INFO 6205: Assignment 4: Union – Find alternatives.

NUID: 001569774.

1. Weighed quick union by depth:

The complexity for weighed quick union storing and union as per depth (Without path compression) takes (MLogN), N – no of sites, and M operations done.

➔ Code implemented in the repository as UnionByDepthUF.

➔ Apart from the benchmarking, Let us consider as below:

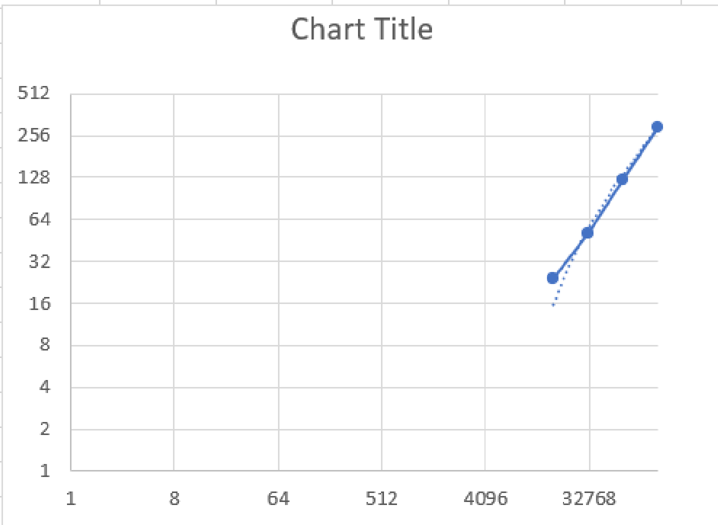
- When considered depth, we increase the depth of the node only when we union 2 equal depth nodes, Hence depth of the node will be incremented only when 2 same depth nodes are made union.
  From the above, the depth of a node will be equal to the depth of the child + 1.
  Depth[n] = Depth[n -1] + 1;
- From the above we can conclude that for every node with depth of d: no of nodes will be >2^d-1.
- We can now deduce that max depth of N will be N/2 + 1; Same way max depth of N/2 sites will be max depth of N/4 + 1;
- When N = 1; the max depth will be as 1, this can deduce that a Union with depth.
- So we can deduce the depth as , D = 1 + 1 + ... Log(N) times.
- This can be considered as the maximum height of the tree.

➔ Consider benchmarking:

Benchmark and plot log-log plot for the values:

Ps: Values may vary depending on the system.

N	T(N)
2000	3.42
4000	6.8
8000	13.09
16000	24.02
32000	51.18
64000	123.96
128000	294.86



Equation as per above slope for a log log plot: T(N) = aN^1.1.

For 8000 takes 13.9 => a = 0.0007.

Now lets consider N as 128000 and apply above values:

T(N) = (128000^1.1)*0.0007 = 291.4 (Same time as per the experiment).

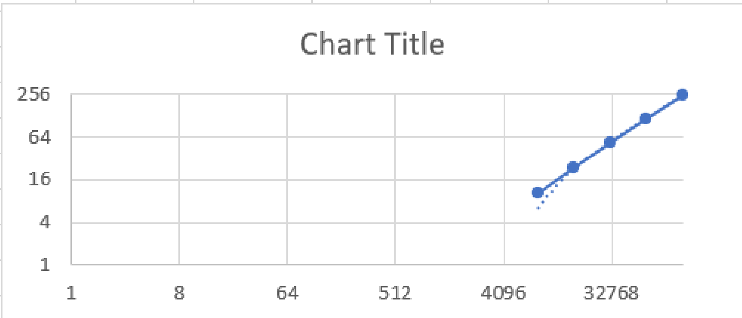
2. Weighed quick union with path compression having all nodes on find path point to root.
   The time complexity for this will be (N + MLoglogN).

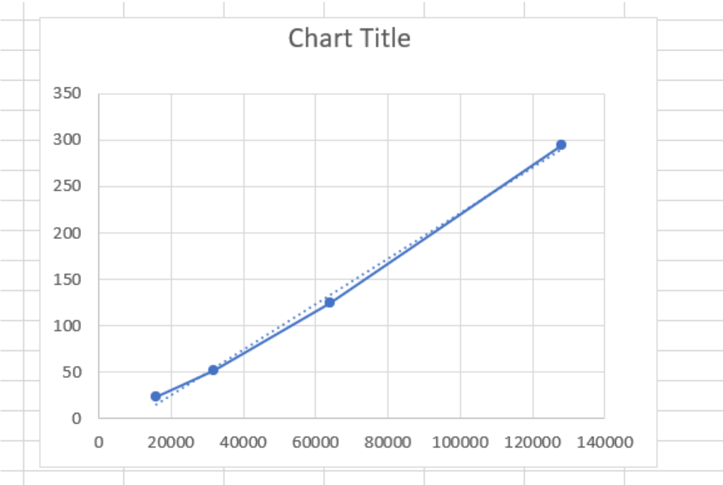
   As we map each node on the find path to the root node. It decreases the height and when called a function next time reduces its run time.
   On this loop initializing takes N for the for loop. And the count takes the constant time but union and find after path compression takes Log*N.

   So the time complexity can be taken as N + MLogLogN, ~ (N + M), where M is the number of pairs used.

- ➔ Code loaded in repository as WeighedUnionFindWithPathCompressionAllNodes.
- ➔ Used UFClient.java in repository to run the benchmarking as a function.

N	T(N)
8000	10.3
16000	23.26
32000	52.04
64000	115.96
128000	252.78






```
data = py.read_excel('Documents/DataSet_Assignment2.xlsx')
print(data)
```

```
        N    T(N)
0    2000    2.94
1    4000    5.88
2    8000   10.30
3   16000   23.26
4   32000   52.04
5   64000  115.96
6  128000  252.78
```

```
data['LogM'] = np.log2(data['T(N)'])
data['LogN'] = np.log2(data['N'])
print(data)
```

```
        N    T(N)      LogM       LogN
0    2000    2.94  1.555816  10.965784
1    4000    5.88  2.555816  11.965784
2    8000   10.30  3.364572  12.965784
3   16000   23.26  4.539779  13.965784
4   32000   52.04  5.701549  14.965784
5   64000  115.96  6.857483  15.965784
6  128000  252.78  7.981739  16.965784
```

The equation as per the graph as per log log plot slope equation:

T(N) = aN^1.13. (Order of growth)

When N = 8000 => a = 0.00043.

Proof of working:

Let us apply the values in the equation for N = 128000:

T(N) = (128000^1.13) * 0.00043 = 253.87 (i.e ~ 252.78)