Program Structures and Algorithms
Fall 2022(SEC 06)

NAME: Mahathi Siddavatam
NUID: 002198134

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.

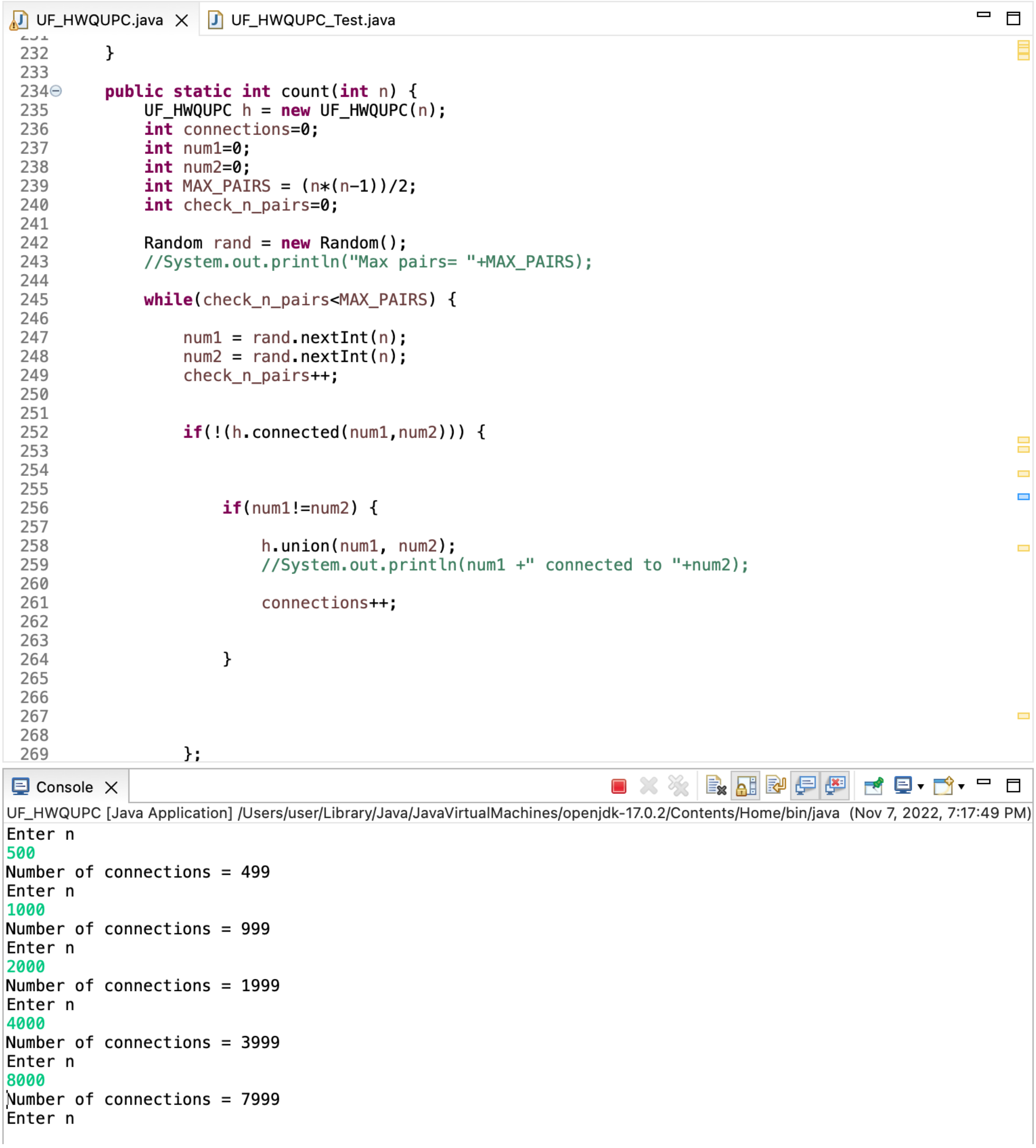
NOTE: although I'm not going to tell you in advance what the relationship is, I can assure you that it is a *simple* relationship.

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

Relationship Conclusion: The relationship between the number of objects (n) and number of pairs generated is:

number of pairs generated = n-1. This is because of the transitive nature of the connections i.e. when 1 is connected to 2 and 2 is connected to 3 then 3 is also connected to 1 although there are only 2 connections. This conclusion was observed after going through 5 large input sizes and getting the same relationship result.

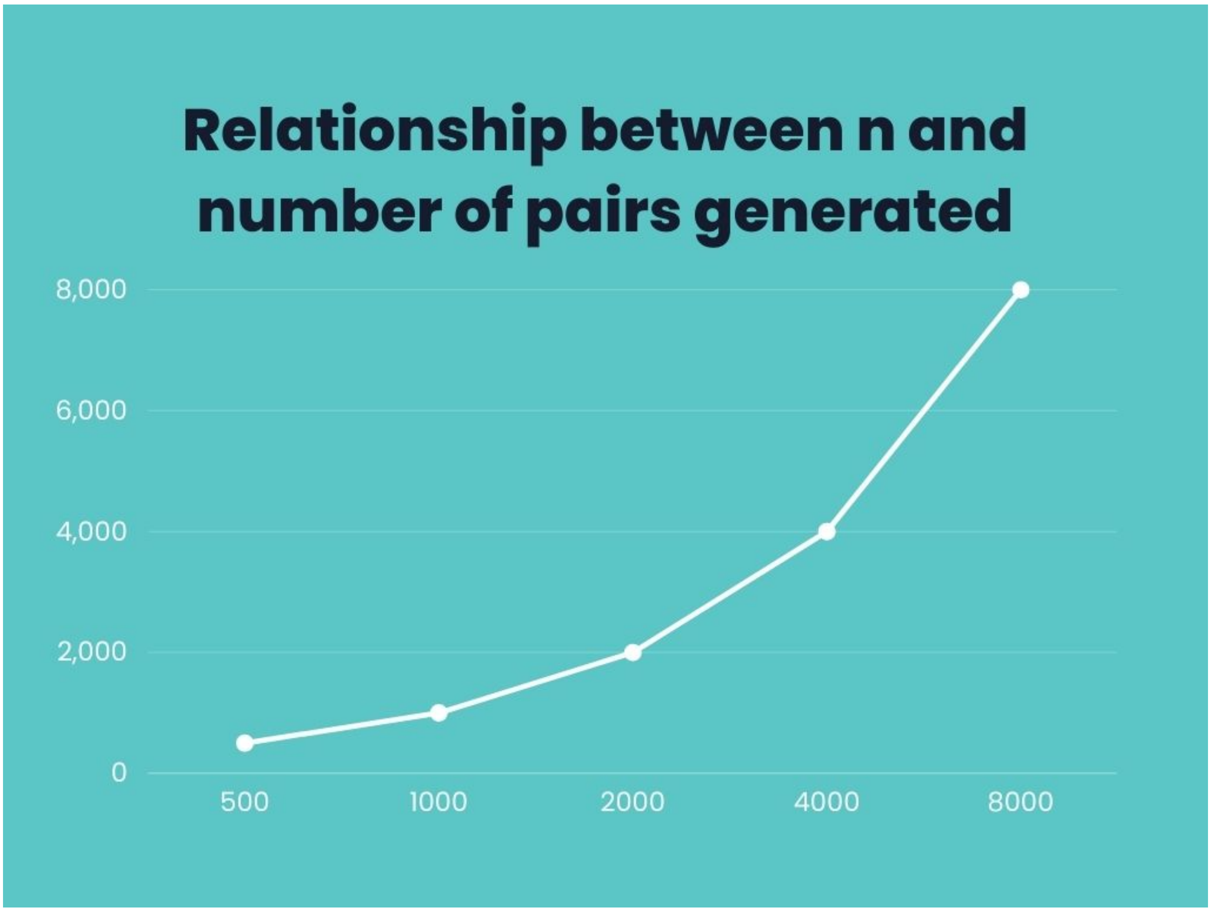
Evidence to support that conclusion:



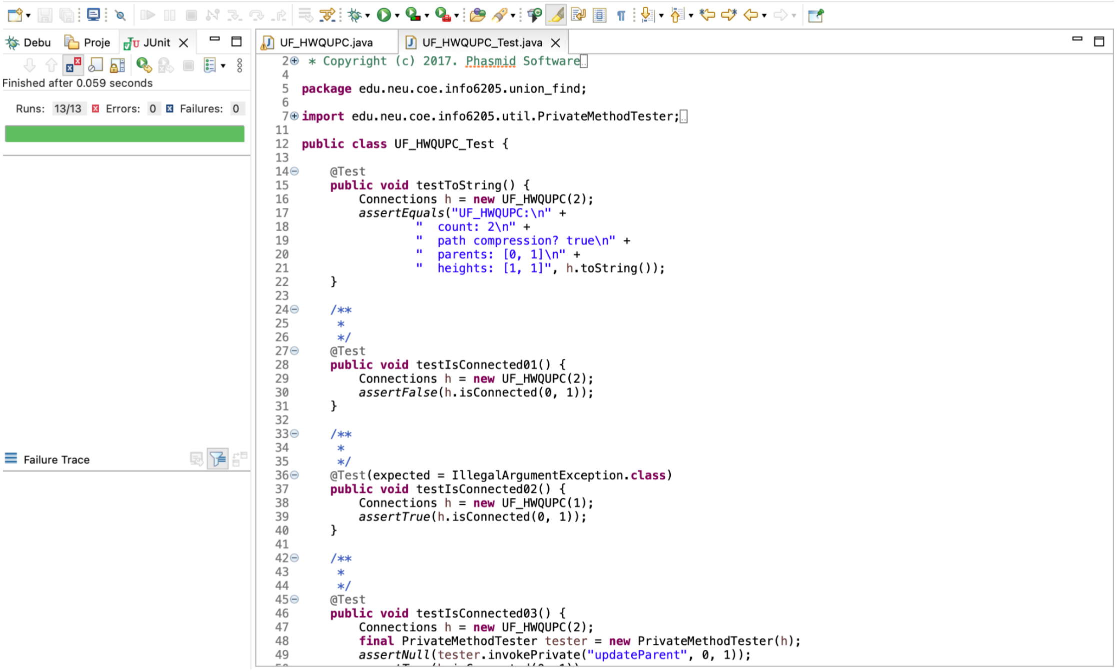
```java
    }

    public static int count(int n) {
        UF_HWQUPC h = new UF_HWQUPC(n);
        int connections=0;
        int num1=0;
        int num2=0;
        int MAX_PAIRS = (n*(n-1))/2;
        int check_n_pairs=0;

        Random rand = new Random();
        //System.out.println("Max pairs= "+MAX_PAIRS);

        while(check_n_pairs<MAX_PAIRS) {

            num1 = rand.nextInt(n);
            num2 = rand.nextInt(n);
            check_n_pairs++;


            if(!(h.connected(num1,num2))) {



                if(num1!=num2) {

                    h.union(num1, num2);
                    //System.out.println(num1 +" connected to "+num2);

                    connections++;



                }





            };
```

```
UF_HWQUPC [Java Application] /Users/user/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java  (Nov 7, 2022, 7:17:49 PM)
Enter n
500
Number of connections = 499
Enter n
1000
Number of connections = 999
Enter n
2000
Number of connections = 1999
Enter n
4000
Number of connections = 3999
Enter n
8000
Number of connections = 7999
Enter n
```

Graphical Representation:



Unit Test Screenshots:



Finished after 0.059 seconds

Runs: 13/13 Errors: 0 Failures: 0

```java
 * Copyright (c) 2017. Phasmid Software

package edu.neu.coe.info6205.union_find;

import edu.neu.coe.info6205.util.PrivateMethodTester;

public class UF_HWQUPC_Test {

    @Test
    public void testToString() {
        Connections h = new UF_HWQUPC(2);
        assertEquals("UF_HWQUPC:\n" +
                "  count: 2\n" +
                "  path compression? true\n" +
                "  parents: [0, 1]\n" +
                "  heights: [1, 1]", h.toString());
    }

    /**
     *
     */
    @Test
    public void testIsConnected01() {
        Connections h = new UF_HWQUPC(2);
        assertFalse(h.isConnected(0, 1));
    }

    /**
     *
     */
    @Test(expected = IllegalArgumentException.class)
    public void testIsConnected02() {
        Connections h = new UF_HWQUPC(1);
        assertTrue(h.isConnected(0, 1));
    }

    /**
     *
     */
    @Test
    public void testIsConnected03() {
        Connections h = new UF_HWQUPC(2);
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertNull(tester.invokePrivate("updateParent", 0, 1));
```