Assignment No 3: Random Walk Experiment

Name: Ramya Kanguri
NUID: 002133068

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).

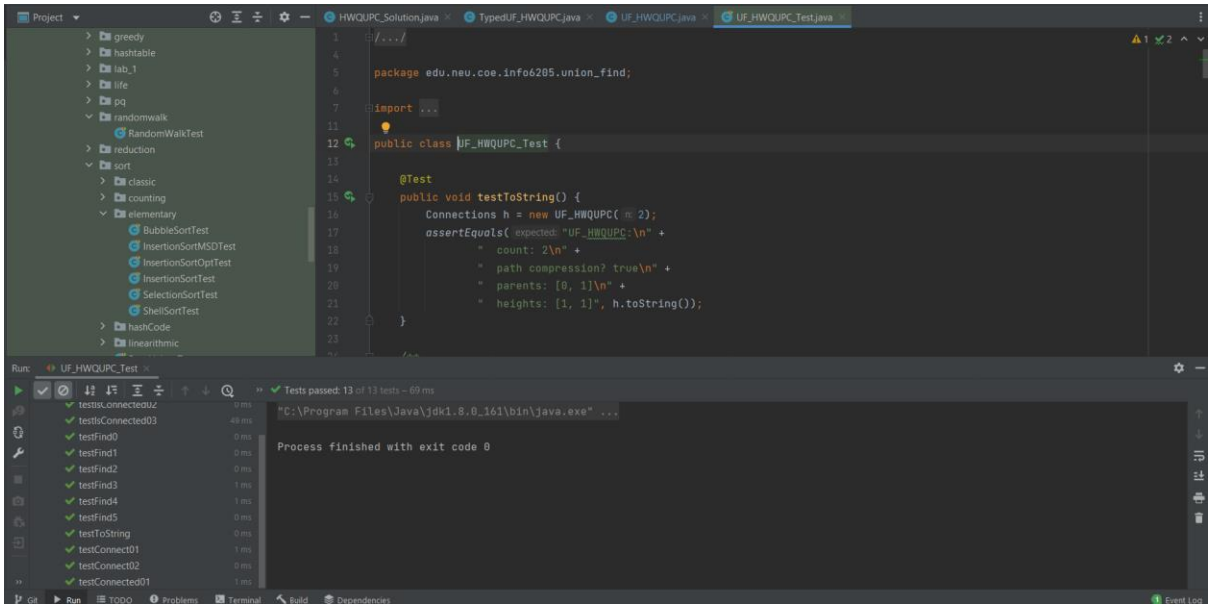
Step 3:
Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion in terms of your observations and what you think might be going on.

1. Implementation and running all the test cases:

```
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;
}
```

```java
private void mergeComponents(int i, int j) {
    // FIXME make shorter root point to taller one
    if(i != j){
        if(height[i] < height[j]){
            height[j] += height[i];
            parent[i] = j;
        } else {
            height[i] += height[j];
            parent[j] = i;
        }
    }
    // END
}
```

```java
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    while(parent[i] != i){
        parent[i] = parent[parent[i]];
        i = parent[i];
    }
    // END
}
```



2. Implemented UFClinet.java Class:

```java
package edu.neu.coe.info6205.union_find;

import java.util.Random;
import java.util.Scanner;

public class UFClient {
    public static int count(int n) {
        UF_HWQUPC unifind = new UF_HWQUPC(n);
        Random random = new Random();
        int count = 0;
        while (unifind.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
            unifind.connect(p, q);
            count++;
        }
        return count;
    }

    public static void main(String[] args) {
        // step 2 : input a number from the command line (eg. 100)
        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");
        // step 3 : determine the relationship between n and m
        for (int i = 1000; i < 160000; i *= 2) {
            int sum = 0;
            // test 10 times to get the average number
```

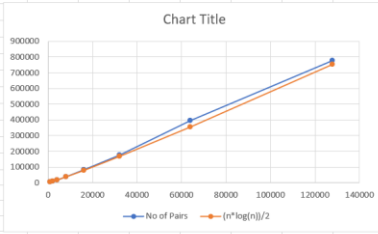
```java
    public static void main(String[] args) {
        // step 2 : input a number from the command line (eg. 100)
        System.out.println("please enter a number from the command line (eg. 100) ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.println("the 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");
        // step 3 : determine the relationship between n and m
        for (int i = 1000; i < 160000; i *= 2) {
            int sum = 0;
            // test 10 times to get the average number
            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);
        }
    }
}
```

3. Determining the relationship between the number of objects and the number of pairs:

Input a number after running the UFClient.java class from the command line to test the count method. We can test multiple values of n and make their values bigger using doubling method, each with 10 times to test the relationship between m and n. We mapped the 2 outputs using above method. First passed the input from command line as 80 and then passed the input from command line as 400. Output of both was mapped in a graphical format.

Input: 80

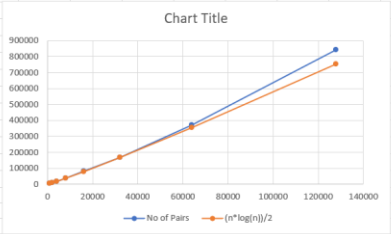
No of Objects	No of Pairs	(n*log(n))/2
1000	3621	3454
2000	8556	7601
4000	18068	16588
8000	38431	35949
16000	82354	77443
32000	175771	165976
64000	393585	354132
128000	777271	752626




```
part 3, test the relationship between m and n
the number of objects is 1000, and the number of pairs is 3621
the number of objects is 2000, and the number of pairs is 8556
the number of objects is 4000, and the number of pairs is 18068
the number of objects is 8000, and the number of pairs is 38431
the number of objects is 16000, and the number of pairs is 82354
the number of objects is 32000, and the number of pairs is 175771
the number of objects is 64000, and the number of pairs is 393585
the number of objects is 128000, and the number of pairs is 777271

Process finished with exit code 0
```

Input: 400

No of Objects	No of Pairs	(n*log(n))/2
1000	3777	3454
2000	8190	7601
4000	19145	16588
8000	38125	35949
16000	80662	77443
32000	167411	165976
64000	369058	354132
128000	842438	752626




```
part 3, test the relationship between m and n
the number of objects is 1000, and the number of pairs is 3777
the number of objects is 2000, and the number of pairs is 8190
the number of objects is 4000, and the number of pairs is 19145
the number of objects is 8000, and the number of pairs is 38125
the number of objects is 16000, and the number of pairs is 80662
the number of objects is 32000, and the number of pairs is 167411
the number of objects is 64000, and the number of pairs is 369058
the number of objects is 128000, and the number of pairs is 842438

Process finished with exit code 0
```

In the above graph X-axis shows the no of objects (n), blue line represents No of pairs for respective no of objects(n), orange line shows the relationship output respective to no of objects(n)

Conclusion: From the above observations we came to following conclusion: Both the lines are approximately same. We can derive the relationship between m and n as below: m = ½ (n log(n))