Program Structures and Algorithms
Spring 2023(SEC –3)

NAME: Akshay Parab
NUID: 002766150

Task:
Assignment-4
Determine the relationship between the number of sites (N) and the number of pairs (m) generated if we run height-weighted quick union with path compression algorithm on N sites, generate random pairs of integers between 0 and N-1 calling connected() to determine if they are connected and union() if not and loop until all sites are connected.

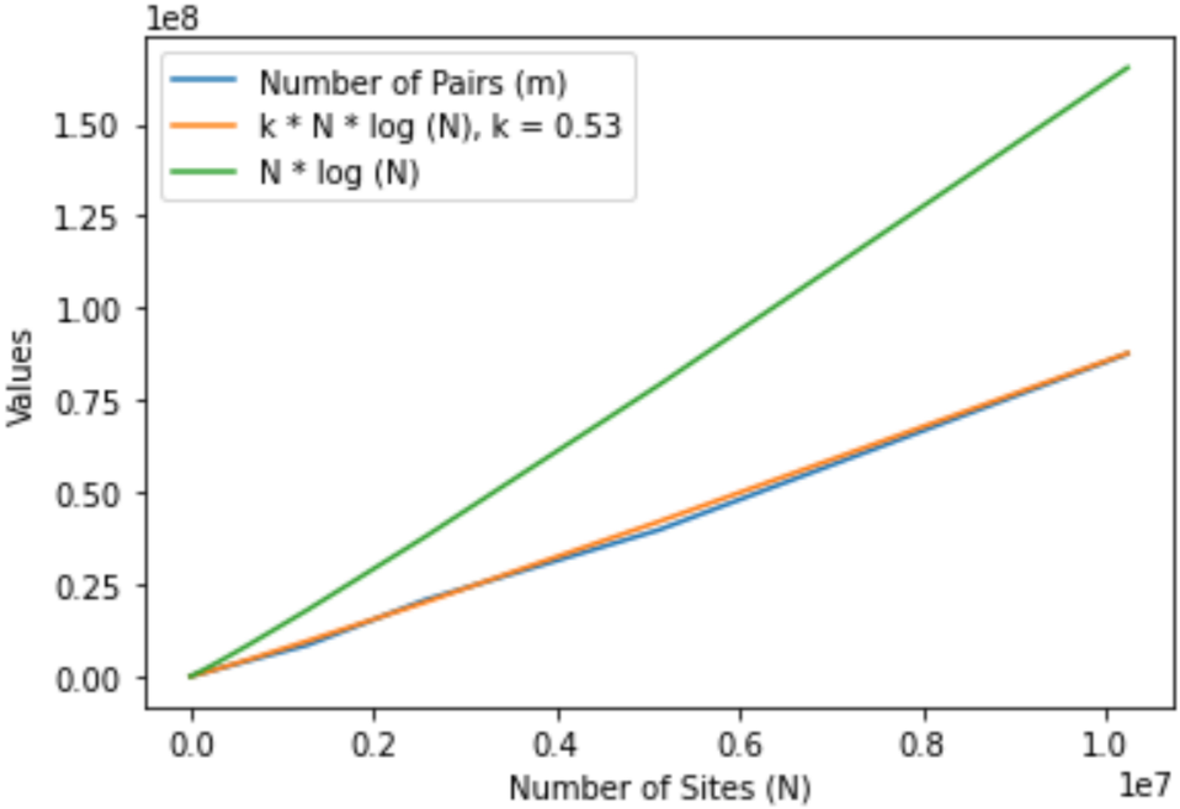
Relationship Conclusion:
The relationship between the number of sites (N) and number of pairs (m) is as follows:
For N sites, we get approximately ***k * N * log (N)*** number of pairs, i.e., we get
m ≅ k * N * log (N) … where k is a constant … equation 1
In my experiments, the range of k was observed to be between 0.47 to 0.53

Evidence to support that conclusion:
I used the doubling method to select number of sites i.e., I chose sufficiently large & different values for N, and I ran the height-weighted Quick Union with Path Compression for each value of N. Following is the table that shows the values of N and the corresponding number of pairs as well as the value of k * N * log (N). Note that, I chose 0.53 as the value of k.

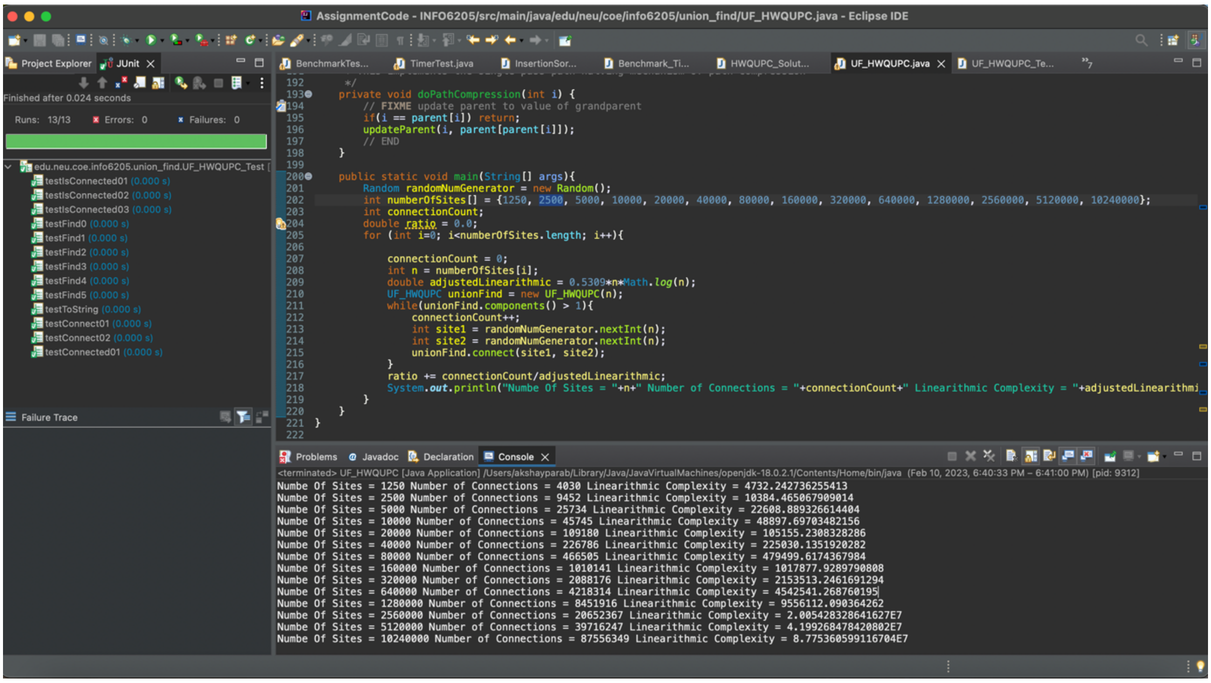
Number of Sites (N)	**Number of Pairs (m)**	**k * N * log (N), …. k = 0.53**
1250	4030	4732.242736255413
2500	9452	10384.465067909014
5000	25734	22608.889326614404
10000	45745	48897.69703482156
20000	109180	105155.2308328286
40000	226786	225030.1351920282
80000	466505	479499.6174367984
160000	1010141	1017877.9289790808
320000	2088176	2153513.2461691294
640000	4218341	4542541.268760195
1280000	8451916	9556112.090364262
2560000	20652367	2.005428328641627E7
5120000	39716247	4.199268478420802E7
10240000	87556349	8.775360599116704E7

Although, we know that since height-weighted quick union with path compression algorithm combines the best of both the quick union and weighted quick union (using height as heuristic in this case) algorithms by using both path compression and weighting to balance the tree structure, union operation takes O (log N) time, and the number of union operations required to connect all N sites will be order of N * log (N). But on continuous running the algorithm, I determined that the value of k in equation was coming to be between 0.47 and 0.53 which means it takes height-weighted quick union with path compression algorithm approximately 0.5 * N * log (N) time to connect all N sites. I believe that due to memorization i.e., the path compression technique, initially it takes more time for the union operation but as and when then path compression kicks in and the tree structure flattens, the further union operations take less time relatively.

Graphical Representation:



Unit Test Screenshots:



Finished after 0.024 seconds

Runs: 13/13 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [
    testIsConnected01 (0.000 s)
    testIsConnected02 (0.000 s)
    testIsConnected03 (0.000 s)
    testFind0 (0.000 s)
    testFind1 (0.000 s)
    testFind2 (0.000 s)
    testFind3 (0.000 s)
    testFind4 (0.000 s)
    testFind5 (0.000 s)
    testToString (0.000 s)
    testConnect01 (0.000 s)
    testConnect02 (0.000 s)
    testConnected01 (0.000 s)
```

```java
     */
    private void doPathCompression(int i) {
        // FIXME update parent to value of grandparent
        if(i == parent[i]) return;
        updateParent(i, parent[parent[i]]);
        // END
    }

    public static void main(String[] args){
        Random randomNumGenerator = new Random();
        int numberOfSites[] = {1250, 2500, 5000, 10000, 20000, 40000, 80000, 160000, 320000, 640000, 1280000, 2560000, 5120000, 10240000};
        int connectionCount;
        double ratio = 0.0;
        for (int i=0; i<numberOfSites.length; i++){

            connectionCount = 0;
            int n = numberOfSites[i];
            double adjustedLinearithmic = 0.5309*n*Math.log(n);
            UF_HWQUPC unionFind = new UF_HWQUPC(n);
            while(unionFind.components() > 1){
                connectionCount++;
                int site1 = randomNumGenerator.nextInt(n);
                int site2 = randomNumGenerator.nextInt(n);
                unionFind.connect(site1, site2);
            }
            ratio += connectionCount/adjustedLinearithmic;
            System.out.println("Numbe Of Sites = "+n+" Number of Connections = "+connectionCount+" Linearithmic Complexity = "+adjustedLinearithmi
        }
    }
}
```

```
<terminated> UF_HWQUPC [Java Application] /Users/akshayparab/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java  (Feb 10, 2023, 6:40:33 PM – 6:41:00 PM) [pid: 9312]
Numbe Of Sites = 1250 Number of Connections = 4030 Linearithmic Complexity = 4732.242736255413
Numbe Of Sites = 2500 Number of Connections = 9452 Linearithmic Complexity = 10384.465067909014
Numbe Of Sites = 5000 Number of Connections = 25734 Linearithmic Complexity = 22608.889326614404
Numbe Of Sites = 10000 Number of Connections = 45745 Linearithmic Complexity = 48897.69703482156
Numbe Of Sites = 20000 Number of Connections = 109180 Linearithmic Complexity = 105155.2308328286
Numbe Of Sites = 40000 Number of Connections = 226786 Linearithmic Complexity = 225030.1351920282
Numbe Of Sites = 80000 Number of Connections = 466505 Linearithmic Complexity = 479499.6174367984
Numbe Of Sites = 160000 Number of Connections = 1010141 Linearithmic Complexity = 1017877.9289790808
Numbe Of Sites = 320000 Number of Connections = 2088176 Linearithmic Complexity = 2153513.2461691294
Numbe Of Sites = 640000 Number of Connections = 4218314 Linearithmic Complexity = 4542541.268760195
Numbe Of Sites = 1280000 Number of Connections = 8451916 Linearithmic Complexity = 9556112.090364262
Numbe Of Sites = 2560000 Number of Connections = 20652367 Linearithmic Complexity = 2.005428328641627E7
Numbe Of Sites = 5120000 Number of Connections = 39716247 Linearithmic Complexity = 4.199268478420802E7
Numbe Of Sites = 10240000 Number of Connections = 87556349 Linearithmic Complexity = 8.775360599116704E7
```