Program Structure and Algorithm

Assignment 2 Union Find

Shouting Lyu

001822094

I developed an UF client satisfied the requirements of the assignment. Specifically, I implemented count() method in the WQUPC Class, and ran the program with different values of n which represents the initial number of components from 2 to 1000, then output each value of n and the total number of pairs which generated by the count() method into the csv file.

The result data from csv file is shown in the figure below. In the figure, the x axis represents n, and the y axis represents the number of pairs generated by the program to accomplish the task. Intuitively, the red line is pretty close to the orange line ($y = ½\ n\ \ln(n)$) with the naked eye. The fitting equation of the red line is $y = 0.0003x^2 + 3.4084x - 86.82$, and the value of R^2 is 0.9066.

Furthermore, the difference between the fitting curve and the hypothesis curve is shown in the figure 2. Obviously, the difference is increasing with the growth of n. In my samples, the largest difference is about 150 while the total number of pairs is more than 3000. Generally, the gradient of the difference function is large in the small value of n between 0 to 400, and decrease as n grows.

In conclusion, in the small value of n, the hypothesis is practical, however, if the value of n is super large, then the hypothesis is not perfectly consistent.

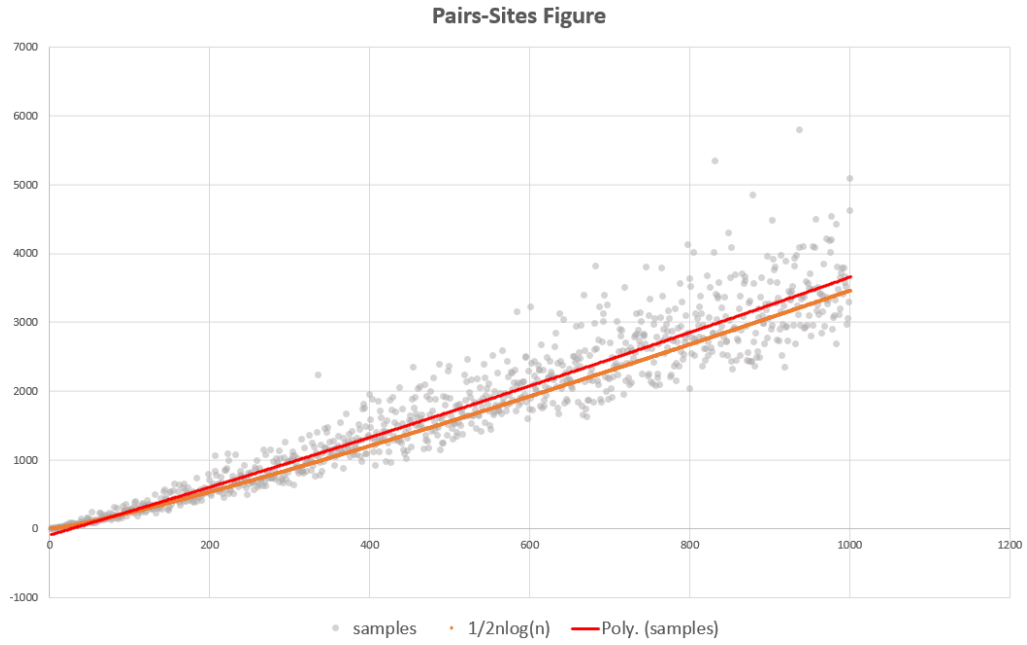

Figure 1 Grey points are results of randomly generated samples, and the red line is polynomial fitting of samples, and the orange line is curve of y = 1/2 n ln(n)

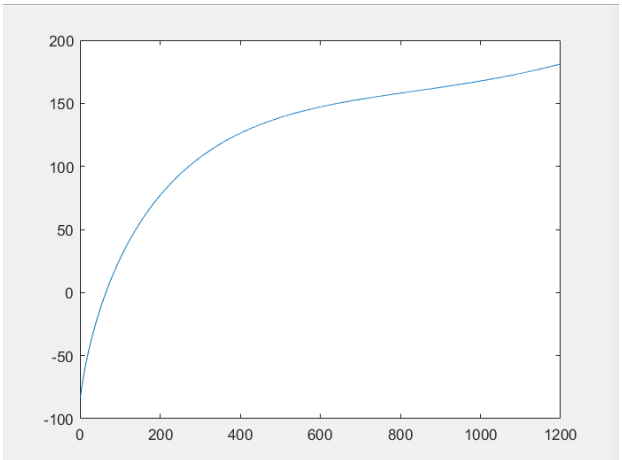

Figure 2 Difference curve referring to previous figure 1

At last, figure 4 demonstrates the code of count(), and figure 3 proves the passing of Unit tests.

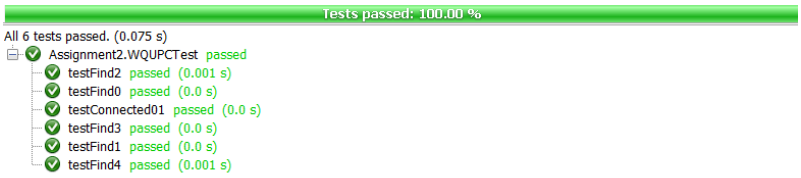

Figure 3

```
public static int count(int n) {
    WQUPC wqupc = new WQUPC(n);
    int pairs = 0;
    boolean complete = false;
    while (complete == false) {
        Random chooser = new Random();
        int a = chooser.nextInt(n);
        int b = chooser.nextInt(n);
        pairs++;
        if (wqupc.connected(a, b)) {
        } else {
            wqupc.union(a, b);
        }
        complete = true;
        for (int k = 0; k < n - 1; k++) {
            if (wqupc.find(k) != wqupc.find(0)) {
                complete = false;
                break;
            }
        }
    }
    return pairs;
}
```

Figure 4