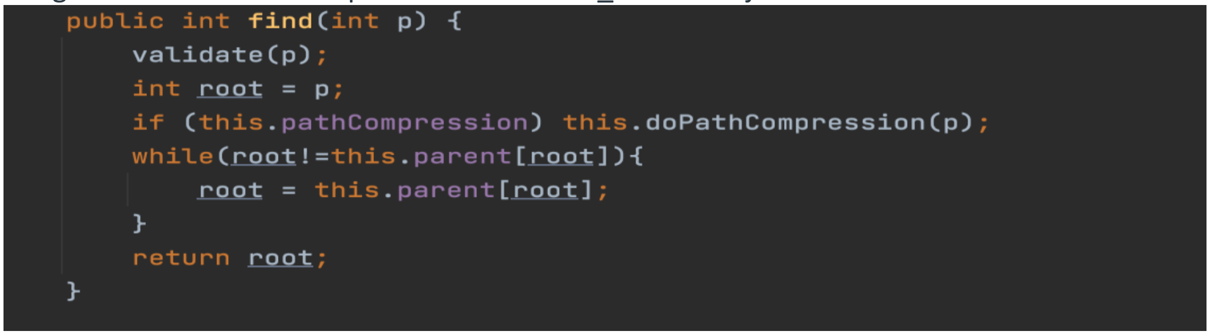
**Venkata Sai Akhil Vemula**
**002981289**

# INFO 6205 Program Structures and Algorithms
# Assignment 3

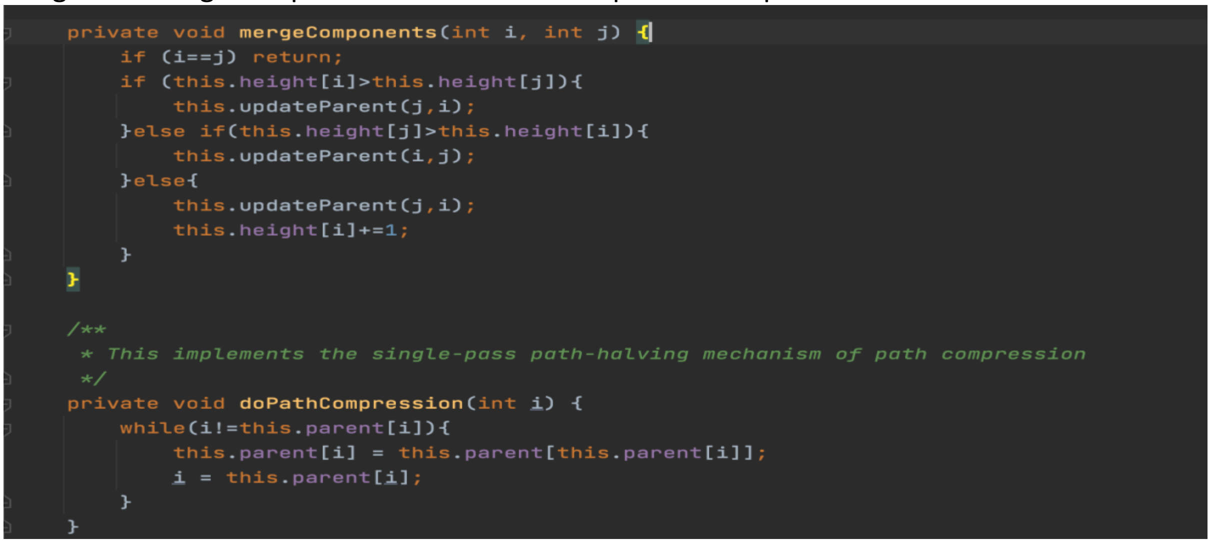
## Task 1: Implement height-weighted Quick Union with Path Compression

Image 1.1: find method implementation in UF_HWQUPC.java



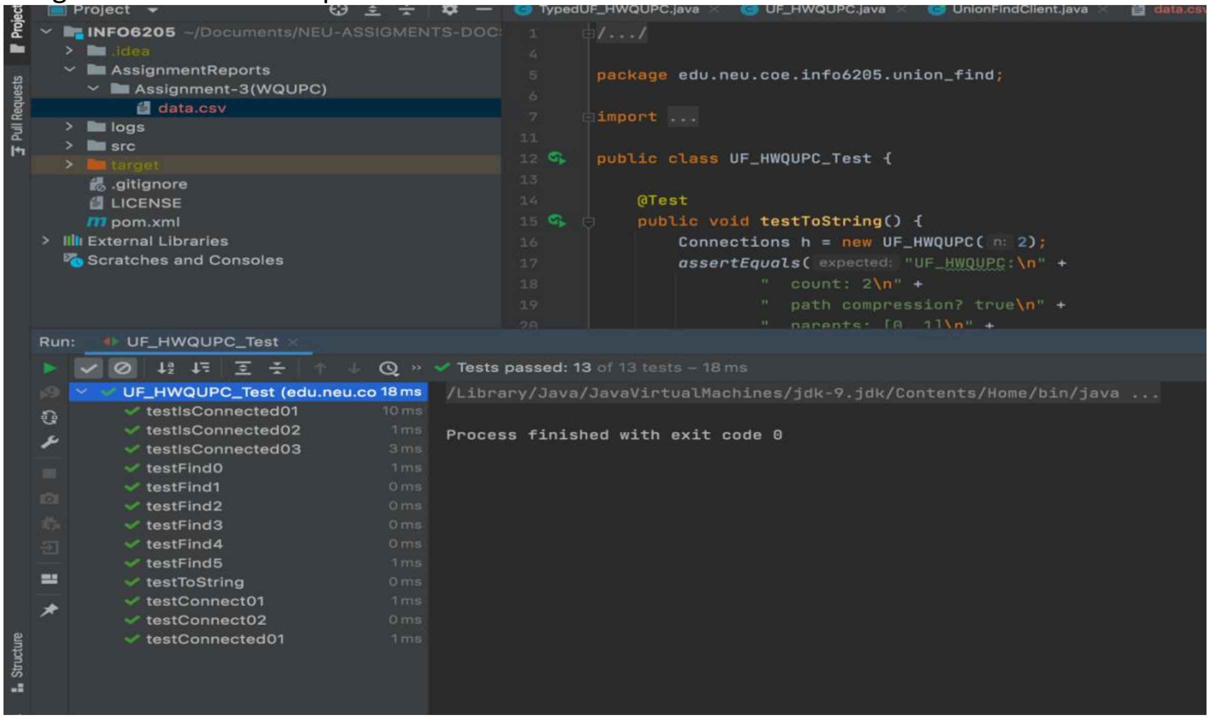
```java
public int find(int p) {
    validate(p);
    int root = p;
    if (this.pathCompression) this.doPathCompression(p);
    while(root!=this.parent[root]){
        root = this.parent[root];
    }
    return root;
}
```

Image 1.2: mergeComponents and doPathCompression Implementation



```java
private void mergeComponents(int i, int j) {
    if (i==j) return;
    if (this.height[i]>this.height[j]){
        this.updateParent(j,i);
    }else if(this.height[j]>this.height[i]){
        this.updateParent(i,j);
    }else{
        this.updateParent(j,i);
        this.height[i]+=1;
    }
}

/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int i) {
    while(i!=this.parent[i]){
        this.parent[i] = this.parent[this.parent[i]];
        i = this.parent[i];
    }
}
```

Image 1.3: All test cases passed



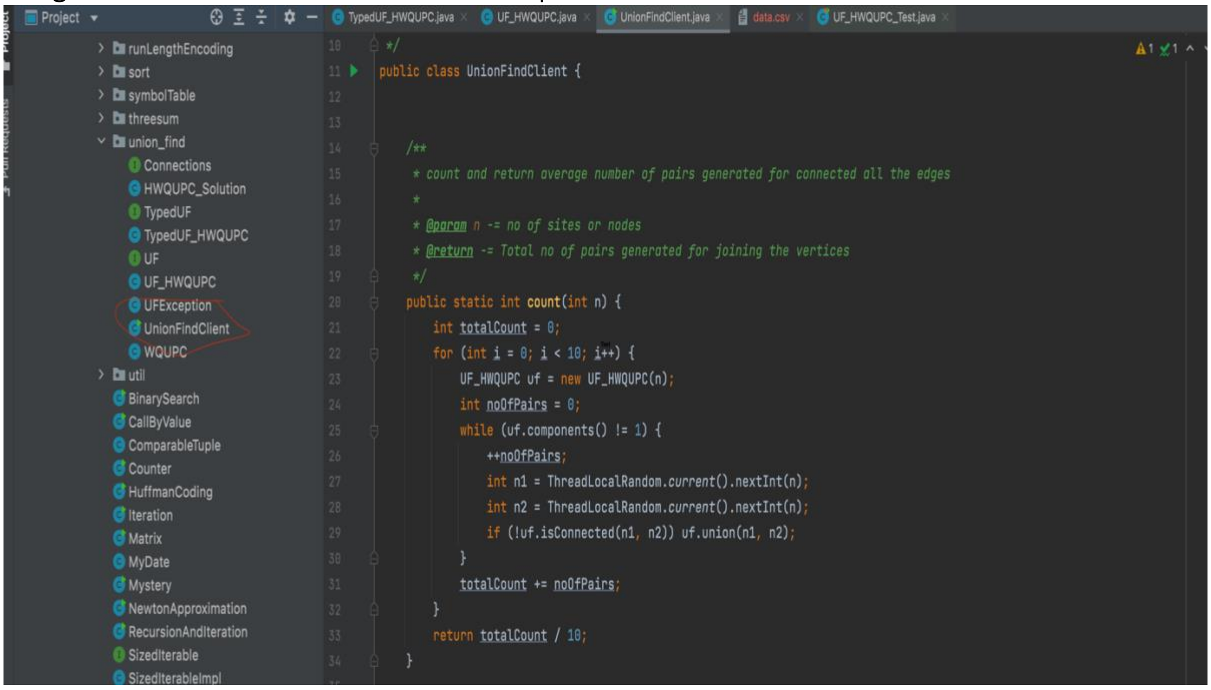
```java
/.../

package edu.neu.coe.info6205.union_find;

import ...

public class UF_HWQUPC_Test {

    @Test
    public void testToString() {
        Connections h = new UF_HWQUPC( n: 2);
        assertEquals( expected: "UF_HWQUPC:\n" +
                "  count: 2\n" +
                "  path compression? true\n" +
                "  parents: [0, 1]\n" +
```

```
Tests passed: 13 of 13 tests – 18 ms

UF_HWQUPC_Test (edu.neu.co 18 ms
    testIsConnected01    10 ms
    testIsConnected02     1 ms
    testIsConnected03     3 ms
    testFind0             1 ms
    testFind1             0 ms
    testFind2             0 ms
    testFind3             0 ms
    testFind4             0 ms
    testFind5             1 ms
    testToString          0 ms
    testConnect01         1 ms
    testConnect02         0 ms
    testConnected01       1 ms

/Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

## Step 2: Using your implementation of UF_HWQUPC, develop a UF ("union-find") client

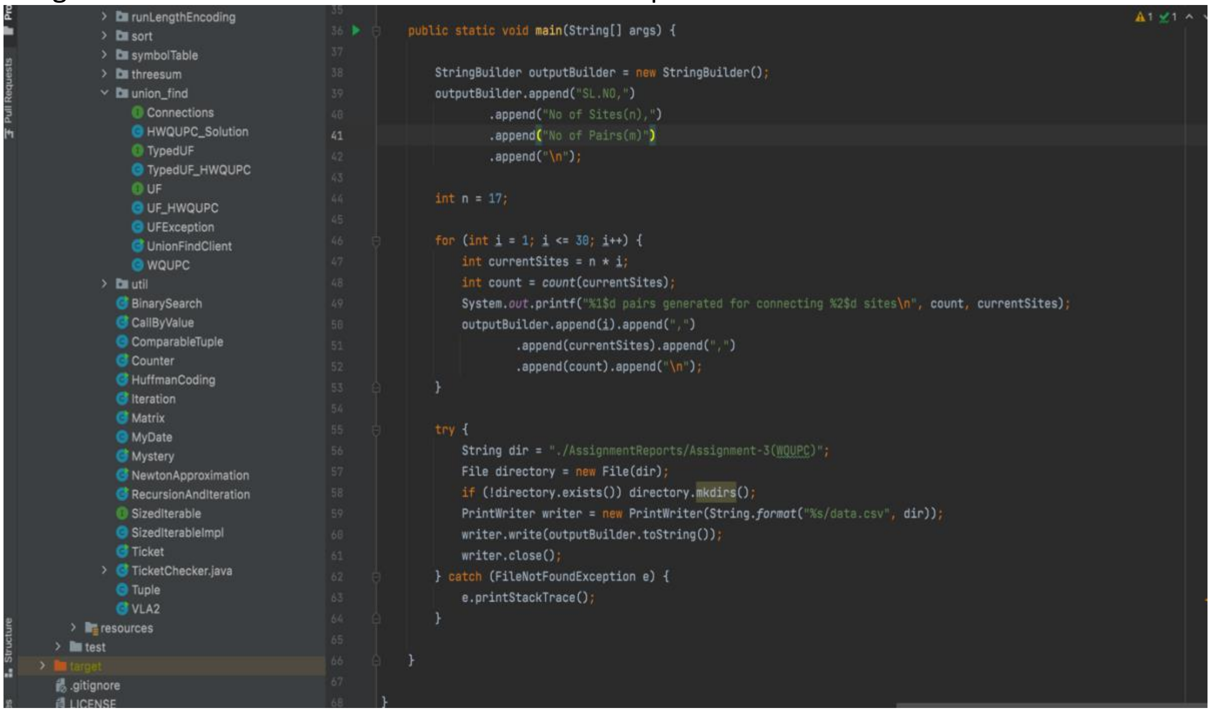
Created class named UnionFindClient in the same package as UF_HWQUPC and implemented count and main static methods. Took initial n= as 17 and incremented every time by 17 for 30 times. Made 10 experiments/loops for generating average no of pairs that are required to complete Union.

Image 2.1 UnionFindClient class count Implementation



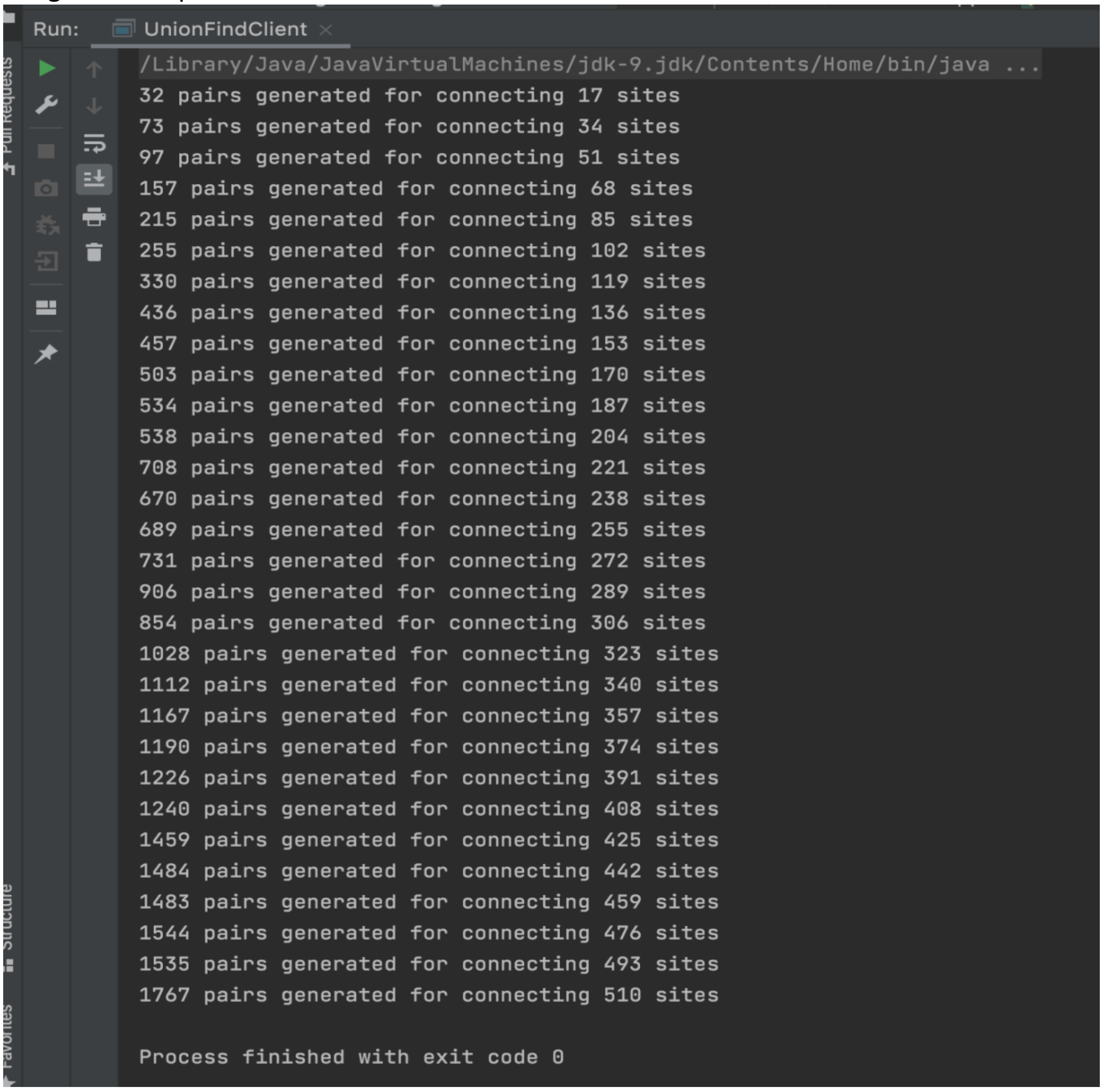
```java
 */
public class UnionFindClient {


    /**
     * count and return average number of pairs generated for connected all the edges
     *
     * @param n -= no of sites or nodes
     * @return -= Total no of pairs generated for joining the vertices
     */
    public static int count(int n) {
        int totalCount = 0;
        for (int i = 0; i < 10; i++) {
            UF_HWQUPC uf = new UF_HWQUPC(n);
            int noOfPairs = 0;
            while (uf.components() != 1) {
                ++noOfPairs;
                int n1 = ThreadLocalRandom.current().nextInt(n);
                int n2 = ThreadLocalRandom.current().nextInt(n);
                if (!uf.isConnected(n1, n2)) uf.union(n1, n2);
            }
            totalCount += noOfPairs;
        }
        return totalCount / 10;
    }
```

Image 2.2: UnionFindClient class Main Method implementation



```java
    public static void main(String[] args) {

        StringBuilder outputBuilder = new StringBuilder();
        outputBuilder.append("SL.NO,")
                .append("No of Sites(n),")
                .append("No of Pairs(m)")
                .append("\n");

        int n = 17;

        for (int i = 1; i <= 30; i++) {
            int currentSites = n * i;
            int count = count(currentSites);
            System.out.printf("%1$d pairs generated for connecting %2$d sites\n", count, currentSites);
            outputBuilder.append(i).append(",")
                    .append(currentSites).append(",")
                    .append(count).append("\n");
        }

        try {
            String dir = "./AssignmentReports/Assignment-3(WQUPC)";
            File directory = new File(dir);
            if (!directory.exists()) directory.mkdirs();
            PrintWriter writer = new PrintWriter(String.format("%s/data.csv", dir));
            writer.write(outputBuilder.toString());
            writer.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

    }

}
```

Image 2.3: Output of runs



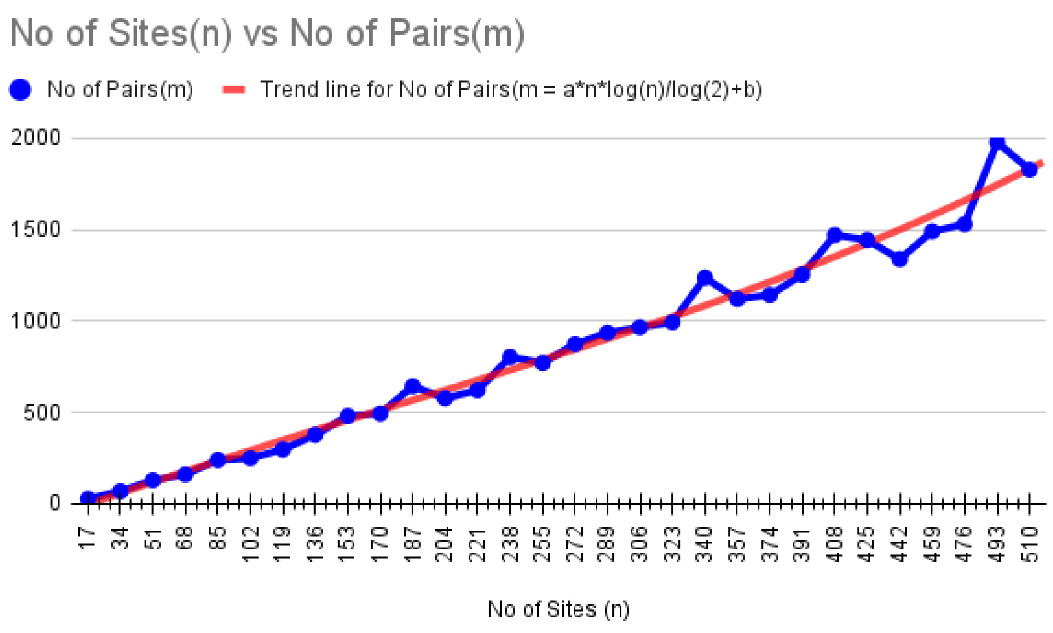
Run: UnionFindClient

```
/Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java ...
32 pairs generated for connecting 17 sites
73 pairs generated for connecting 34 sites
97 pairs generated for connecting 51 sites
157 pairs generated for connecting 68 sites
215 pairs generated for connecting 85 sites
255 pairs generated for connecting 102 sites
330 pairs generated for connecting 119 sites
436 pairs generated for connecting 136 sites
457 pairs generated for connecting 153 sites
503 pairs generated for connecting 170 sites
534 pairs generated for connecting 187 sites
538 pairs generated for connecting 204 sites
708 pairs generated for connecting 221 sites
670 pairs generated for connecting 238 sites
689 pairs generated for connecting 255 sites
731 pairs generated for connecting 272 sites
906 pairs generated for connecting 289 sites
854 pairs generated for connecting 306 sites
1028 pairs generated for connecting 323 sites
1112 pairs generated for connecting 340 sites
1167 pairs generated for connecting 357 sites
1190 pairs generated for connecting 374 sites
1226 pairs generated for connecting 391 sites
1240 pairs generated for connecting 408 sites
1459 pairs generated for connecting 425 sites
1484 pairs generated for connecting 442 sites
1483 pairs generated for connecting 459 sites
1544 pairs generated for connecting 476 sites
1535 pairs generated for connecting 493 sites
1767 pairs generated for connecting 510 sites

Process finished with exit code 0
```

**STEP 3 IN NEXT PAGE**

**Step 3: (Evidences & Conclusion)**
Determine the relationship between the number of objects (*n*) and the number of pairs (*m*)

Graph 1: n vs m



Google Sheet Link [https://docs.google.com/spreadsheets/d/1ed8vp7l4cgqy1cxlZZ5VHia-kUyPI-xsJiSpBcisW40/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1ed8vp7l4cgqy1cxlZZ5VHia-kUyPI-xsJiSpBcisW40/edit?usp=sharing)

** CSV file is in the same directory as this report

**Conclusion**: It is observed that the values(no of pairs) obtained from the experiment are almost equal to NLog$_2$(N) in most of the cases. The same trend was observed in the trend line which is generated by plotting a graph in google sheet(Trend line was plotted using google sheet in-built functions). Though total no of pairs are almost equal to above said values there is some fluctuation due to the randomness. Hence it can be concluded that

$$\mathbf{M = A*N*Log_2(N) + B}$$

Where

M = total number of pairs
N = total no of sites/nodes
A = co-efficient (According to google sheet this is equal to 0.65433, however this is changing in every main method run)
B = co-efficient