Assignment 3 - PSA

Name: Koovehithilu Shrikrishna Joisa

NUID : 002920963

Github Link: https://github.com/falcon-head/PSA-Assignments.git (Private repo)

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 (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).

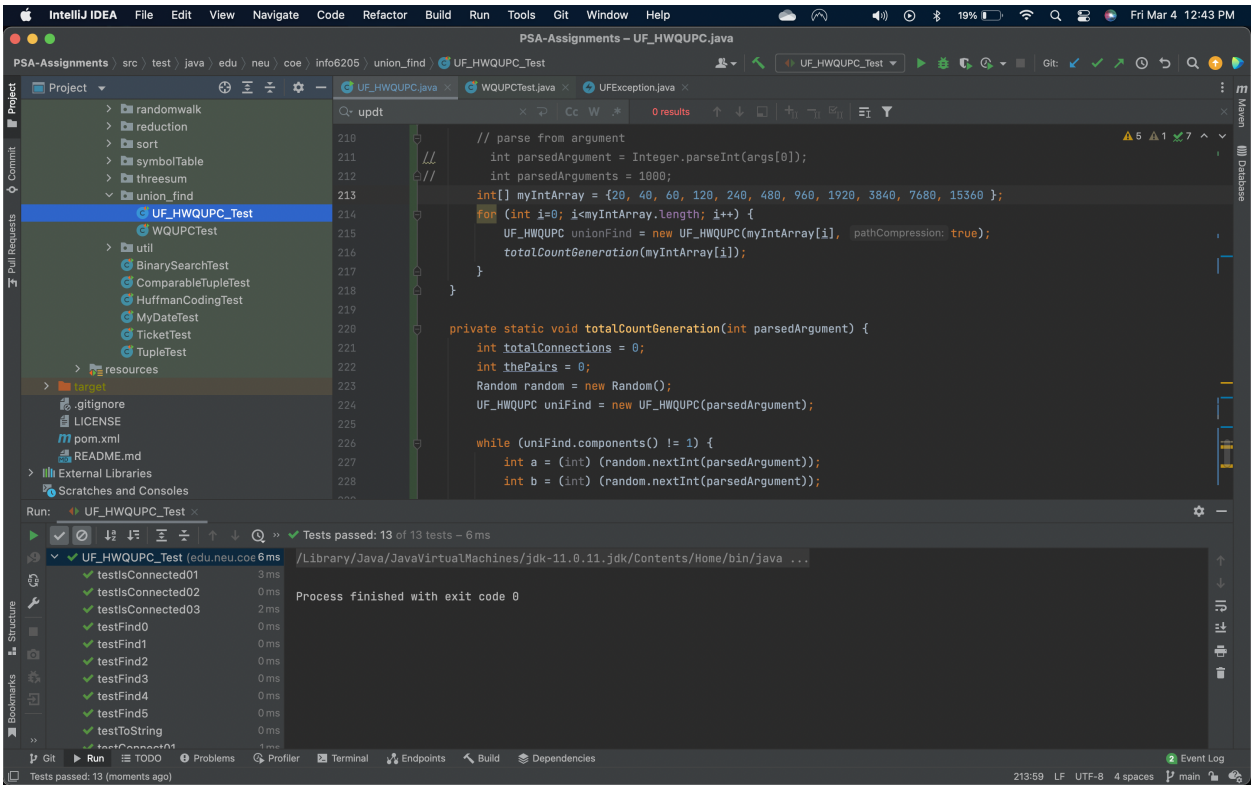
Step 3:Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion in terms of your observations and what you think might be going on.

NOTE: although I'm not going to tell you in advance what the relationship is, I can assure you that it is a *simple* relationship.

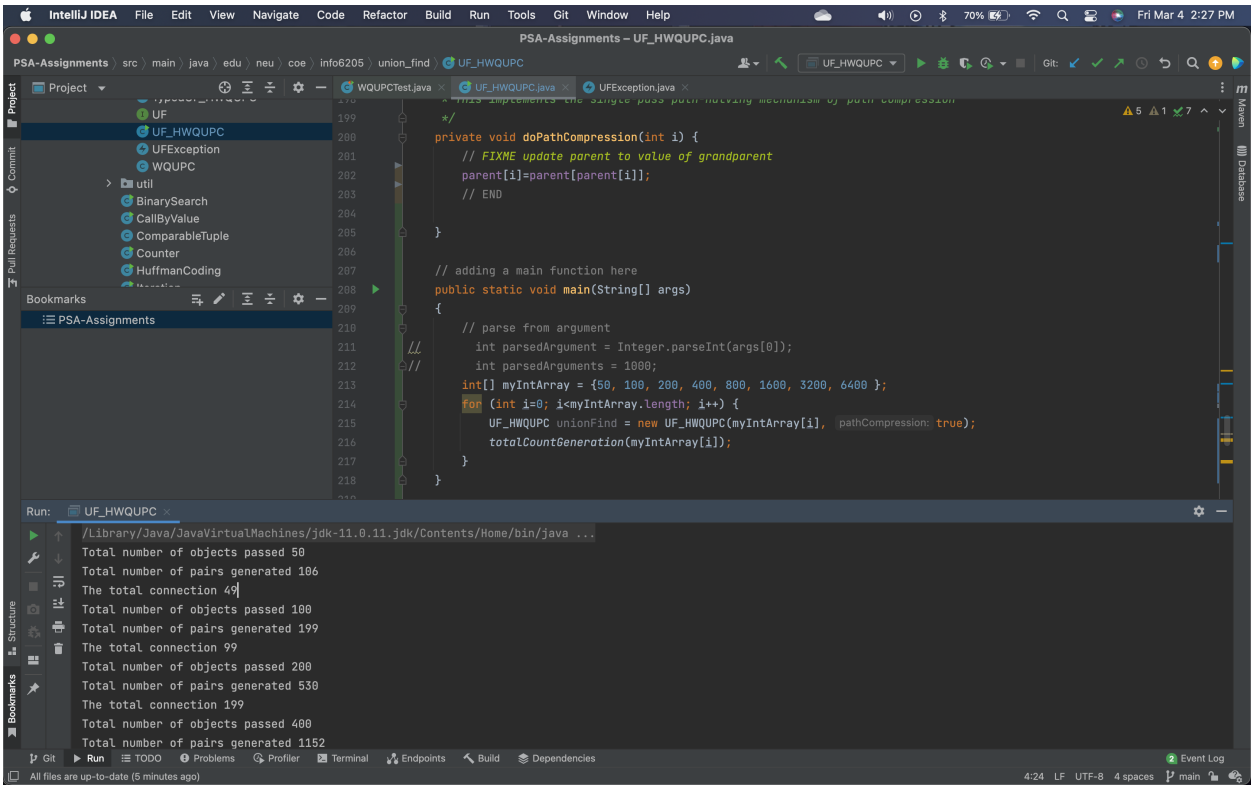
Don't forget to follow the submission guidelines. And to use sufficient (and sufficiently large) different values of n.

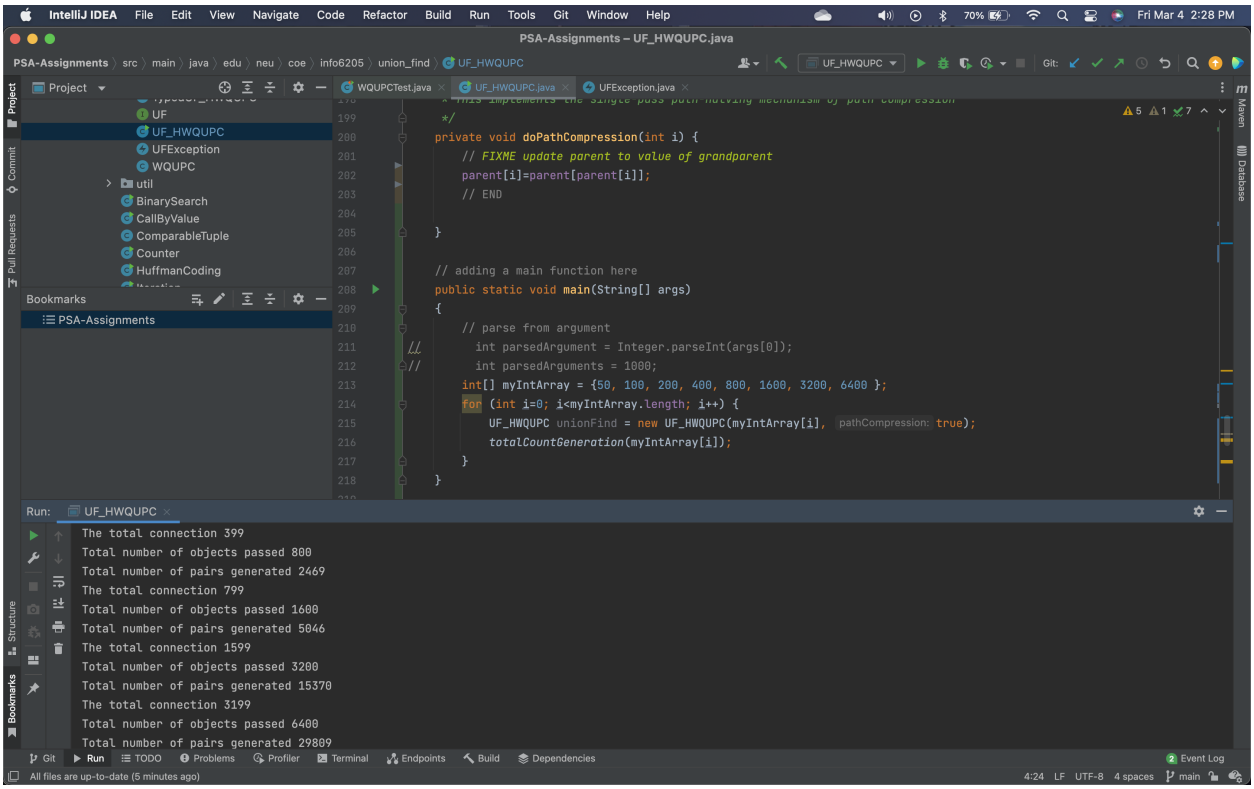
Output / Unit Test Screenshot

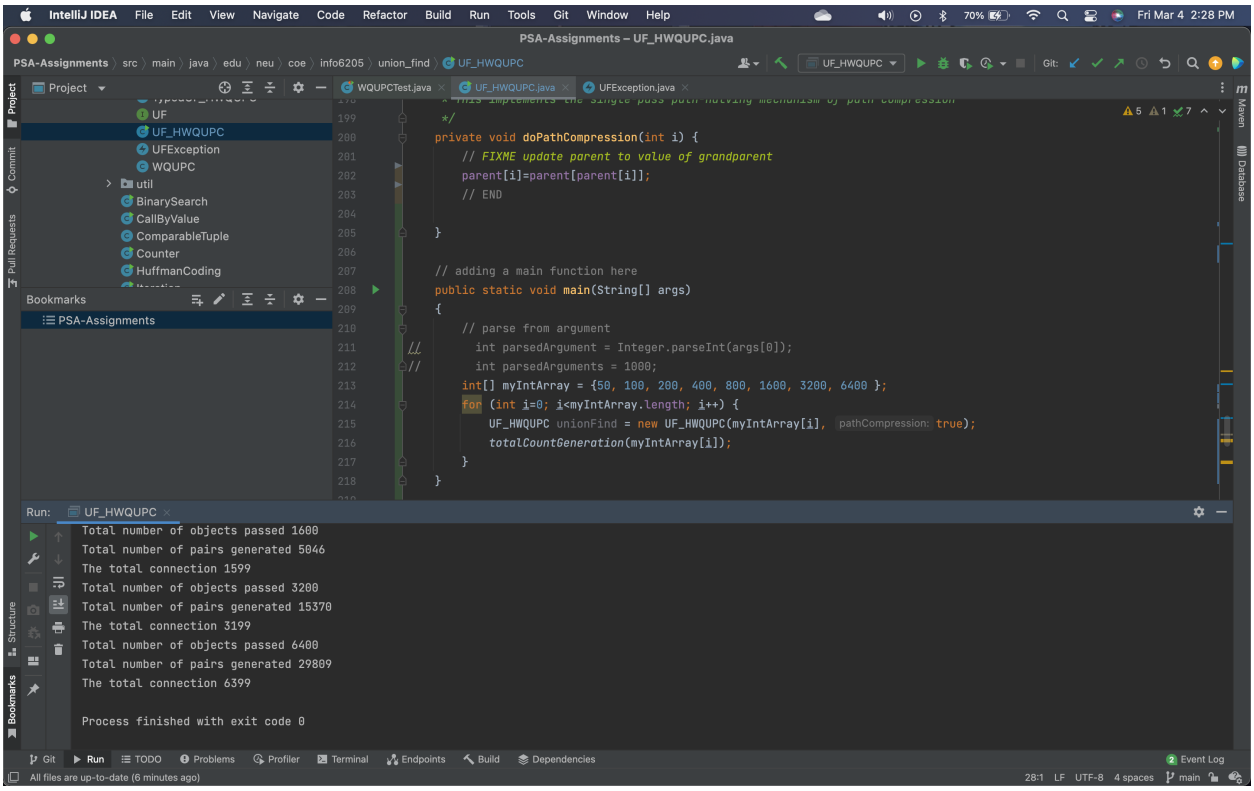
All the test cases are passing:

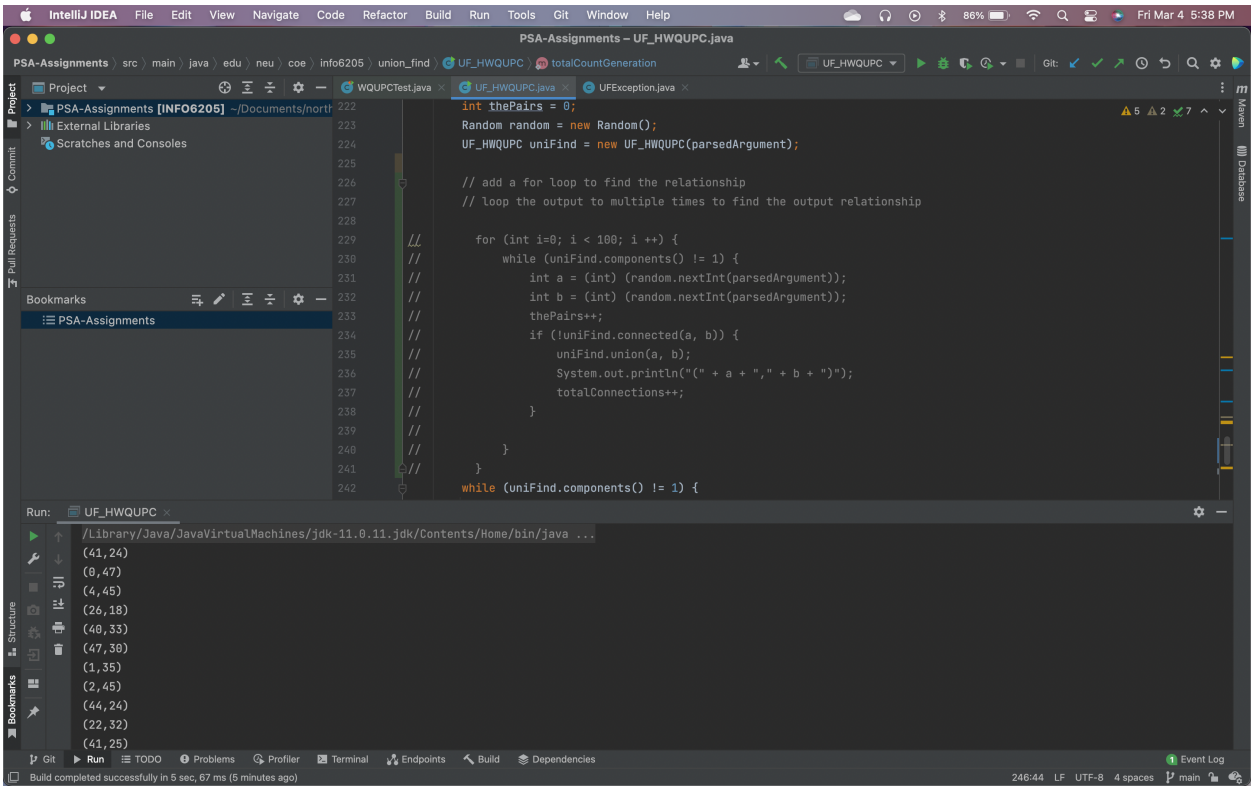


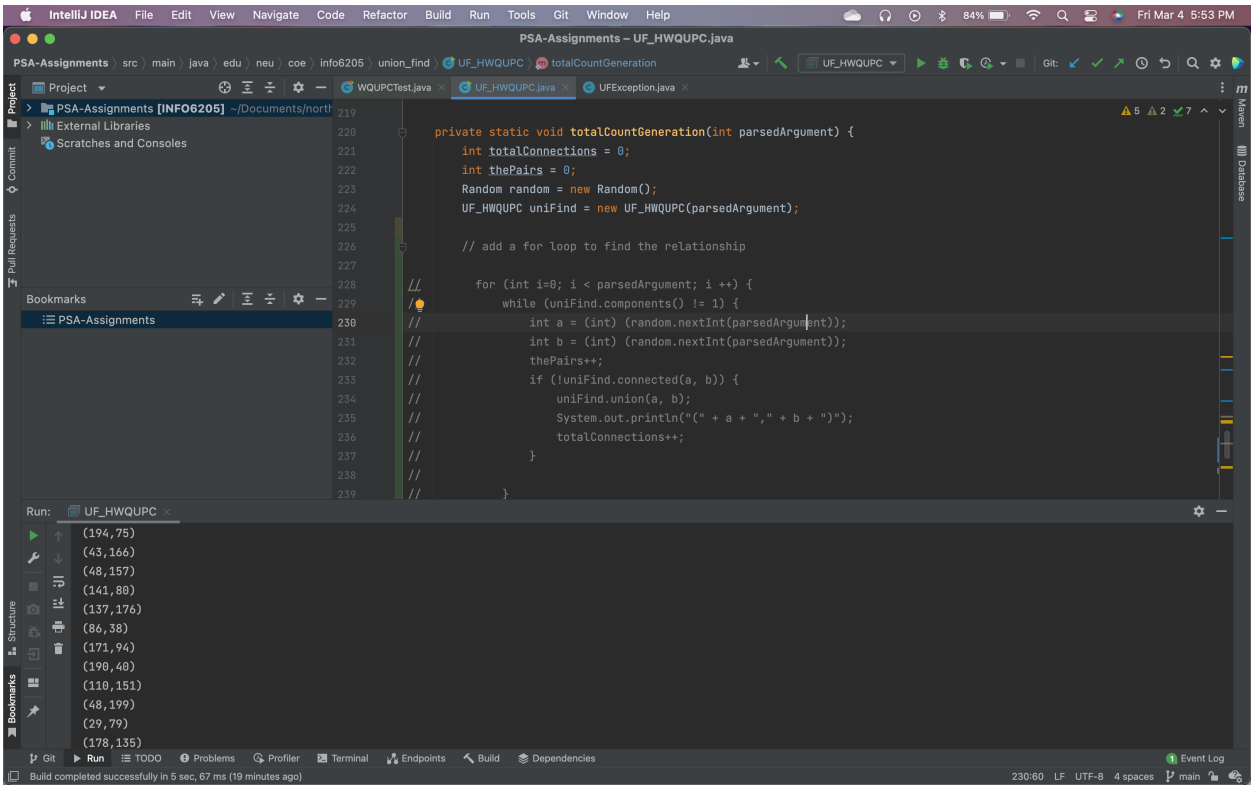
Program output


PSA-Assignments – UF_HWQUPC.java
/Library/Java/JavaVirtualMachines/jdk-11.0.11.jdk/Contents/Home/bin/java ...
Total number of objects passed 50
Total number of pairs generated 106
The total connection 49
Total number of objects passed 100
Total number of pairs generated 199
The total connection 99
Total number of objects passed 200
Total number of pairs generated 530
The total connection 199
Total number of objects passed 400
Total number of pairs generated 1152


PSA-Assignments – UF_HWQUPC.java
The total connection 399
Total number of objects passed 800
Total number of pairs generated 2469
The total connection 799
Total number of objects passed 1600
Total number of pairs generated 5046
The total connection 1599
Total number of objects passed 3200
Total number of pairs generated 15370
The total connection 3199
Total number of objects passed 6400
Total number of pairs generated 29809


PSA-Assignments – UF_HWQUPC.java
/*
*/
private void doPathCompression(int i) {
// FIXME update parent to value of grandparent
parent[i]=parent[parent[i]];
// END
}
// adding a main function here
public static void main(String[] args)
{
// parse from argument
// int parsedArgument = Integer.parseInt(args[0]);
// int parsedArguments = 1000;
int[] myIntArray = {50, 100, 200, 400, 800, 1600, 3200, 6400 };
for (int i=0; i<myIntArray.length; i++) {
UF_HWQUPC unionFind = new UF_HWQUPC(myIntArray[i], pathCompression: true);
totalCountGeneration(myIntArray[i]);
}
}
Run: UF_HWQUPC
Total number of objects passed 1600
Total number of pairs generated 5046
The total connection 1599
Total number of objects passed 3200
Total number of pairs generated 15370
The total connection 3199
Total number of objects passed 6400
Total number of pairs generated 29809
The total connection 6399
Process finished with exit code 0


PSA-Assignments – UF_HWQUPC.java
int thePairs = 0;
Random random = new Random();
UF_HWQUPC uniFind = new UF_HWQUPC(parsedArgument);
// add a for loop to find the relationship
// loop the output to multiple times to find the output relationship
// for (int i=0; i < 100; i ++) {
// while (uniFind.components() != 1) {
// int a = (int) (random.nextInt(parsedArgument));
// int b = (int) (random.nextInt(parsedArgument));
// thePairs++;
// if (!uniFind.connected(a, b)) {
// uniFind.union(a, b);
// System.out.println("(" + a + "," + b + ")");
// totalConnections++;
// }
// }
// }
while (uniFind.components() != 1) {
Run: UF_HWQUPC
/Library/Java/JavaVirtualMachines/jdk-11.0.11.jdk/Contents/Home/bin/java ...
(41,24)
(0,47)
(4,45)
(26,18)
(40,33)
(47,30)
(1,35)
(2,45)
(44,24)
(22,32)
(41,25)

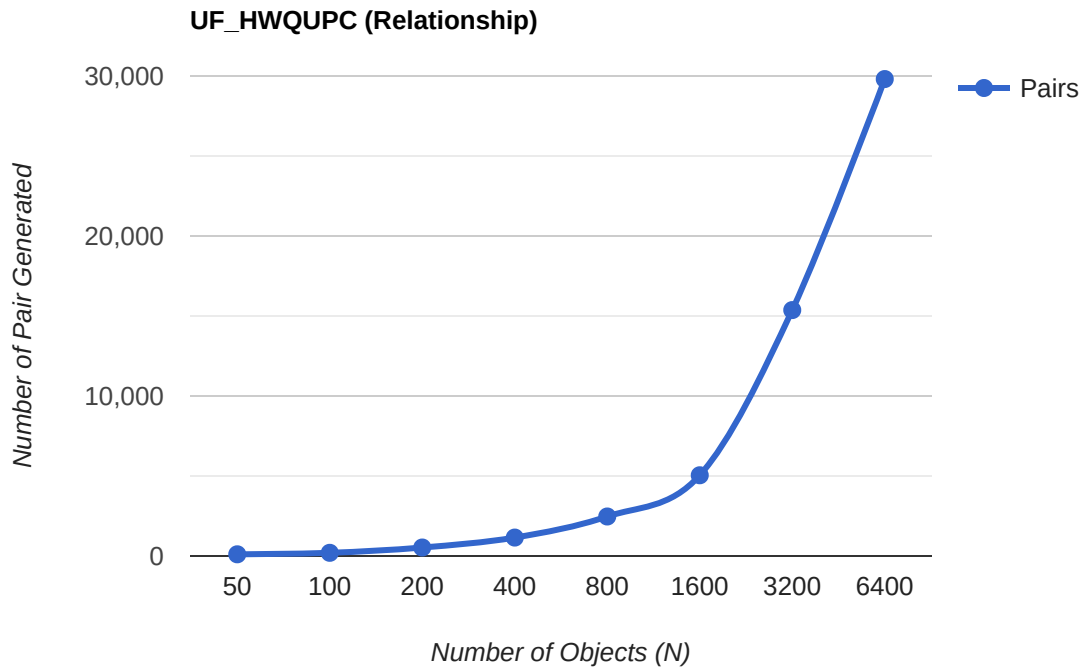


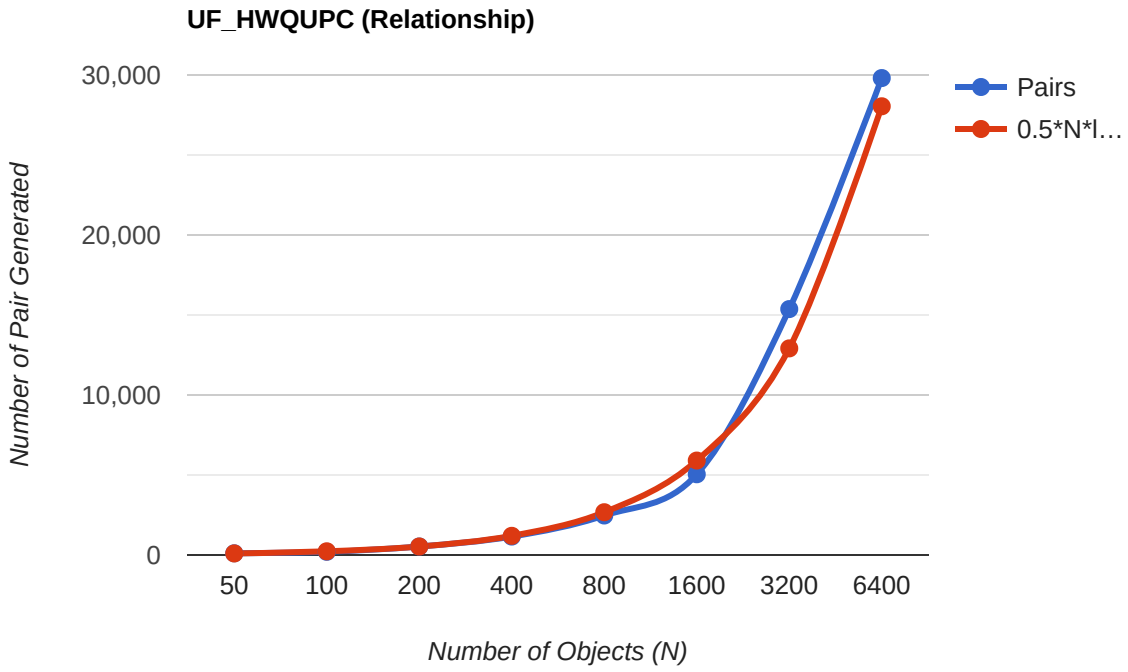
Evidences / Graphs

Number of objects passed	Number of pairs generated	Total number of connection
50	106	49
100	199	99
200	530	199
400	1152	399
800	2469	799
1600	5046	1599
3200	15370	3199
6400	29809	6399

Number of objects passed (N)	Number of pairs generated	Relationship (1/2*N*ln(N))
50	106	97.80
100	199	230.25
200	530	529.83
400	1152	1198.2929

Number of objects passed (N)	Number of pairs generated	Relationship (1/2*N*ln(N))
800	2469	2673.84
1600	5046	5902.20
3200	15370	12913.45
6400	29809	28044.97





Relationship

We can deduce 2 relationship fromt the given experiment.

- The total number of connection generated is equal to the following formula

$$connection = n - 1$$

  `Connection` : total number of connection

  `n` : number of objects passed

- From the graph we observed the relationship between the number of objects (n) & the number of pairs generated to reduce the component from n to 1 is

$$m = 1/2 * n * ln(n)$$

  `m` : number of pair used to generate the component

  `n` : number of objects passed

  We can conclude the above relationship is valid by plotting the graph which we see in the evidence section above.