Program Structures and Algorithms Spring 2022
Assignment 3: Union Find

Tushar Kurhekar
NUID - 001521707

Task to Accomplish in Assignment 3:

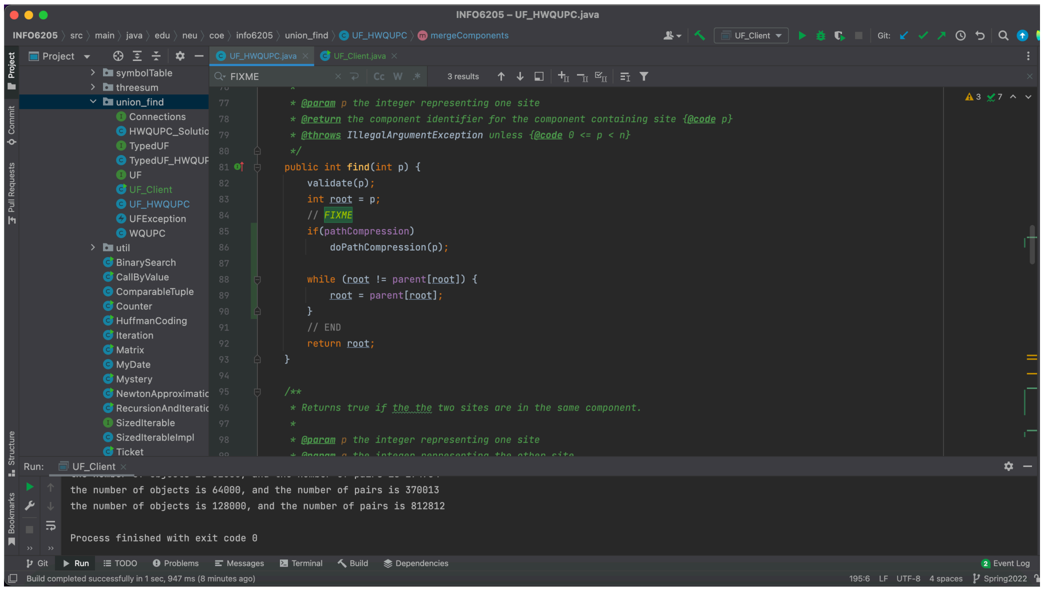
I. Implement height-weighted Quick Union with Path Compression. Check all unit test cases.

II. 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.

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

Step 1:

Changes in find() method:

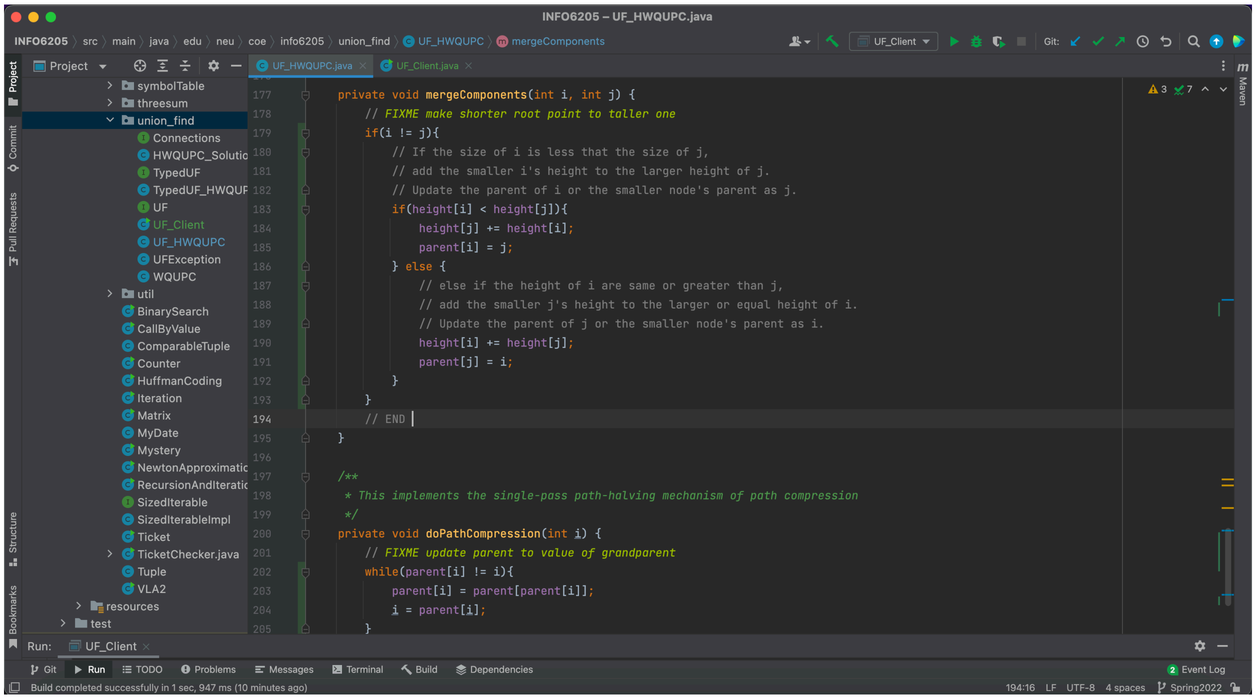

INFO6205 – UF_HWQUPC.java
INFO6205 › src › main › java › edu › neu › coe › info6205 › union_find › UF_HWQUPC › mergeComponents
UF_HWQUPC.java
UF_Client.java
FIXME
3 results
Run: UF_Client
Git Run TODO Problems Messages Terminal Build Dependencies

```java
     * @param p the integer representing one site
     * @return the component identifier for the component containing site {@code p}
     * @throws IllegalArgumentException unless {@code 0 <= p < n}
     */
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        if(pathCompression)
            doPathCompression(p);

        while (root != parent[root]) {
            root = parent[root];
        }
        // END
        return root;
    }

    /**
     * Returns true if the the two sites are in the same component.
     *
     * @param p the integer representing one site
```

```
the number of objects is 64000, and the number of pairs is 370013
the number of objects is 128000, and the number of pairs is 812812

Process finished with exit code 0
```

Build completed successfully in 1 sec, 947 ms (8 minutes ago)

Changes in mergeComponents() and doPathCompression()

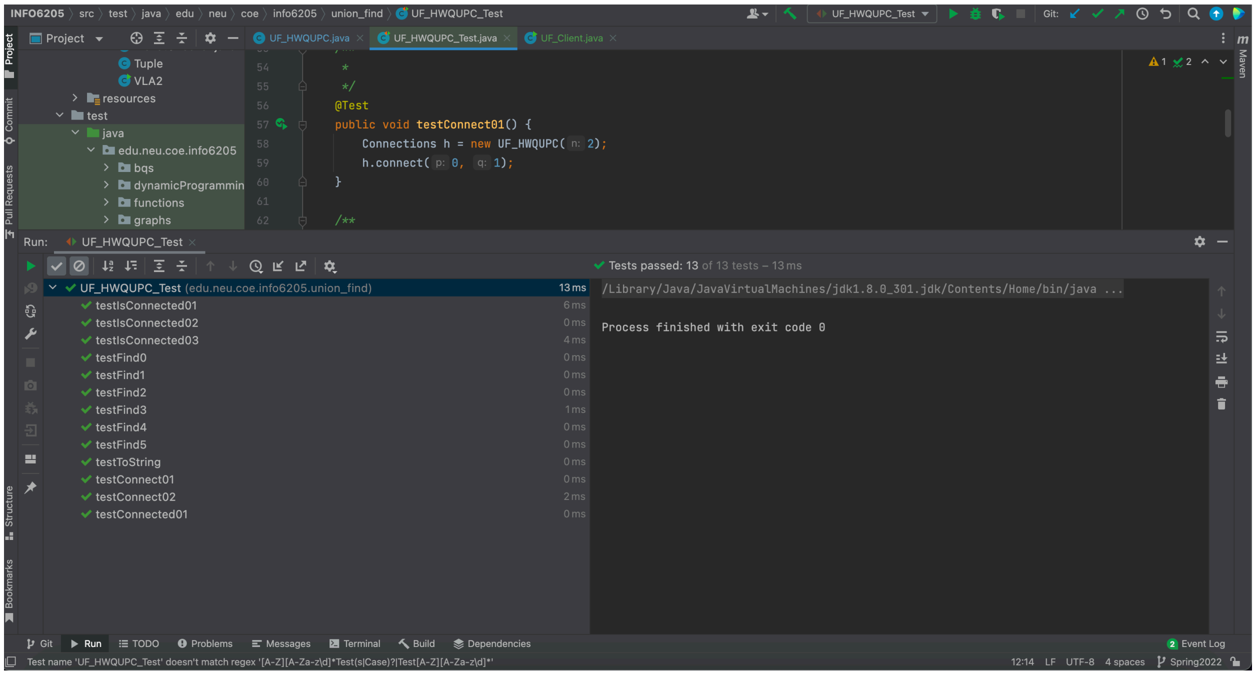


INFO6205 – UF_HWQUPC.java

```java
    private void mergeComponents(int i, int j) {
        // FIXME make shorter root point to taller one
        if(i != j){
            // If the size of i is less than the size of j,
            // add the smaller i's height to the larger height of j.
            // Update the parent of i or the smaller node's parent as j.
            if(height[i] < height[j]){
                height[j] += height[i];
                parent[i] = j;
            } else {
                // else if the height of i are same or greater than j,
                // add the smaller j's height to the larger or equal height of i.
                // Update the parent of j or the smaller node's parent as i.
                height[i] += height[j];
                parent[j] = i;
            }
        }
        // END
    }

    /**
     * This implements the single-pass path-halving mechanism of path compression
     */
    private void doPathCompression(int i) {
        // FIXME update parent to value of grandparent
        while(parent[i] != i){
            parent[i] = parent[parent[i]];
            i = parent[i];
        }
```

```
Build completed successfully in 1 sec, 947 ms (10 minutes ago)
```

Unit Test Cases Passed: UF_HWQUPC.java:



```java
     */
    @Test
    public void testConnect01() {
        Connections h = new UF_HWQUPC( n: 2);
        h.connect( p: 0,  q: 1);
    }

    /**
```

Tests passed: 13 of 13 tests – 13 ms

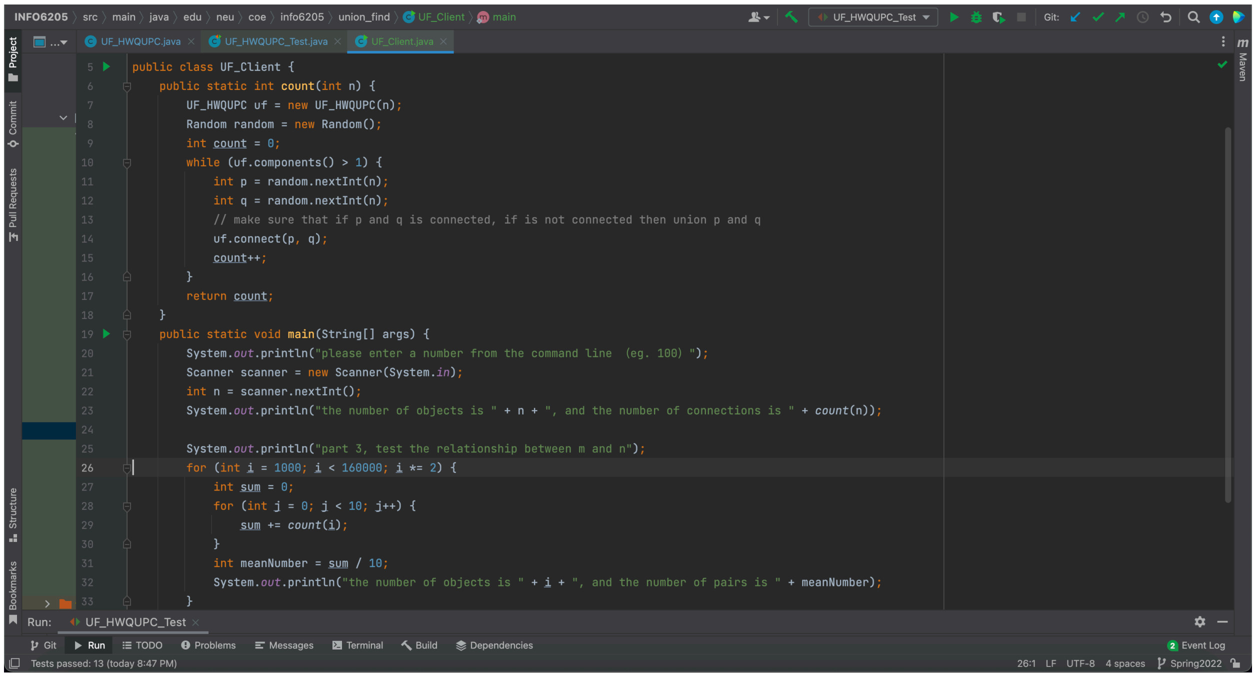
```
UF_HWQUPC_Test (edu.neu.coe.info6205.union_find)   13 ms
    testIsConnected01    6 ms
    testIsConnected02    0 ms
    testIsConnected03    4 ms
    testFind0            0 ms
    testFind1            0 ms
    testFind2            0 ms
    testFind3            1 ms
    testFind4            0 ms
    testFind5            0 ms
    testToString         0 ms
    testConnect01        0 ms
    testConnect02        2 ms
    testConnected01      0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

```
Test name 'UF_HWQUPC_Test' doesn't match regex '[A-Z][A-Za-z\d]*Test(s|Case)?|Test[A-Z][A-Za-z\d]*'
```

Step 1:

Implemented UF_Client.java class:



```java
public class UF_Client {
    public static int count(int n) {
        UF_HWQUPC uf = new UF_HWQUPC(n);
        Random random = new Random();
        int count = 0;
        while (uf.components() > 1) {
            int p = random.nextInt(n);
            int q = random.nextInt(n);
            // make sure that if p and q is connected, if is not connected then union p and q
            uf.connect(p, q);
            count++;
        }
        return count;
    }
    public static void main(String[] args) {
        System.out.println("please enter a number from the command line  (eg. 100) ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.println("the number of objects is " + n + ", and the number of connections is " + count(n));

        System.out.println("part 3, test the relationship between m and n");
        for (int i = 1000; i < 160000; i *= 2) {
            int sum = 0;
            for (int j = 0; j < 10; j++) {
                sum += count(i);
            }
            int meanNumber = sum / 10;
            System.out.println("the number of objects is " + i + ", and the number of pairs is " + meanNumber);
        }
```

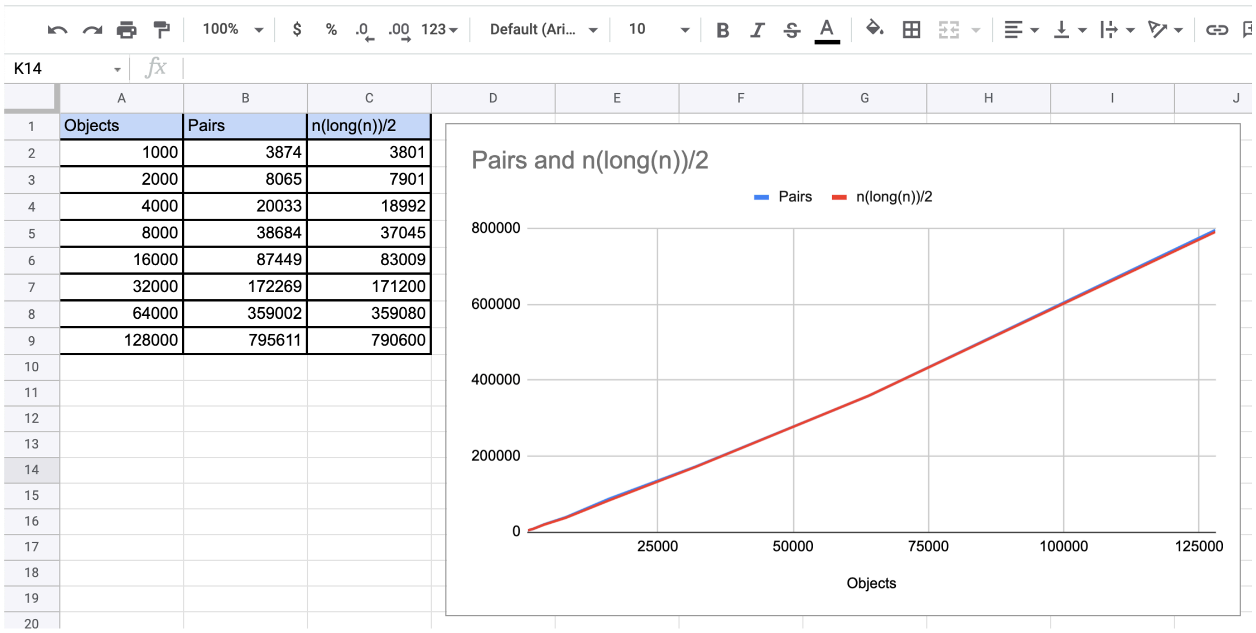
```
Tests passed: 13 (today 8:47 PM)
```

Relation Analysis and Graph Reports

Running UF_Client with the main() function I ran it for 150 and 250 as a value of m

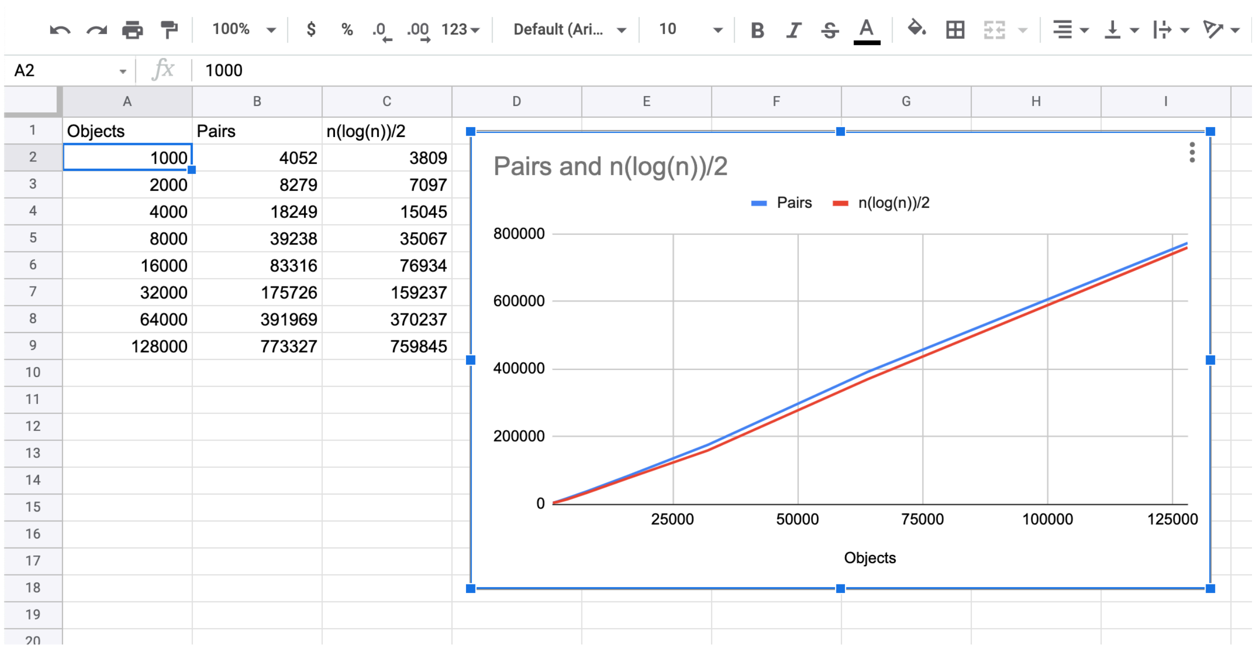
I have attached the result below

For 150 :



Objects	Pairs	n(long(n))/2
1000	3874	3801
2000	8065	7901
4000	20033	18992
8000	38684	37045
16000	87449	83009
32000	172269	171200
64000	359002	359080
128000	795611	790600

For 250 :



Objects	Pairs	n(log(n))/2
1000	4052	3809
2000	8279	7097
4000	18249	15045
8000	39238	35067
16000	83316	76934
32000	175726	159237
64000	391969	370237
128000	773327	759845

In the above graph, we can see the relation with m to n is almost linear but its more of an n log (n) for weighted quick union and n log n /2 for weighted quick union with path compression

Hence we can deduce the relation of m with n

$$m=n \log(n)$$

It also can be **m=n log(n)/2** if path compression is considered where coefficient won't make much of the changes in the values.

Number of Objects: n

From the above observations we came to the following conclusion:

Both the lines are approximately the same with n log(n). We can derive the relationship between m and n as given above.