Archil Lelashvili – NUID 001522269
Program Structures & Algorithms, Spring 2021
Assignment No 3

Task:

1. Implement height-weighted Quick Union with Path Compression. Complete the class UF_HWQUPC.

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).

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.

Output:

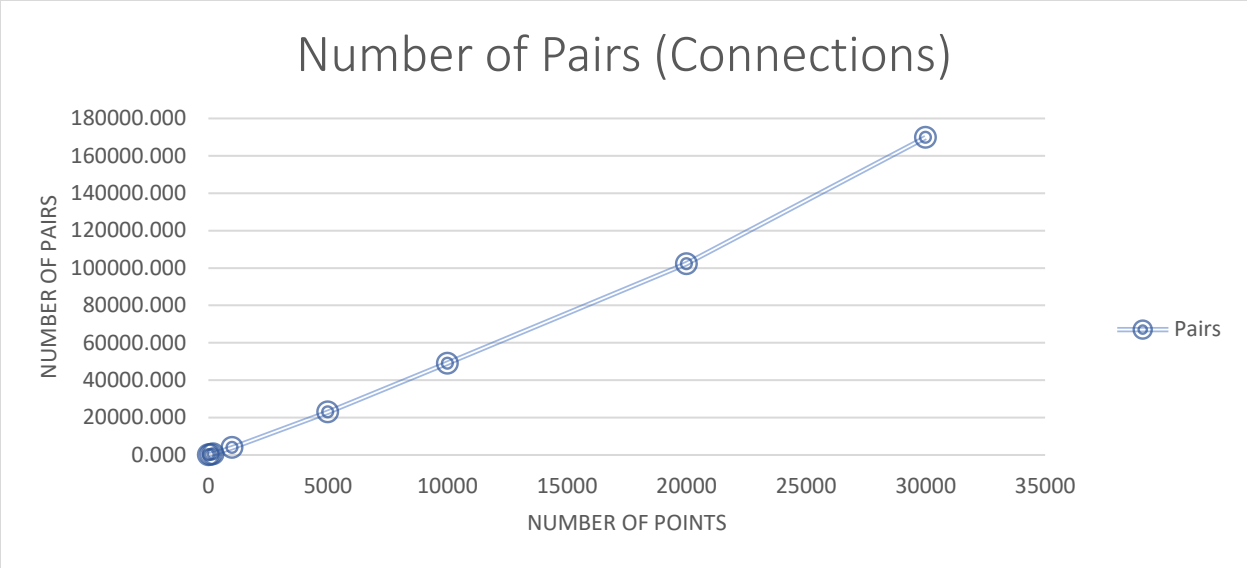
Step 1 complete as as required. Please the unit tests screenshot below.

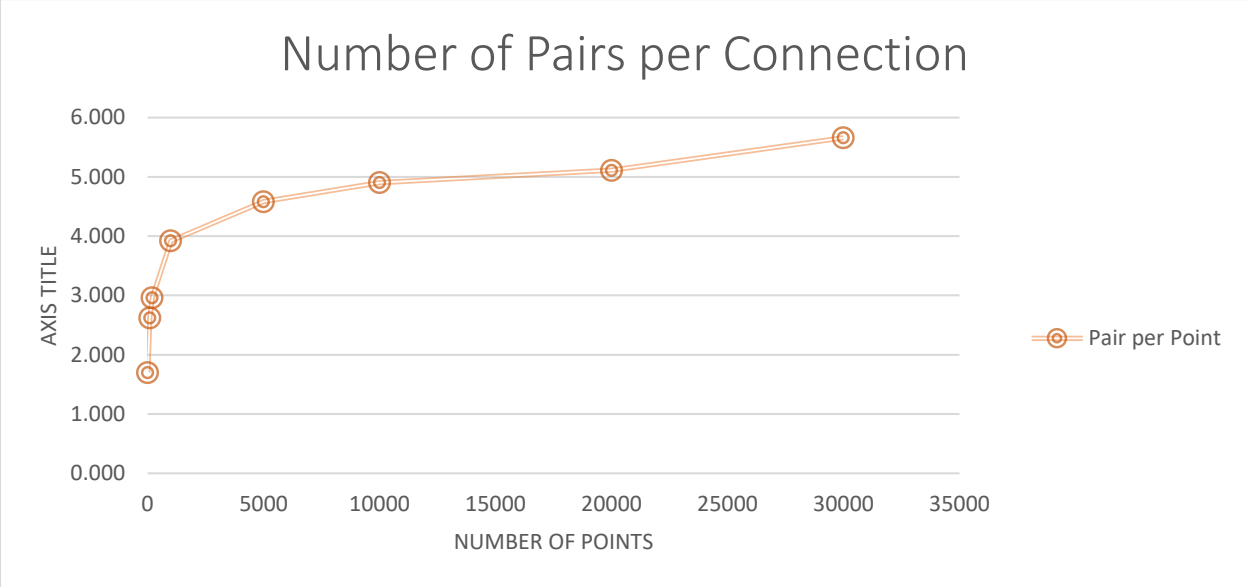
Step 2 implemented via separate UF_Client class that contains public static main method and calls count() method to determine the number of pair required to connect the points. Because the pairs are generated randomly their number can vary on each run. To compensate this variance the main() method call count() 50 times for every specific n and outputs the average number.

Step 3. Based on Step 2 we have a table and a chart. Please see below.

Table 1. Number of points

N	10	100	200	1000	5000	10000	20000	30000
Pairs	17.000	262.000	592.000	3922.000	22914.000	49042.000	102239.000	169774.000
Pair per Point	1.700	2.620	2.960	3.922	4.583	4.904	5.112	5.659





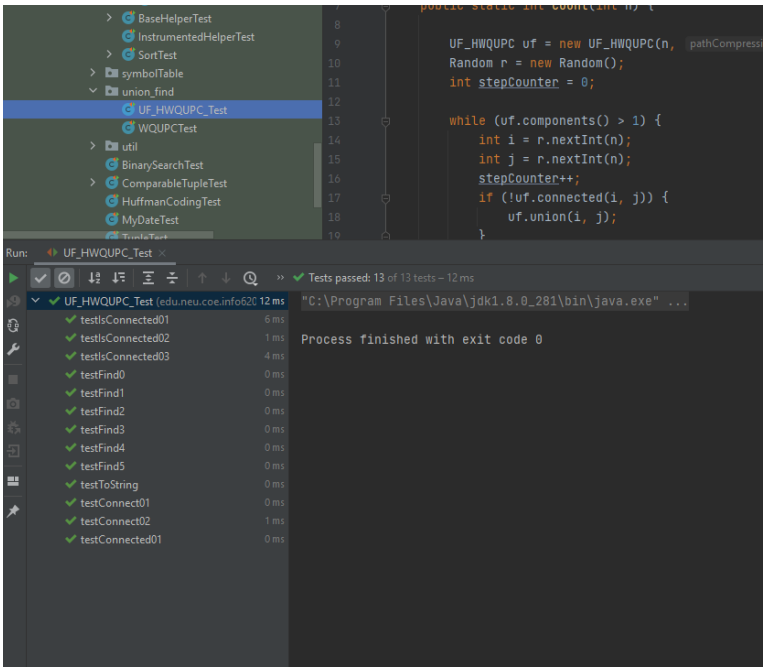
Relationship Conclusion:

Looking at the charts we see that the number of pairs needed to connect points very slowly increases over time. Pair per point growth is logarithmic.

The total number of points is almost linear but looks more like **O(nlogn)** complexity.

Unit Test Results:

Step 1:



Step 2 & 3: