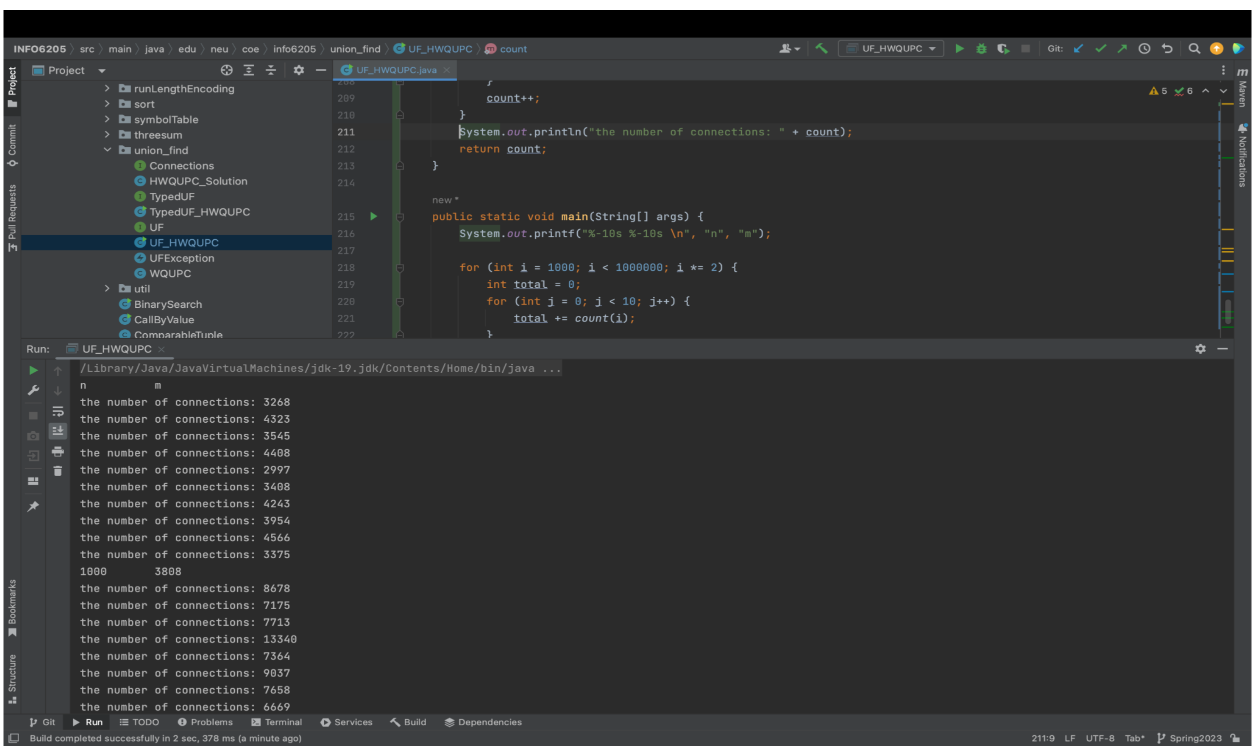
2. For each value of n, the program is running 100 times, and the average value of m is shown below.

3. The constant c = m/n * log(n) can be approximated to 1.22 And therefore the relationship can be established as

m = c* n* log(n)   or   m ∝ n* log(n)

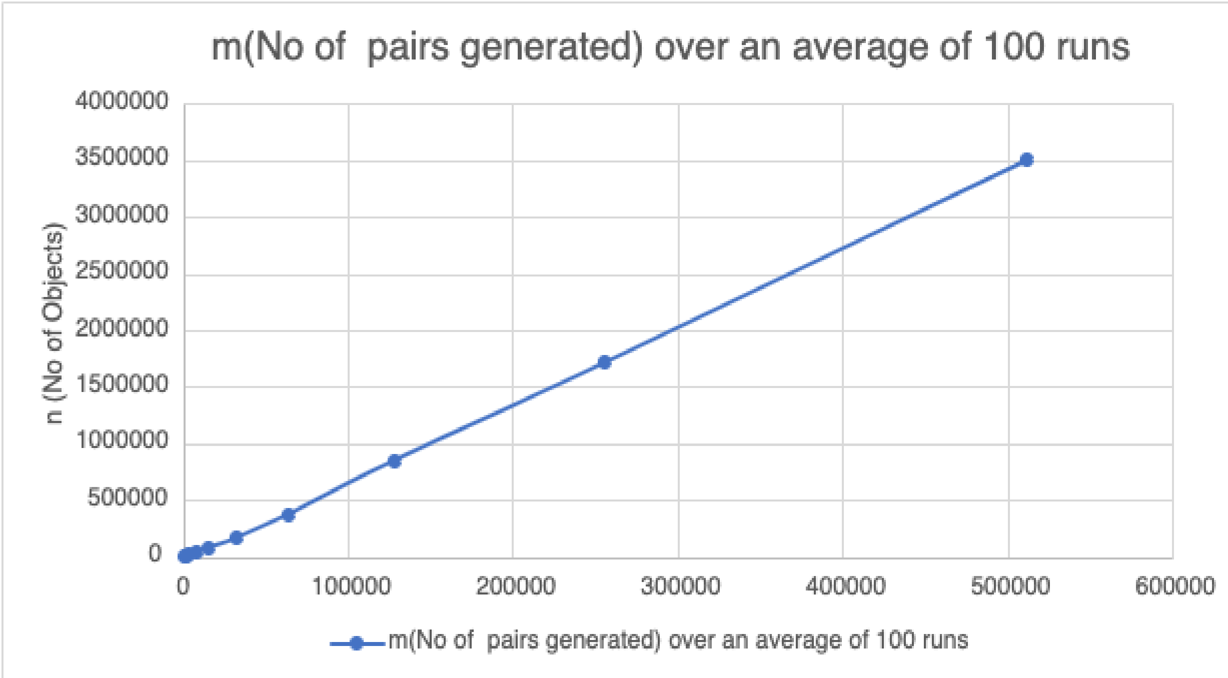


No. of Connections screenshots:



Graphical Representation:

n(No of Objects)	m(No of pairs generated) over an average of 100 runs
1000	3578
2000	8235
4000	17348
8000	37467
16000	81598
32000	163145
64000	374727
128000	848195
256000	1715281
512000	3502265



Unit Test Screenshots: