

**Narendra Raj R K (001553969)**  
**Program Structures & Algorithms**  
**Spring 21**  
**Assignment No. 3**

**Tasks:**

Part 1: In this task I am to implement Height Weighted Quick Union with Path Compression by adding logic to the following methods: **find(), mergeComponents() & doPathCompression()**.

⇒ Successfully added the logic to the methods required in order to achieve Height Weighted Quick Union with Path Compression. All the unit tests for this class passed.

Part 2: Develop a **UF ("union-find") client** that takes an integer value n from the command line to determine the number of "sites". Finally arrive on the number of connections generated.

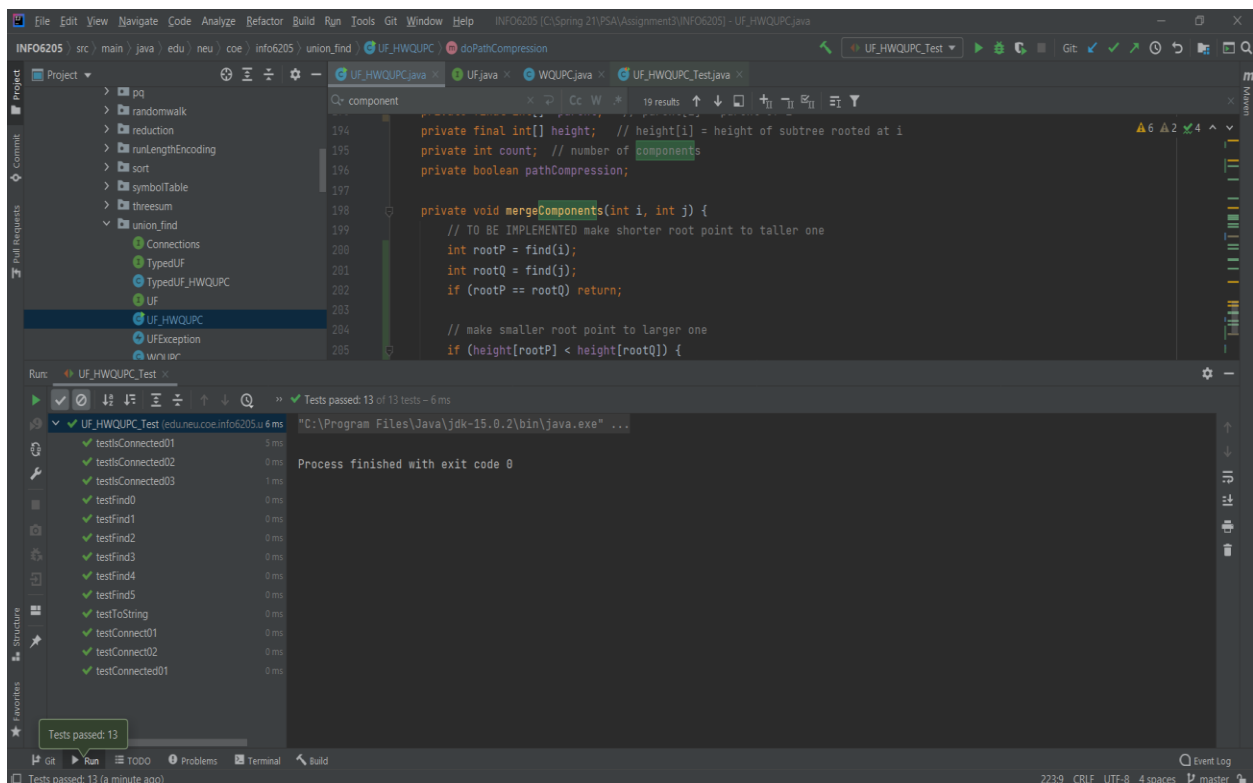
⇒ Generated a random pair of integers and called the connect() which iterates until all pairs are connected and finally returned the number of connections "m" value.

Part 3: Determine the relationship between the number of objects **(n)** and the number of pairs **(m)**.

⇒ In order to establish a relation between the number of objects (n) and the number of pairs (m) I reduced the size of the component from size of n to 1.

## Unit Test Results:

13 test cases under the UF\_HWQUPC\_Test.java file passed successfully.



## Snippets of Methods Implemented:

### I. find()

```
/**
 * Returns the component identifier for the component containing site {@code p}.
 *
 * @param p the integer representing one site
 * @return the component identifier for the component containing site {@code p}
 * @throws IllegalArgumentException unless {@code 0 <= p < n}
 */
public int find(int p) {
    validate(p);
    int root = p;
    // TO BE IMPLEMENTED
    while (root != parent[root])
        root = parent[root];

    if(pathCompression) {
        doPathCompression(p);
        return root;
    }
    else
        return root;
}
```

### II. doPathCompression()

```
/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int n) {
    // TO BE IMPLEMENTED update parent to value of grandparent
    parent[n] = parent[parent[n]];
}
```

### III. mergeComponents()

```
private void mergeComponents(int i, int j) {
    // TO BE IMPLEMENTED make shorter root point to taller one
    int rootP = find(i);
    int rootQ = find(j);
    if (rootP == rootQ) return;

    // make smaller root point to larger one
    if (height[rootP] < height[rootQ]) {
        parent[rootP] = rootQ;
    }
    //If Height is equal then point root of one component to another
    else if (height[rootP] == height[rootQ]){
        parent[rootQ] = rootP;
        height[rootP]++;
    }
    else {
        parent[rootQ] = rootP;
    }
}
```

### IV. generatePairs()

```
//to generate the number of pairs required to make a component of size 1
public static int generatePairs(int n){
    UF_HWQUPC obj = new UF_HWQUPC(n, pathCompression: true);

    int pairCount = 0;

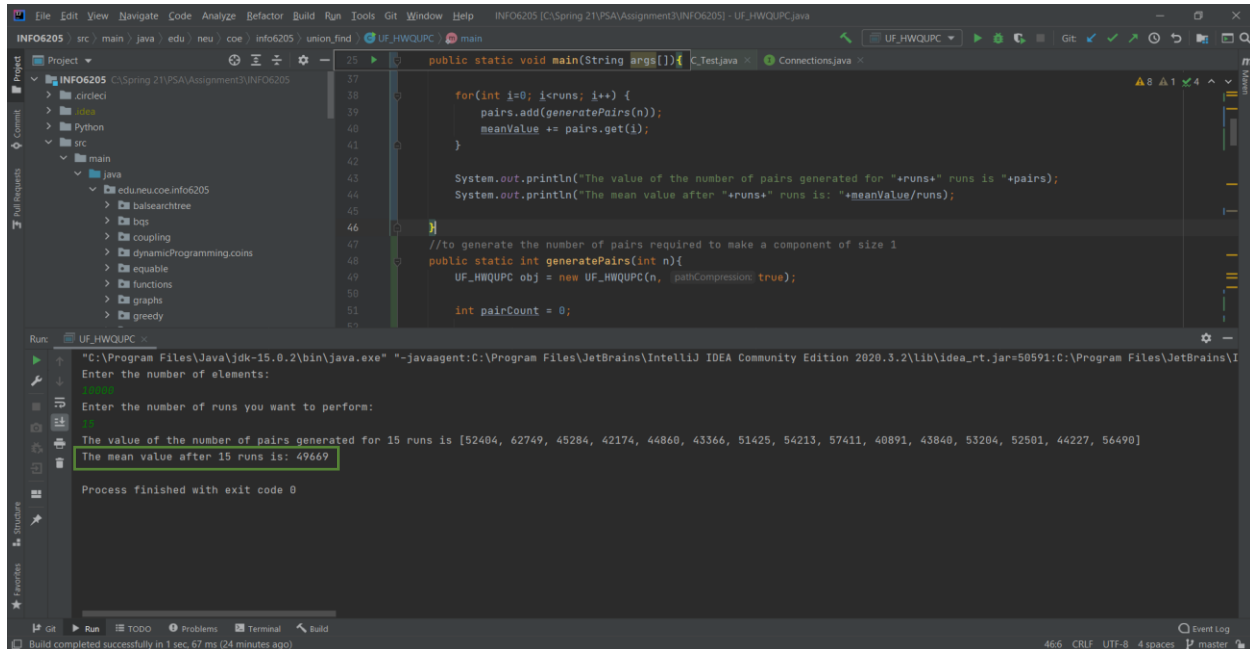
    while (obj.components() != 1) {
        int pair_val1 = ThreadLocalRandom.current().nextInt(origin: 0, n);
        int pair_val2 = ThreadLocalRandom.current().nextInt(origin: 0, n);
        obj.connect(pair_val1, pair_val2);
        pairCount++;
    }
    return pairCount;
}
```

**Note:** The above method was added by me to generate the number of pairs.

## Evidence to Support Conclusion:

**Attempt 1:** Number of Elements is 10000 & the Number of Runs is 15

➔ Number of Pairs: **49669**



```
public static void main(String args[]) {
    for(int i=0; i<runs; i++) {
        pairs.add(generatePairs(n));
        meanValue += pairs.get(i);
    }

    System.out.println("The value of the number of pairs generated for "+runs+" runs is "+pairs);
    System.out.println("The mean value after "+runs+" runs is: "+meanValue/runs);
}

//to generate the number of pairs required to make a component of size 1
public static int generatePairs(int n){
    UF_HWQUPC obj = new UF_HWQUPC(n, pathCompression: true);

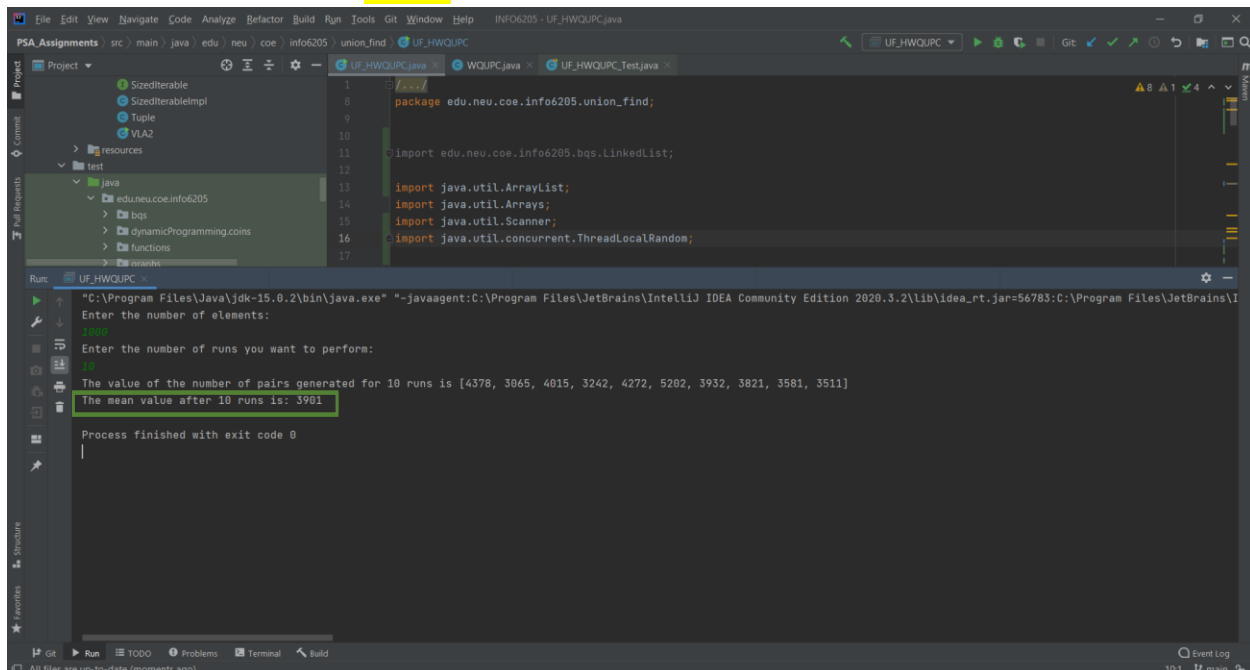
    int pairCount = 0;
```

Run: UF\_HWQUPC

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2\lib\idea_rt.jar=50591:C:\Program Files\JetBrains\I
Enter the number of elements:
10000
Enter the number of runs you want to perform:
15
The value of the number of pairs generated for 15 runs is [52404, 62749, 45284, 42174, 44860, 43366, 51425, 54213, 57411, 40891, 43840, 53204, 52501, 44227, 56490]
The mean value after 15 runs is: 49669
Process finished with exit code 0
```

**Attempt 2:** Number of Elements is 1000 & the Number of Runs is 10

➔ Number of Pairs: **3901**



```
package edu.neu.coe.info6205.union_find;

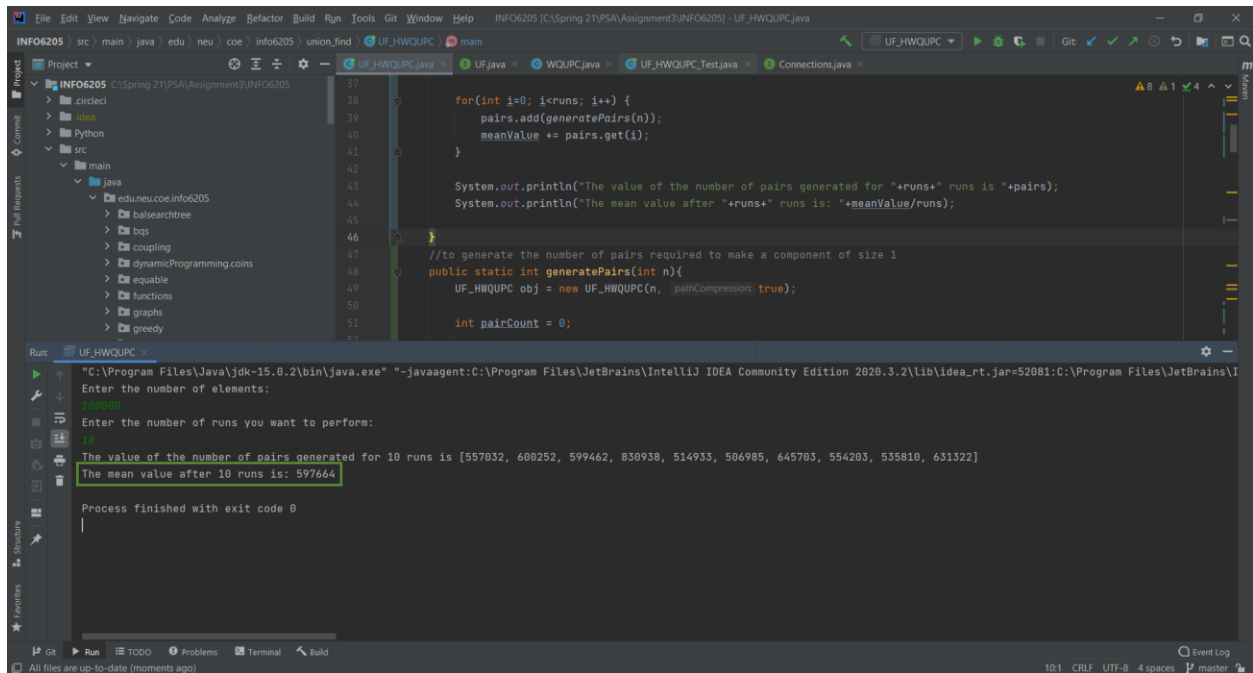
import edu.neu.coe.info6205.bqs.LinkedList;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.ThreadLocalRandom;
```

Run: UF\_HWQUPC

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2\lib\idea_rt.jar=56783:C:\Program Files\JetBrains\I
Enter the number of elements:
1000
Enter the number of runs you want to perform:
10
The value of the number of pairs generated for 10 runs is [4378, 3065, 4015, 3242, 4272, 5202, 3932, 3821, 3581, 3511]
The mean value after 10 runs is: 3901
Process finished with exit code 0
```

**Attempt 3:** Number of Elements is 100000 & the Number of Runs is 10

➔ Number of Pairs: **597664**



The screenshot shows an IDE with a project named 'INFO6205'. The code in 'UF\_HWQUPC.java' includes a loop for generating pairs and a method 'generatePairs'. The output window shows the results of 10 runs, with the mean value highlighted as 597664.

```
for(int i=0; i<runs; i++) {
    pairs.add(generatePairs(n));
    meanValue += pairs.get(i);
}

System.out.println("The value of the number of pairs generated for "+runs+" runs is "+pairs);
System.out.println("The mean value after "+runs+" runs is: "+meanValue/runs);
}
```

```
//to generate the number of pairs required to make a component of size 1
public static int generatePairs(int n){
    UF_HWQUPC obj = new UF_HWQUPC(n, pathCompression: true);

    int pairCount = 0;
```

Run: UF\_HWQUPC

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2\lib\idea_rt.jar=52081:C:\Program Files\JetBrains\I
Enter the number of elements:
100000
Enter the number of runs you want to perform:
10
The value of the number of pairs generated for 10 runs is [557032, 600252, 599462, 830938, 514933, 506985, 645703, 554203, 535810, 631322]
The mean value after 10 runs is: 597664
Process finished with exit code 0
```

## Final Conclusion:

After running the Path Compression logic on Height Weighted Quick Union multiple times and getting the mean value since we are dealing with random pairs of integers, I have arrived at the following relation between the number of elements (n) and the number of pairs (m).

$$M = ( N * \ln N ) / 2$$

**ln** – It is the Natural Logarithm with base e whose value is 2.718281828459