Assignment 4 (WQUPC)

1. Intro

In this question, I was asked to finish three tasks. There are

> - Implement height-weighted Quick Union with Path Compression (We already did it once in quiz)
> - Write a UF client class to connect a specified number of nodes. Deduce the relationship between the number of objects (*n*) and the number of pairs (*m*)

2. Implement

2.1 UF_HWQUPC

```
    public int find(int p) {
        validate(p);
        int root = p;
        //Find the ancestor
        while (root != parent[root]) {
            if (this.pathCompression) {
                doPathCompression(root);
            }
            root = parent[root];
        }
        // END
        return root;
    }
    private void mergeComponents(int i, int j) {
        if (height[i] >= height[j]) {
            updateParent(j, i);
            updateHeight(i, j);
        } else {
            updateParent(i, j);
            updateHeight(j, i);
        }
        // END
    }

    private void doPathCompression(int i) {
        setPathCompression(false);
        int root = find(i);
        setPathCompression(true);
        int temp = parent[i];
        while (parent[i] != root) {
            parent[i] = root;
            i = temp;
        }
        // END
    }
```

- **find(int p)**: We can implement this method recursively or use while loop. In this case, we should do the loop continuely until we find the ancestor of p.
- **mergeComponents(int i, int j)**: In order to get O(N log N) when we covert this to a tree. It is necessary for us to balance tree by linking root of smaller tree to root of larger tree.
- **doPathCompression(int i):** In order to save time we need to do the path compression. After we find the root of the **p**, we should set the prnt[] of each examined node to point to that root.

2.2 UnionFind Client

```
private static final Random random = new Random();
    private static int count(int n) {
        UF_HWQUPC uf = new UF_HWQUPC(n, false);
        int connectionCnt = 0;     //How many times have taken in order to connect n nodes which have different roots
        int num1, num2;
        while (uf.components() > 1) {
            num1 = random.nextInt(n);
            num2 = random.nextInt(n);
            uf.connect(num1, num2);
            connectionCnt++;
        }
        return connectionCnt;
    }
```

I use Random.class to generate numbers between 0 and n-1. Besides, I connect n nodes which have different roots. Then, figure out the relationship between the number of objects (*n*) and the number of pairs (*m*).

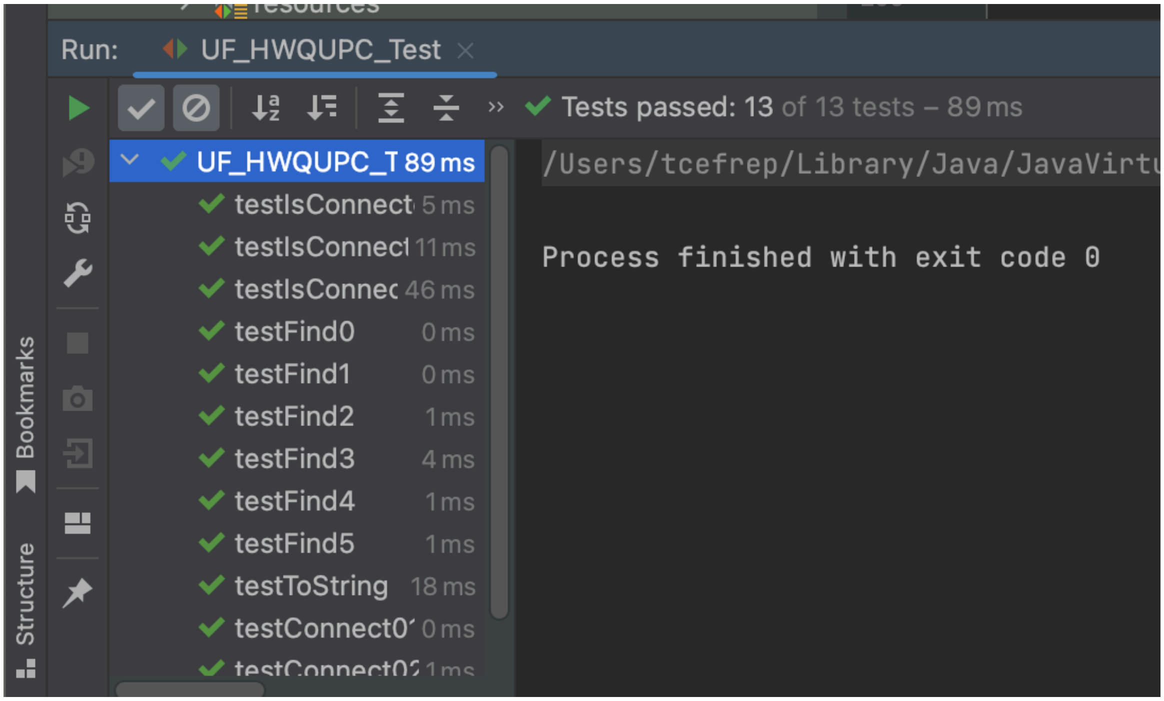
2.3 Benchmarks

```
    private static int getMeanConnections(int n) {
        int totalConnectionCnt = 0;
        int times = 100;
        for (int i = 0; i < times; i++) {
            totalConnectionCnt += UnionFindClient.count(n);
        }
        return totalConnectionCnt / times;
    }
    private static int calculate(int num) {
        return (int) (0.5 * num * Math.log(num));
    }

    public static void main(String[] args) {
//          int n = getInput();
        System.out.println("  N: " + "The number of connection: " + "  1/2Nln(N): ");
        for (int i = 100; i <= 10000; i += 100) {
            System.out.printf("%4d%14d%22d", i, getMeanConnections(i), calculate(i));
            System.out.println();
        }
    }
```

I used 6 various lengh of n to run the test. Each time, every method will run 100 times. The length of array will start at 1000, and it will increase 100 by each time.

4. Evidence

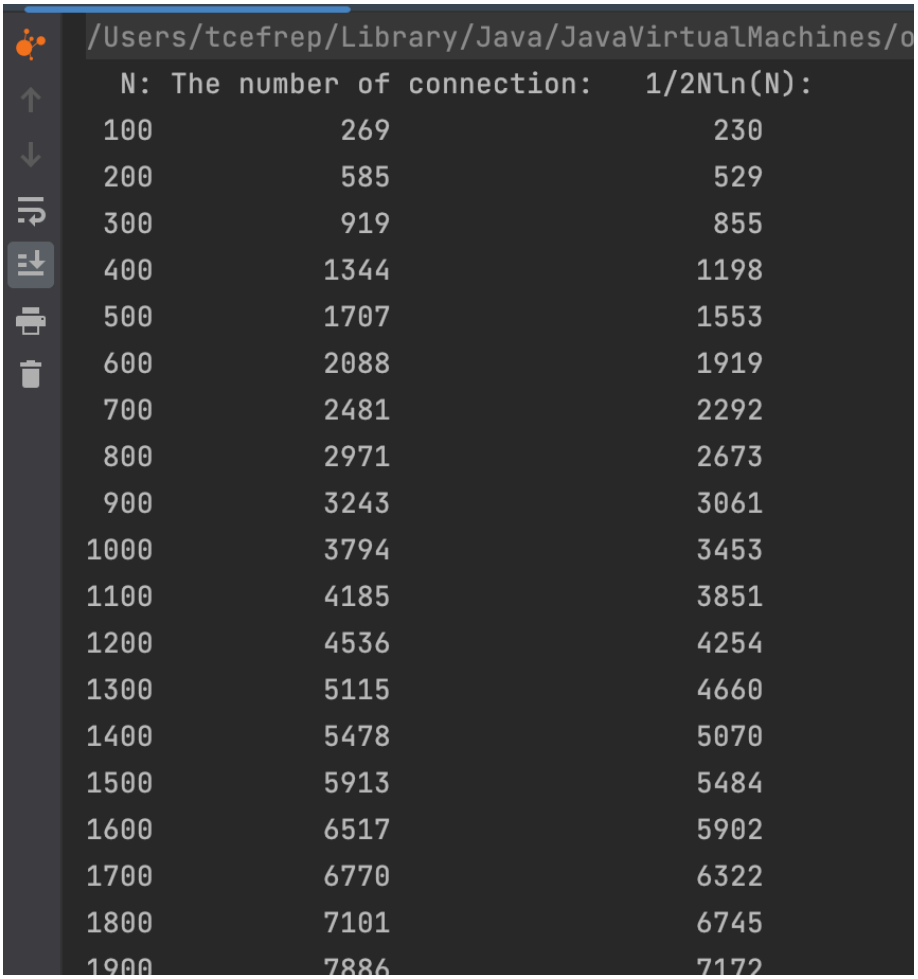
4.1 UF_HWQUPC_Test



```
Tests passed: 13 of 13 tests – 89 ms

UF_HWQUPC_T 89 ms
    testIsConnect 5 ms
    testIsConnect 11 ms
    testIsConnec 46 ms
    testFind0      0 ms
    testFind1      0 ms
    testFind2      1 ms
    testFind3      4 ms
    testFind4      1 ms
    testFind5      1 ms
    testToString  18 ms
    testConnect0' 0 ms
    testConnect0: 1 ms

/Users/tcefrep/Library/Java/JavaVirtu

Process finished with exit code 0
```

Figure 1: Sceenshots of UF HWQUPC Test

4.2 UF Client



```
/Users/tcefrep/Library/Java/JavaVirtualMachines/o
  N: The number of connection:   1/2Nln(N):
 100              269                 230
 200              585                 529
 300              919                 855
 400             1344                1198
 500             1707                1553
 600             2088                1919
 700             2481                2292
 800             2971                2673
 900             3243                3061
1000             3794                3453
1100             4185                3851
1200             4536                4254
1300             5115                4660
1400             5478                5070
1500             5913                5484
1600             6517                5902
1700             6770                6322
1800             7101                6745
1900             7886                7172
```

Figure 2: Sceenshots of UF Client

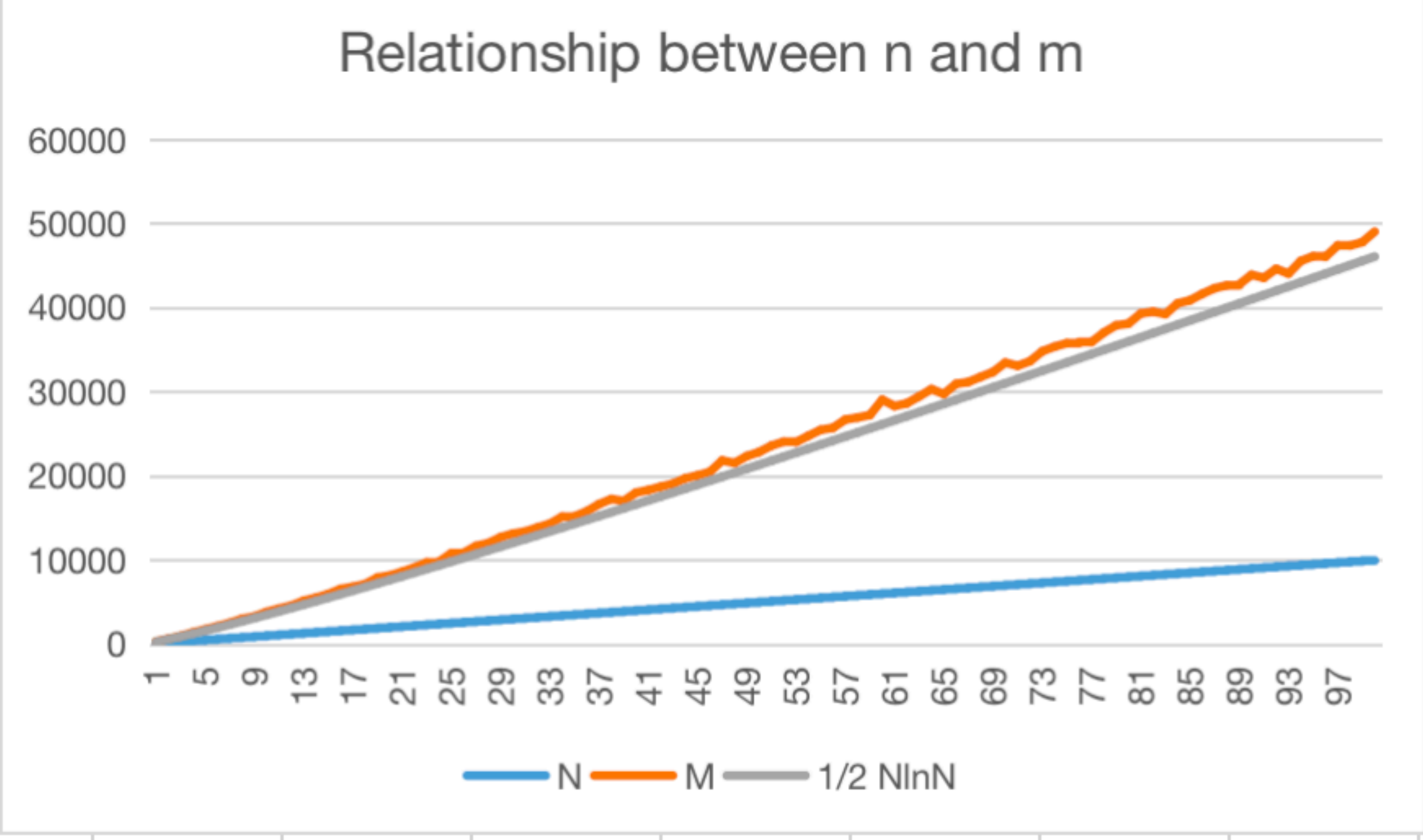


Figure 3: Sceenshots of line chart

5. Conclusion

5.1 Relationship

It can be predicted that when N is larger, it will take more time to connect all points.

> For example:
>
> - N = 100, when there is only one point which is not connected. The probability of drawing it is one percent.
> - N = 1000, when there is only one point which is not connected. The probability of drawing it is one thousandth.

According to the line chart of chapter 4.2. We can deduce the relationship between M (number of pairs) and N (number of objects).

$$M = \frac{1}{2} N \ln N \tag{1}$$