Program Structures and Algorithms
Fall 2023

NAME:Shangqing Hu
NUID:001374342

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.

Relationship Conclusion:
The number of pairs (m) needed to reduce the number of components from n to 1 is initially estimated to be proportional to n log n. This estimation arises from the notion that the union operation needs to be executed log n times for each of the n objects to achieve this reduction.

However, in practical implementation, the actual number of pairs generated may turn out to be less than n log n. This is because the height-weighted Quick Union Find algorithm, coupled with path compression, effectively reduces the height of the trees, resulting in fewer nodes within the trees. As a consequence, this reduction in tree size minimizes the number of union operations required to reach a single component.

In summary, while the theoretical relationship between the number of objects (n) and the number of pairs (m) generated to accomplish the reduction of components from n to 1 is proportional to n log n, in practice, the actual number of pairs generated may be lower due to the beneficial impact of path compression.

Evidence to support that conclusion:

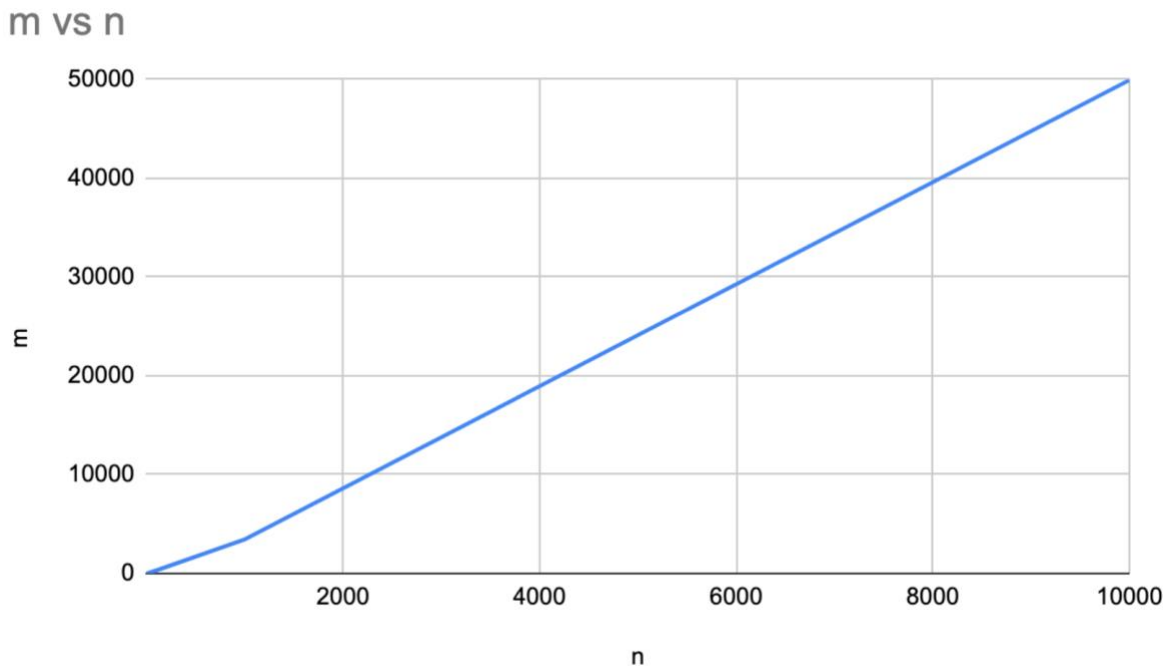
```
n = 1, Number of connections(m) : 0
n = 10, Number of connections(m) : 11
n = 100, Number of connections(m) : 290
n = 1000, Number of connections(m) : 3430
n = 10000, Number of connections(m) : 49935

Process finished with exit code 0
```

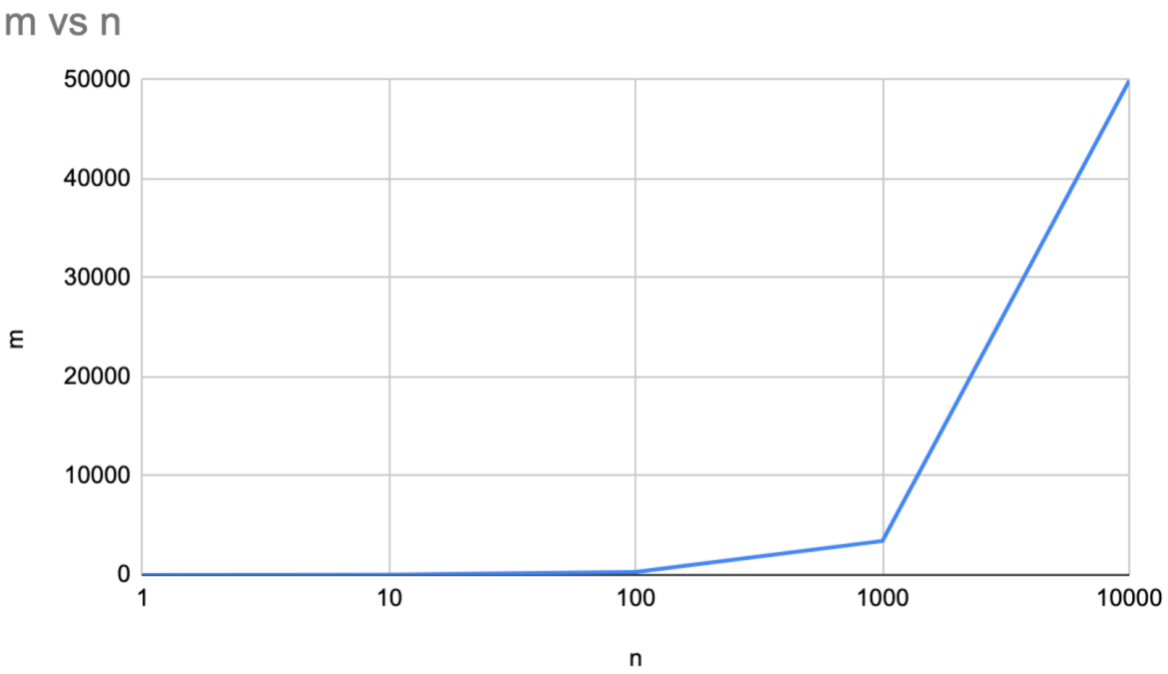
n	m
1	0
10	11
100	290
1000	3430
10000	49935

These observations highlight the efficiency of the height-weighted Quick Union Find with path compression algorithm in reducing the number of components from n to 1. However, as the number of objects increases, the algorithm's time complexity also grows. Path compression mitigates this growth to some extent, but the overall time complexity still exhibits a logarithmic increase with the number of objects.

Graphical Representation:



Logarimathtic Scale



Screenshots of run and/or Unit Test: