Assignment 3 (WQUPC)

Name : Ganesh Dharani

Nuid : 002191922

Step 1:
(a) Implement height-weighted Quick Union with Path Compression.

Done the code and ran all the test cases for height weighted quick union with path compression.

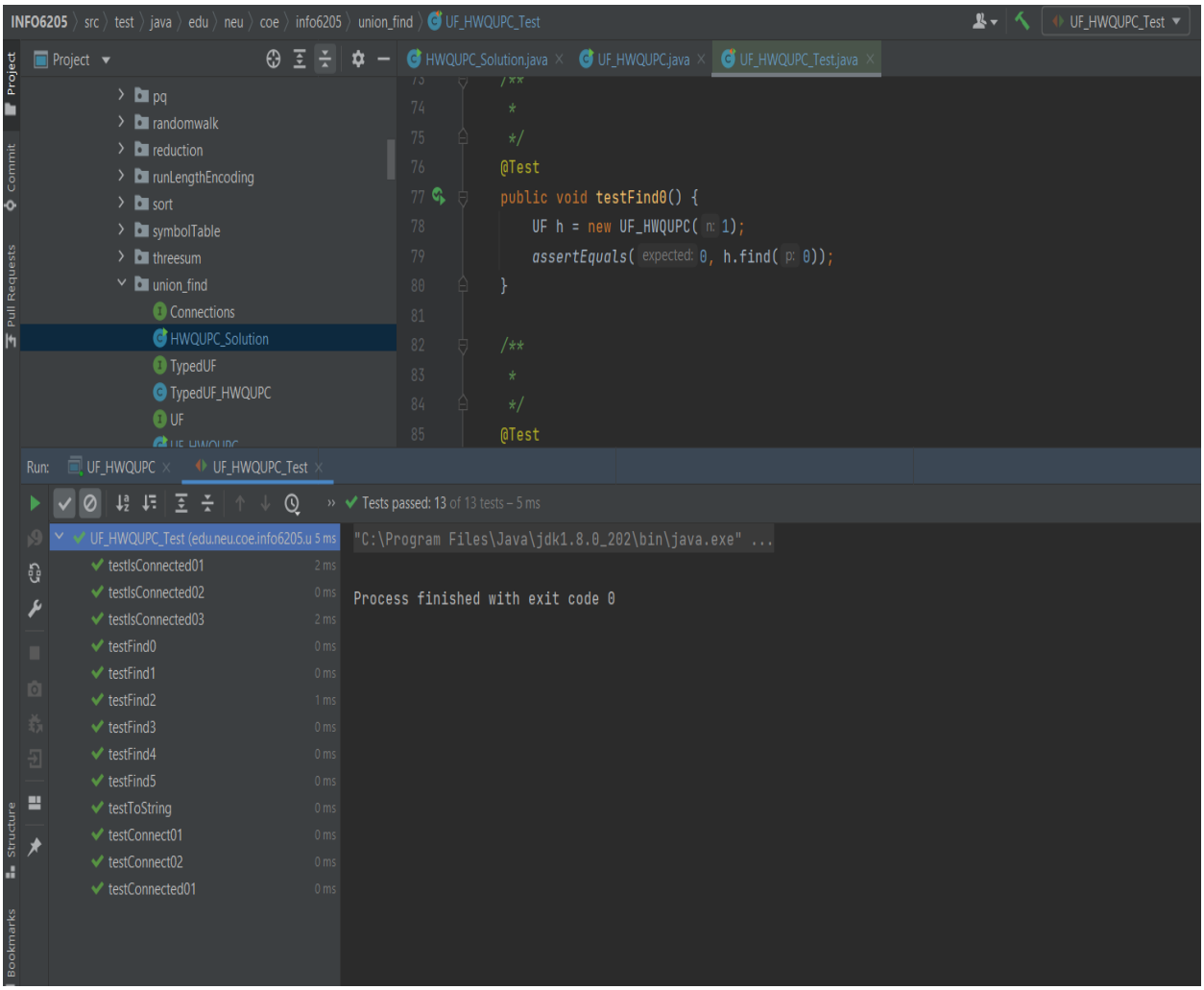
```java
    private void mergeComponents(int i, int j) {
        // FIXME make shorter root point to taller one
        if(i==j) return;
        if(height[i] < height[j]){
            updateHeight(j,i);
            updateParent(i,j);}

        else if(height[i] > height[j])    {
            updateHeight(i,j);
            updateParent(j,i);
        }
        else{
            updateParent(j,i);
            height[i]++;
        }
    }
        // END
```

```java
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        while (root != parent[root]) {
            doPathCompression(root);
            root = parent[root];
        }
        // END
        return root;
    }
```

(b) Check that the unit tests for this class all work

The test cases have been passed successfully as mentioned below:



Step 2:

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.

Done the code and created count() which takes n as the argument and returns the number of connections. And prints the returned value.

HWQUPC_Solution.java | UF_HWQUPC.java | UF_HWQUPC_Test.java

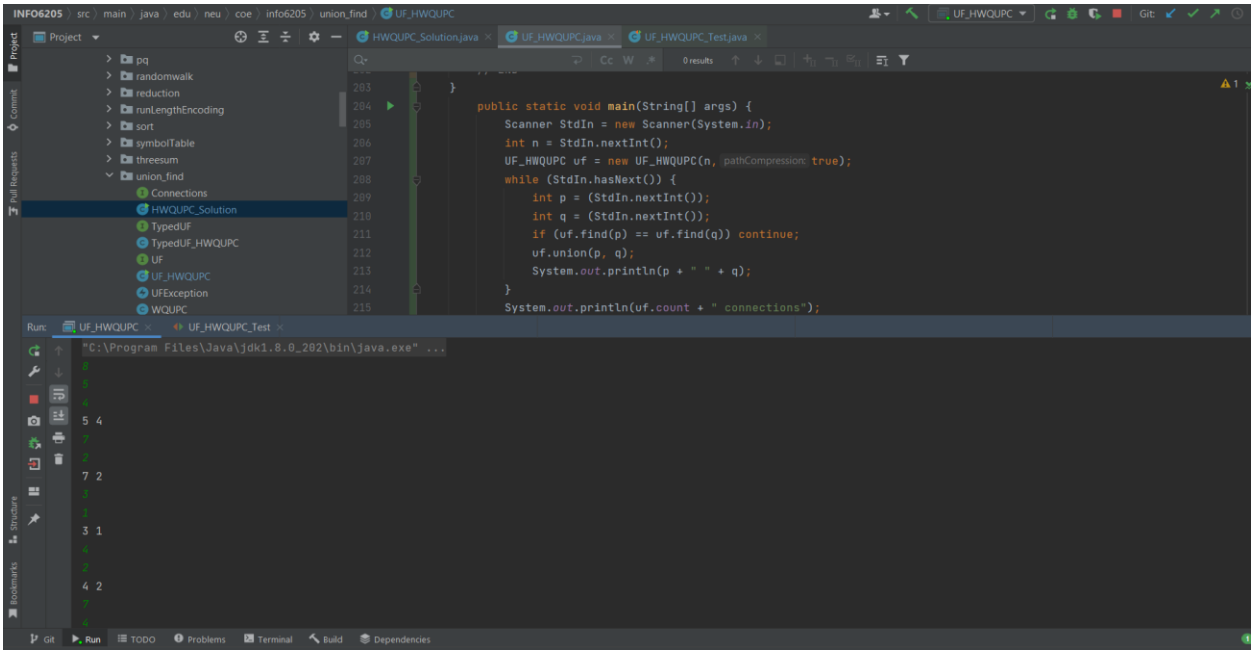
```java
package edu.neu.coe.info6205.union_find;
import edu.neu.coe.info6205.graphs.BFS_and_prims.StdRandom;

public class HWQUPC_Solution {
    public static int count(int n) {

        int edges = 0;
        UF_HWQUPC uf = new UF_HWQUPC(n, pathCompression: true);
        while (uf.count > 1) {
            int i = StdRandom.uniform(n);
            int j = StdRandom.uniform(n);
            if (uf.find(i) != uf.find(j)) uf.union(i, j);
            edges++;
        }
        return edges;
    }

```

```java
    public static void main(String[] args) {
        int n = 10000;
        int trials = 1000;
        int[] edges = new int[trials];
        // repeat the experiment
        for (int t = 0; t < trials; t++) {
            edges[t] = count(n);
        }
        System.out.println("Count of objects = " + n);
        System.out.println("No of pairs generated   = " + mean(edges));
    }
}
```

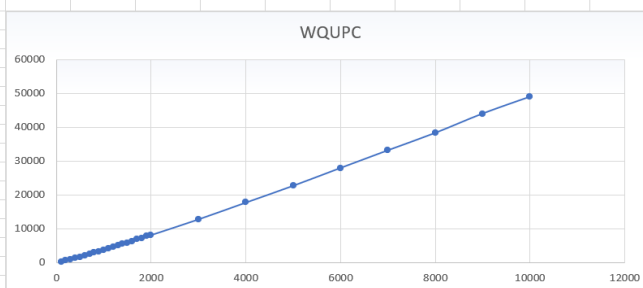


Step 3:

Determine the relationship between the number of objects (*n*) and the number of pairs (*m*).

For the number of objects (n) and the number of pairs(m) the relationship has been generated as shown in the below graph.

Count of objects (N)	No pairs generated (M)
100	261.213
200	589.162
300	945.415
400	1313.207
500	1696.825
600	2094.872
700	2494.24
800	2922.708
900	3320.477
1000	3739.289
1100	4154.713
1200	4627.447
1300	5023.695
1400	5523.603
1500	5890.488
1600	6337.873
1700	6851.312
1800	7286.838
1900	7759.416
2000	8188.159
3000	12819.984
4000	17756.894
5000	22710.204
6000	27965.156
7000	33166.017
8000	38283.203
9000	43966.438
10000	48936.924



Conclusion:

With the derived graph with the plotting points. The final conclusion regarding the relation is as getting as below.

Number of Pairs = (N*logN)/2, where N=Number of Objects