Tanay Saxena (001586302)
Program Structures & Algorithms
Fall 2021
Assignment No. 3

- Task (List down the tasks performed in the Assignment)
  - Added missing code in 4 methods -
    Class UF_HWQUPC.java
    find() -

```java
public int find(int p) {
    validate(p);
    int root = p;
    // TO BE IMPLEMENTED
    if (pathCompression) doPathCompression(root);
    while (root != parent[root]) {
        root = parent[root];
    }
    return root;
}
```

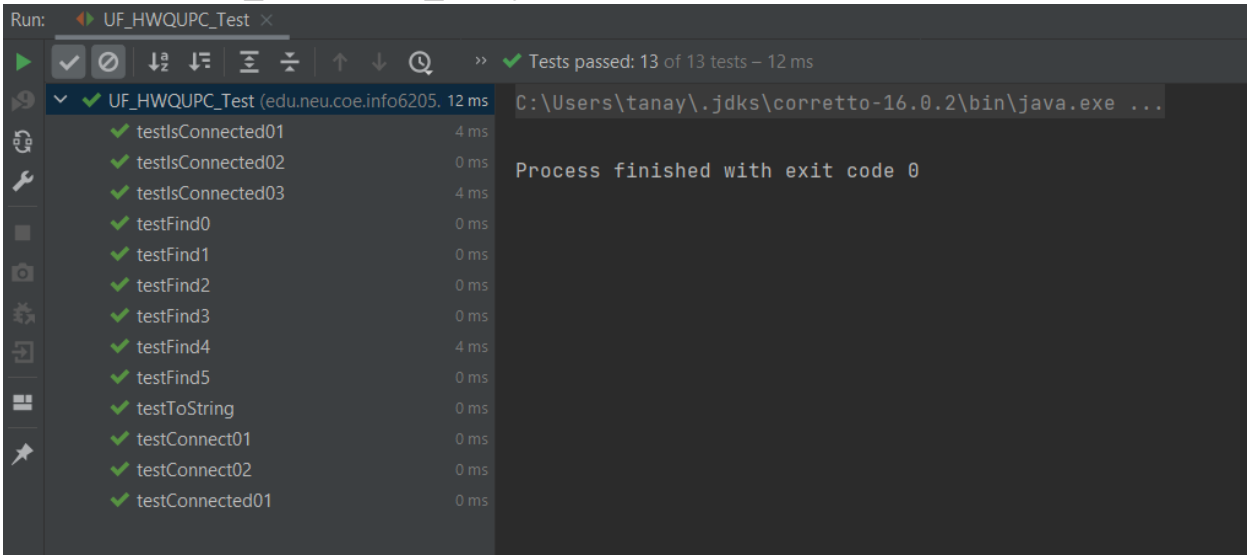
mergeComponents() -

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

doPathCompression()

```java
private void doPathCompression(int i) {
    // TO BE IMPLEMENTED update parent to value of grandparent
    while (i != parent[i]) {
        parent[i] = parent[parent[i]];
        i = parent[i];
    }
}
```

- Ran tests for UF_HWQUPC_Test.java-



- Created a new class UnionFindClient.java for the analysis -

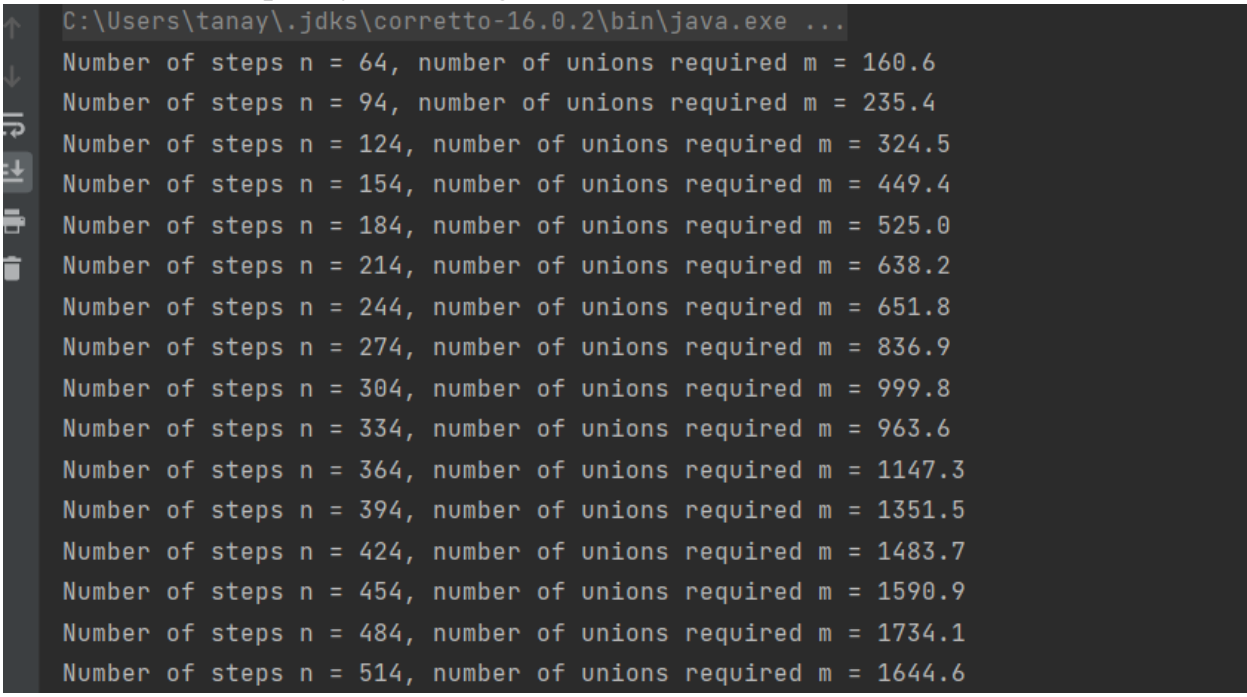
```java
public class UnionFindClient {

    public static double count(int n, int epochs) {
        double avgCount = 0;
        for (int t = 0; t < epochs; t++) {
            UF_HWQUPC dsu = new UF_HWQUPC(n, pathCompression: true);
            Random random = new Random();
            int curCOunt = 0;
            while (dsu.components() != 1) {
                dsu.connect((int)(Math.random() * n), (int)(Math.random() * n));
                curCOunt++;
            }
            avgCount += curCOunt;
        }
        return avgCount/epochs;
    }
```

```java

    public static void main(String[] args) {
        try {
            String fileSeparator = File.separator;
            FileWriter writer = new FileWriter(Paths.get( first: "assignment_reports",
                    ...more: "assignment3_Tanay_Saxena", "union_find.csv").toString());
            writer.write( str: "inputSize,pairCount\n");
            for (int n = 64; n <= 10000; n += 30) {
                double m = UnionFindClient.count(n,  epochs: 10);
                System.out.println("Number of steps n = " + n + ", number of unions required m = " + m);
                writer.write( str: n + "," + m + "\n");
            }
            writer.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Generated the output by executing the main method -

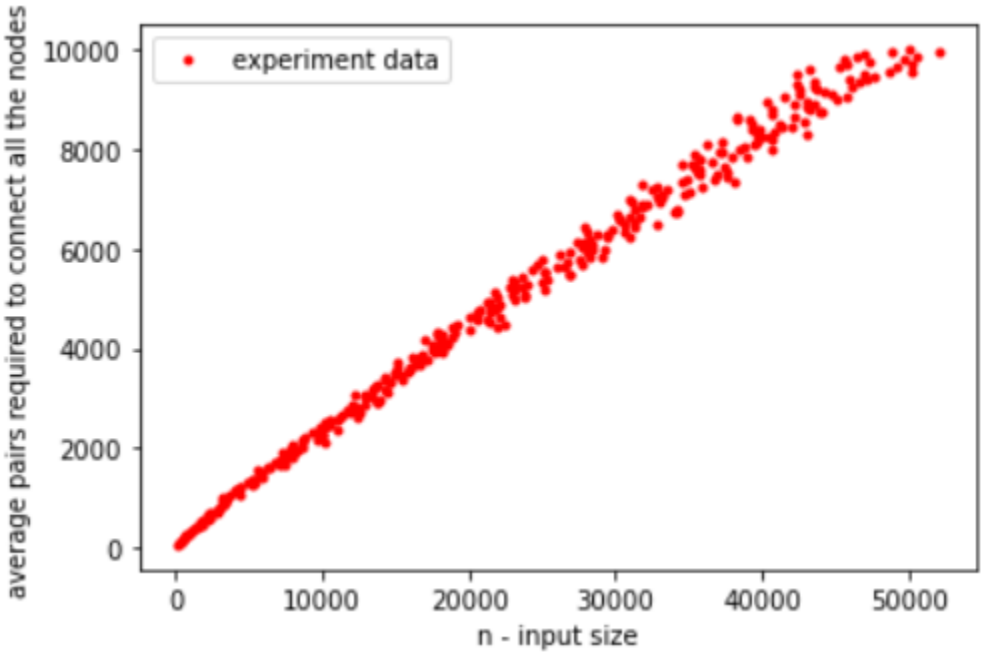


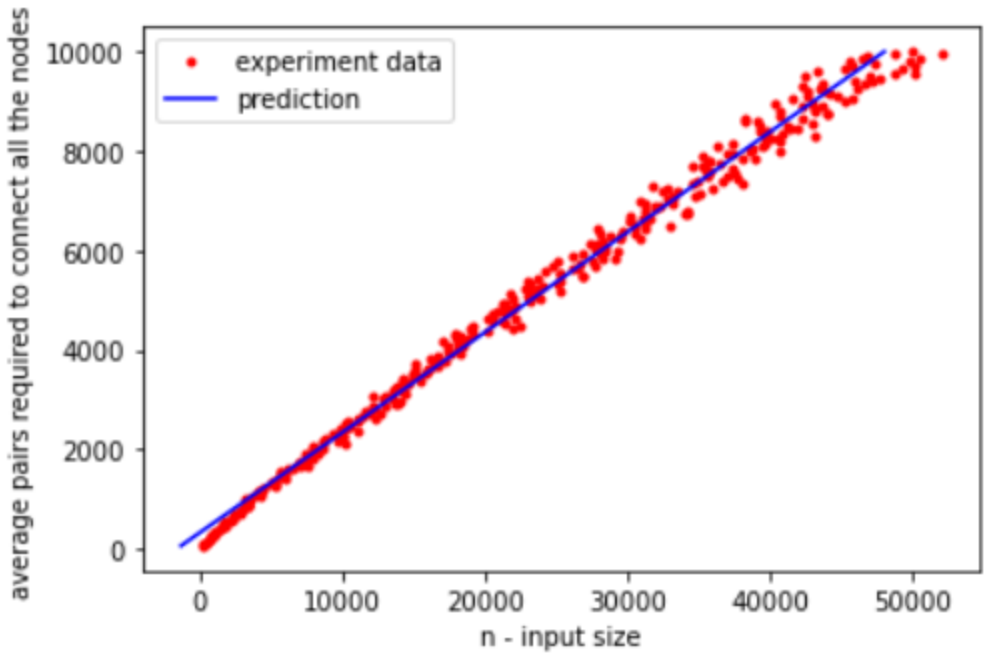
.....

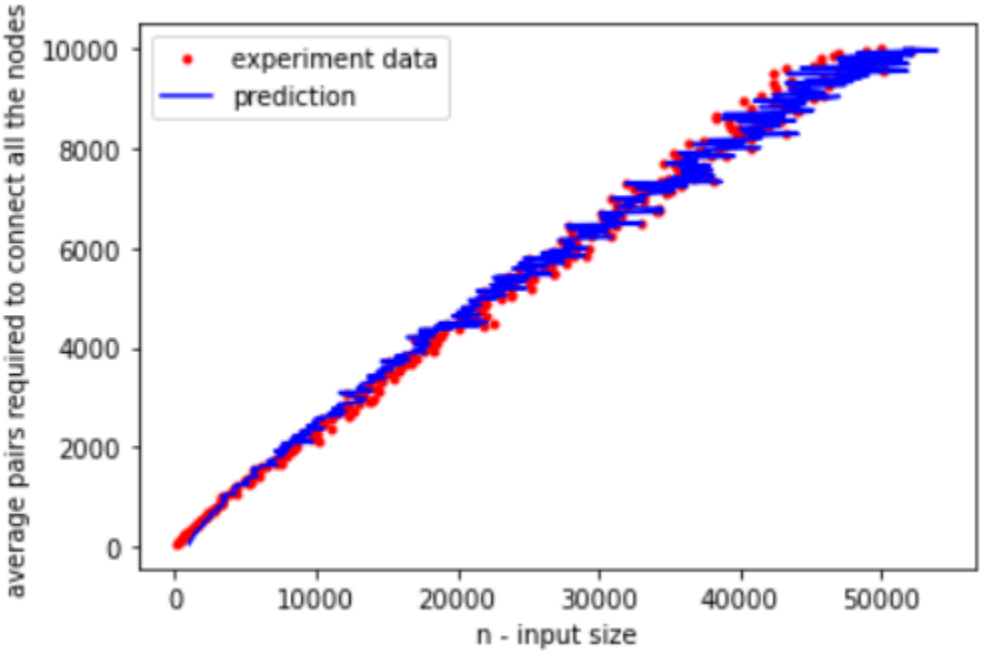
```
Number of steps n = 9694, number of unions required m = 50195.4
Number of steps n = 9724, number of unions required m = 45784.3
Number of steps n = 9754, number of unions required m = 47330.9
Number of steps n = 9784, number of unions required m = 49748.8
Number of steps n = 9814, number of unions required m = 45638.3
Number of steps n = 9844, number of unions required m = 46525.4
Number of steps n = 9874, number of unions required m = 50509.1
Number of steps n = 9904, number of unions required m = 46921.9
Number of steps n = 9934, number of unions required m = 48843.1
Number of steps n = 9964, number of unions required m = 52037.7
Number of steps n = 9994, number of unions required m = 49968.8

Process finished with exit code 0
```

- Created a jupyter notebook and performed analysis on the output (csv format)



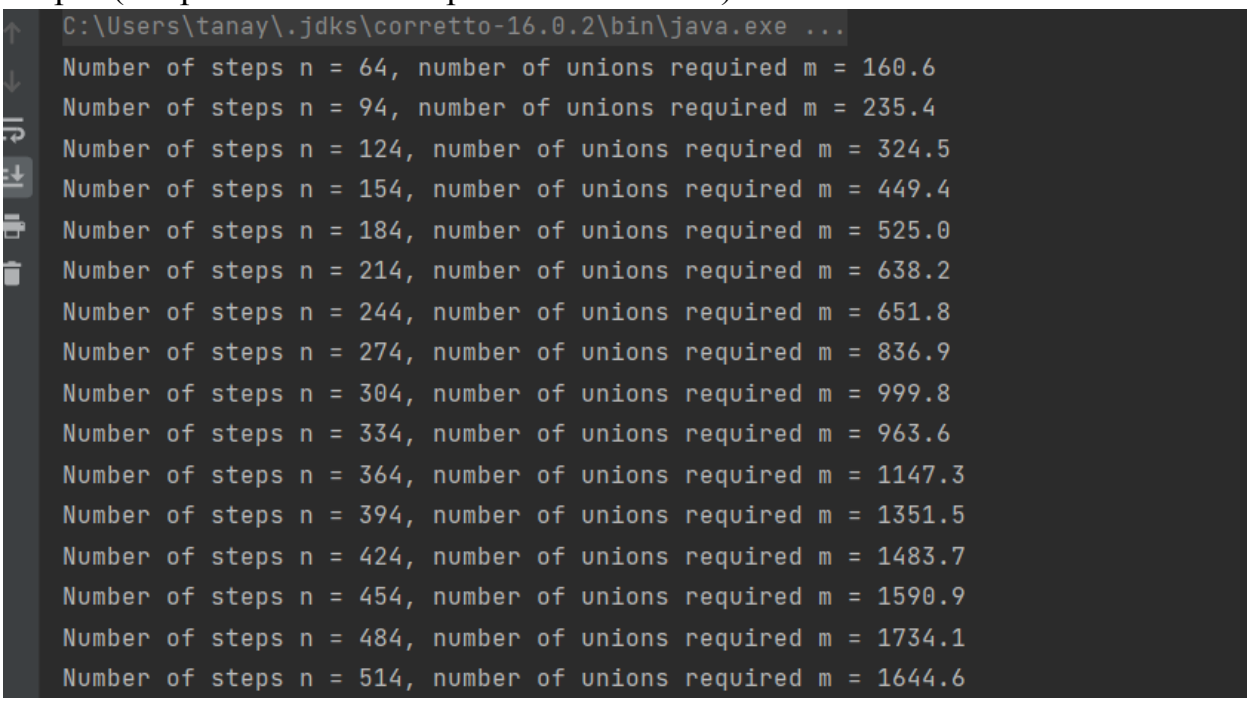


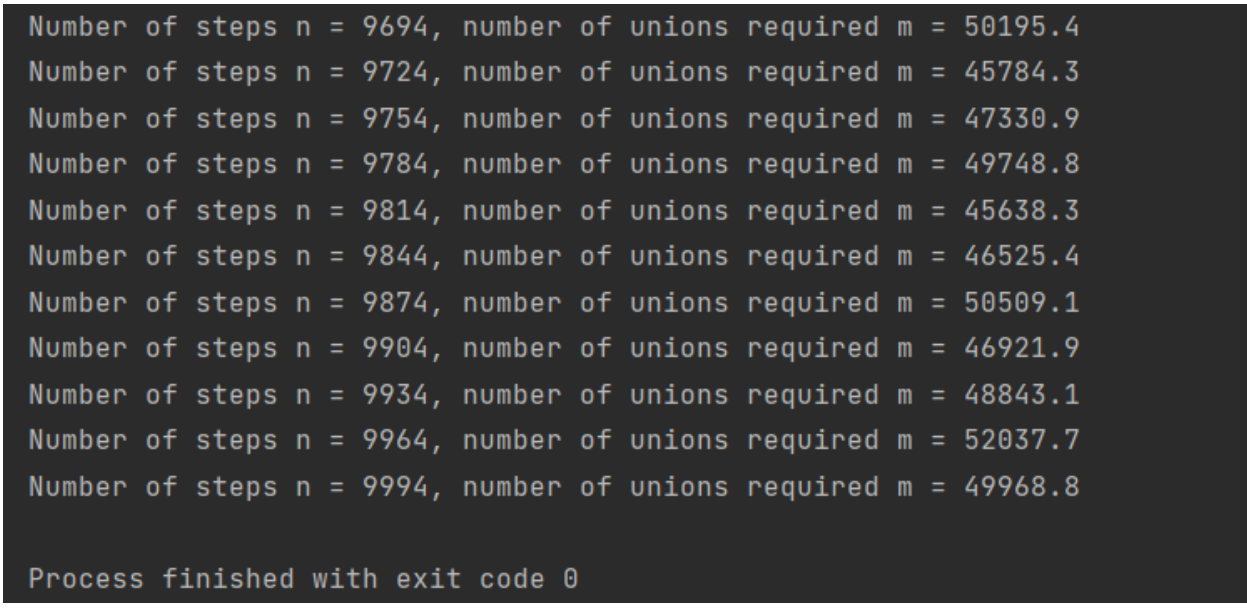


- Relationship Conclusion:
  - The graph appears to be linearithmic,
    $M = a * N \lg(N) + b$
  - N: input size
  - M: average number of pairs
  - lg: log base 2
  - a & b are constants

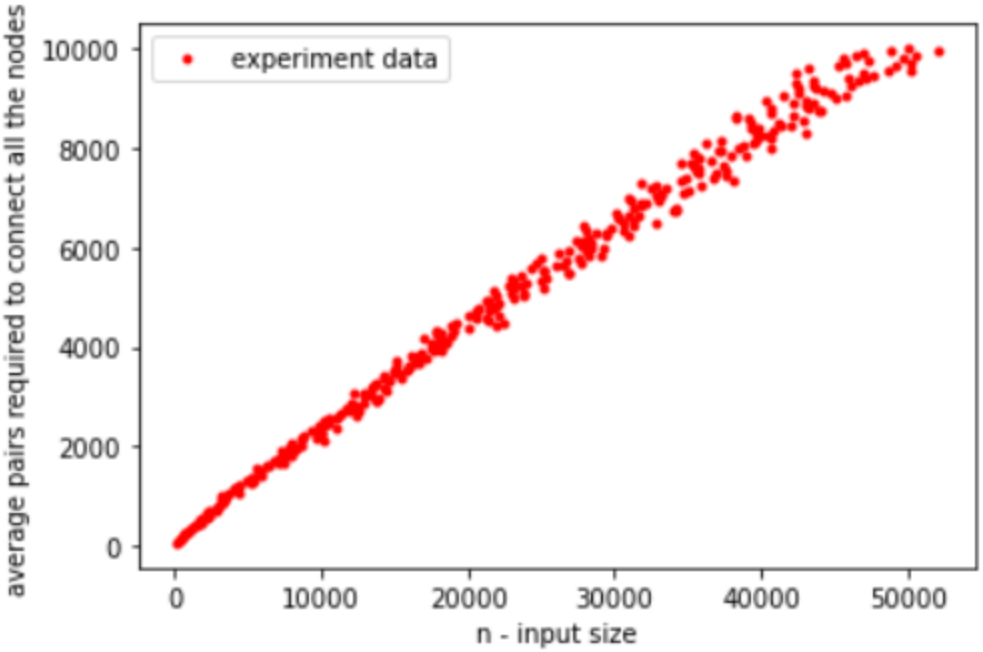
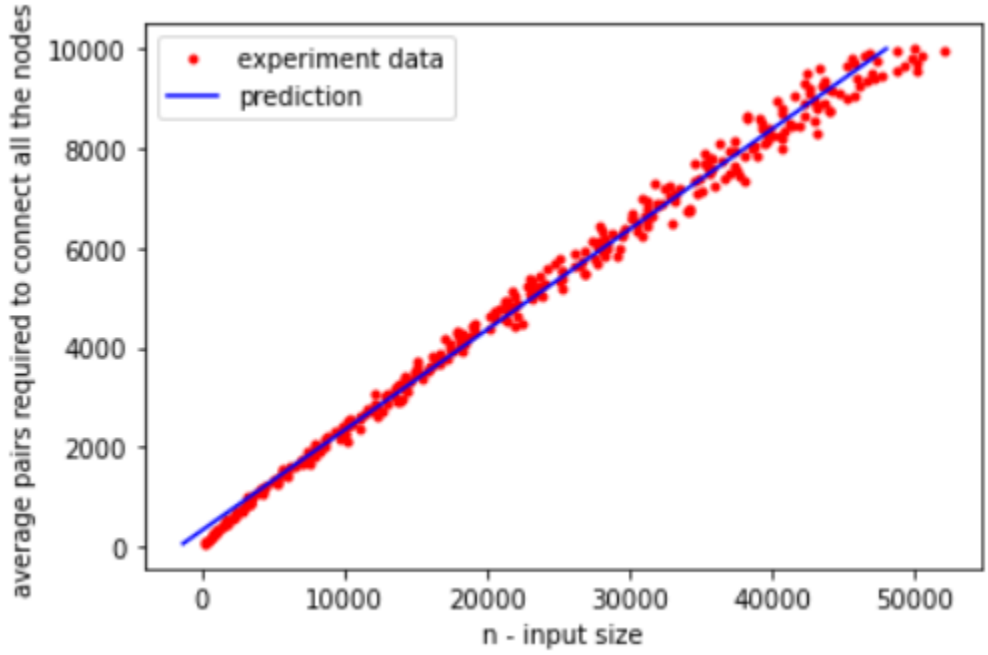
- The detailed experiment can be found here

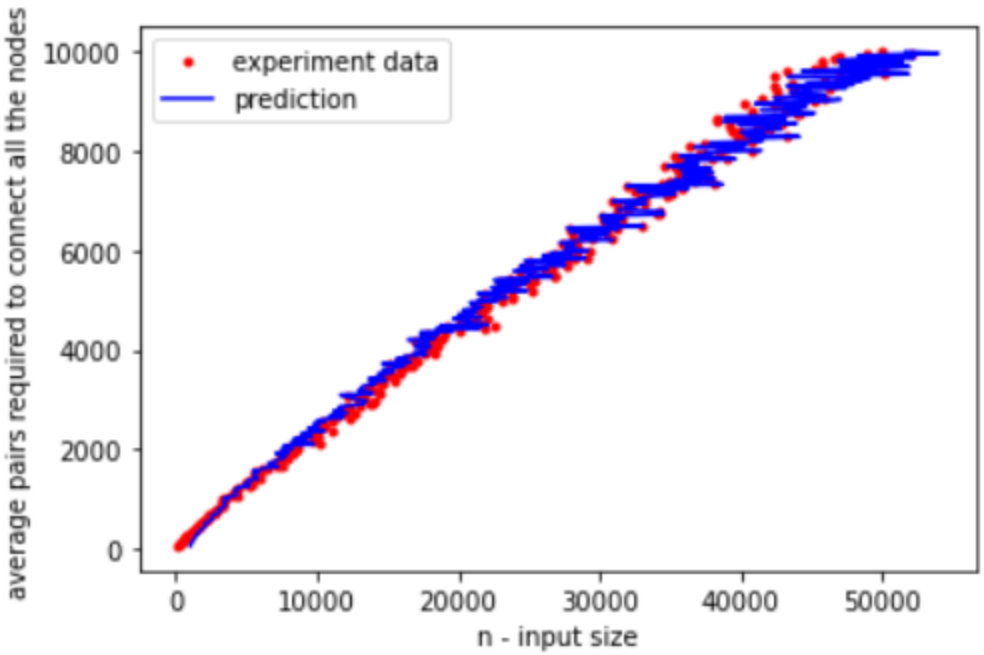
- Evidence to support the conclusion:
  1. Output (Snapshot of Code output in the terminal)





  2. Graphical Representation(Observations)

- 
experiment data
average pairs required to connect all the nodes
n - input size
0
2000
4000
6000
8000
10000
0
10000
20000
30000
40000
50000
- 
experiment data
prediction
average pairs required to connect all the nodes
n - input size
0
2000
4000
6000
8000
10000
0
10000
20000
30000
40000
50000



○

3. Unit tests result:(Snapshot of successful unit test run)