Program Structures and Algorithms
Spring 2023(SEC – 01)

NAME: Kasavaraju Venkata Pranay Kumar
NUID: 002701155

Task:

1. Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF_HWQUPC.
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).
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:
As we observe various cases we find that there is a distinct relationship the number of nodes and number of pairs generated randomly and connected using union Find. The relationship observed between number of nodes and number of pairs is Linear i.e. $y = mx + c$. Here y denoted number of pairs generated that are connected using union and x denotes number of nodes.
With the data calculated, we can say that m is approximately 4.925.

Evidence to support that conclusion:
Observations Nodes vs Pairs:

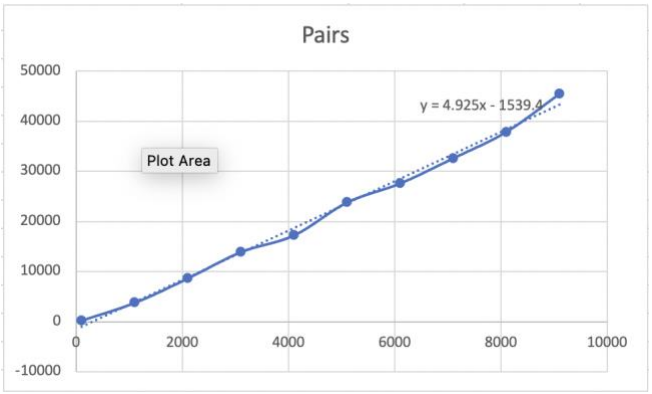
Number of Nodes (n)	Number of Pairs (m)
100	262
1100	3782
2100	8658
3100	13909
4100	17232
5100	23801
6100	27588
7100	32616
8100	37861

```
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java
============================================================
Number of nodes: 100
Number of pairs:262
Number of connections:99
============================================================
Number of nodes: 1100
Number of pairs:3782
Number of connections:1099
============================================================
Number of nodes: 2100
Number of pairs:8658
Number of connections:2099
============================================================
Number of nodes: 3100
Number of pairs:13909
Number of connections:3099
============================================================
Number of nodes: 4100
Number of pairs:17232
Number of connections:4099
============================================================
Number of nodes: 5100
Number of pairs:23801
Number of connections:5099
============================================================
Number of nodes: 6100
Number of pairs:27588
Number of connections:6099
```

```
============================================================
Number of nodes: 7100
Number of pairs:32616
Number of connections:7099
============================================================
Number of nodes: 8100
Number of pairs:37861
Number of connections:8099
============================================================
Number of nodes: 9100
Number of pairs:45449
Number of connections:9099
```

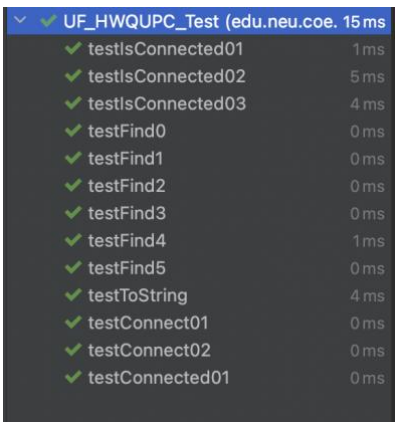
Graphical Representation:

1. Nodes Vs Pairs



Unit Test Screenshots:

UF_HWQUPC_Test.java



Code Snippets:

UF_HWQUPC.java:

```java
public int find(int p) {
    validate(p);
    int root = p;
    // FIXME
    while (root != getParent(root)){
        if(this.pathCompression) {
            doPathCompression(root);
        }
        root = getParent(root);
    }
    // END
    return root;
}
```

```java
private void mergeComponents(int i, int j) {
    // FIXME make shorter root point to taller one
    if (i == j )
        return;
    if(height[i]<height[j]){
        updateParent(i,j);
    }
    else if(height[j]<height[i]) {
        updateParent(j,i);
    }
    else{
        updateParent(j,i);
        height[i]++;
    }
    // END
}

/**
 * This implements the single-pass path-halving mechanism of path compression
 */
1 usage  Kasavaraju-v *
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    parent[i]=getParent(getParent(i));
    // END
}
}
```