

## Assignment 3 (WQUPC)

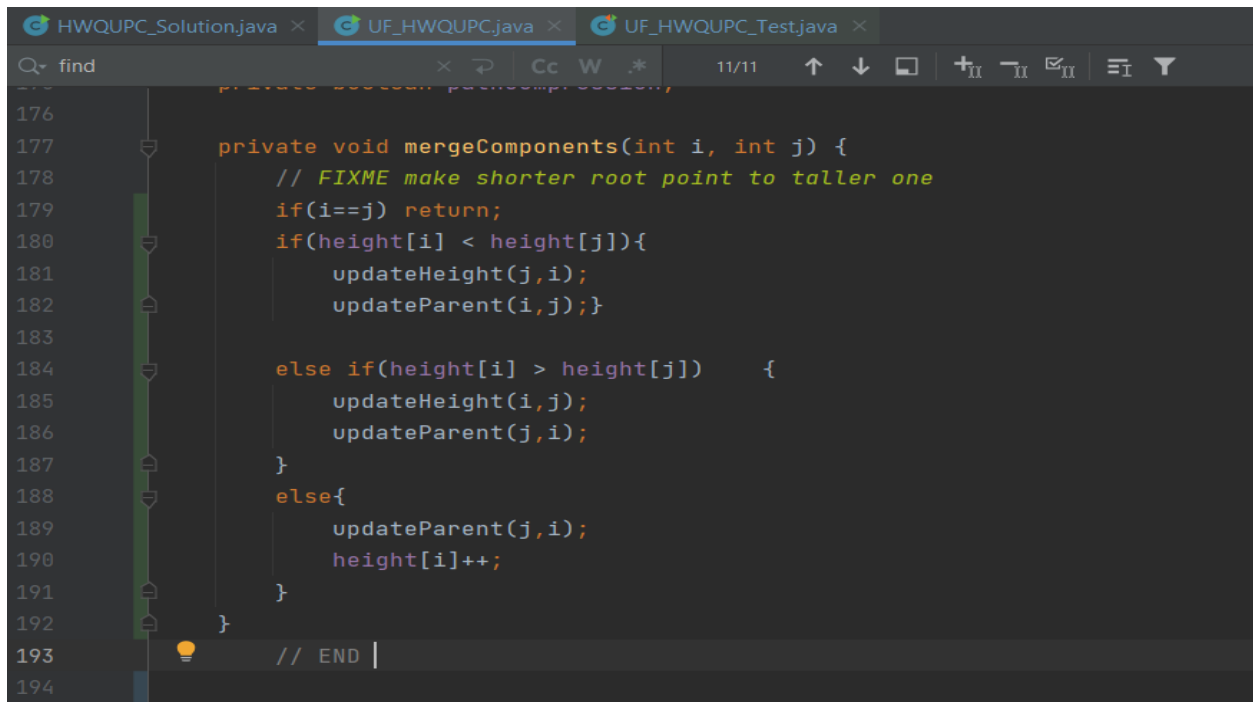
Name : Ganesh Dharani

Nuid : 002191922

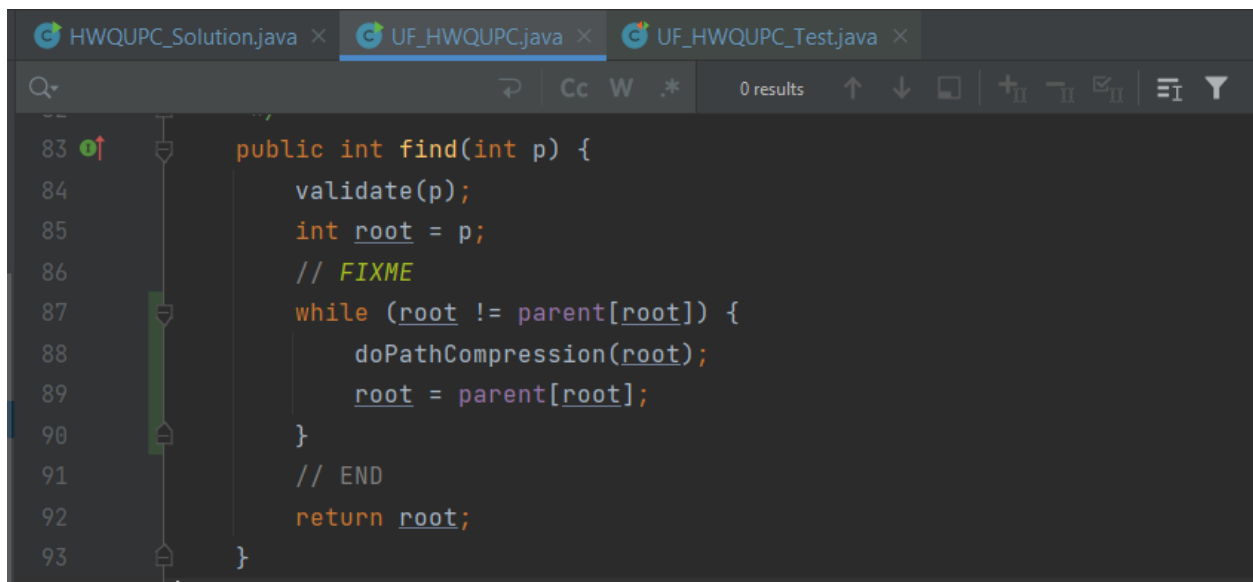
Step 1:

(a) Implement height-weighted Quick Union with Path Compression.

Done the code and ran all the test cases for height weighted quick union with path compression.



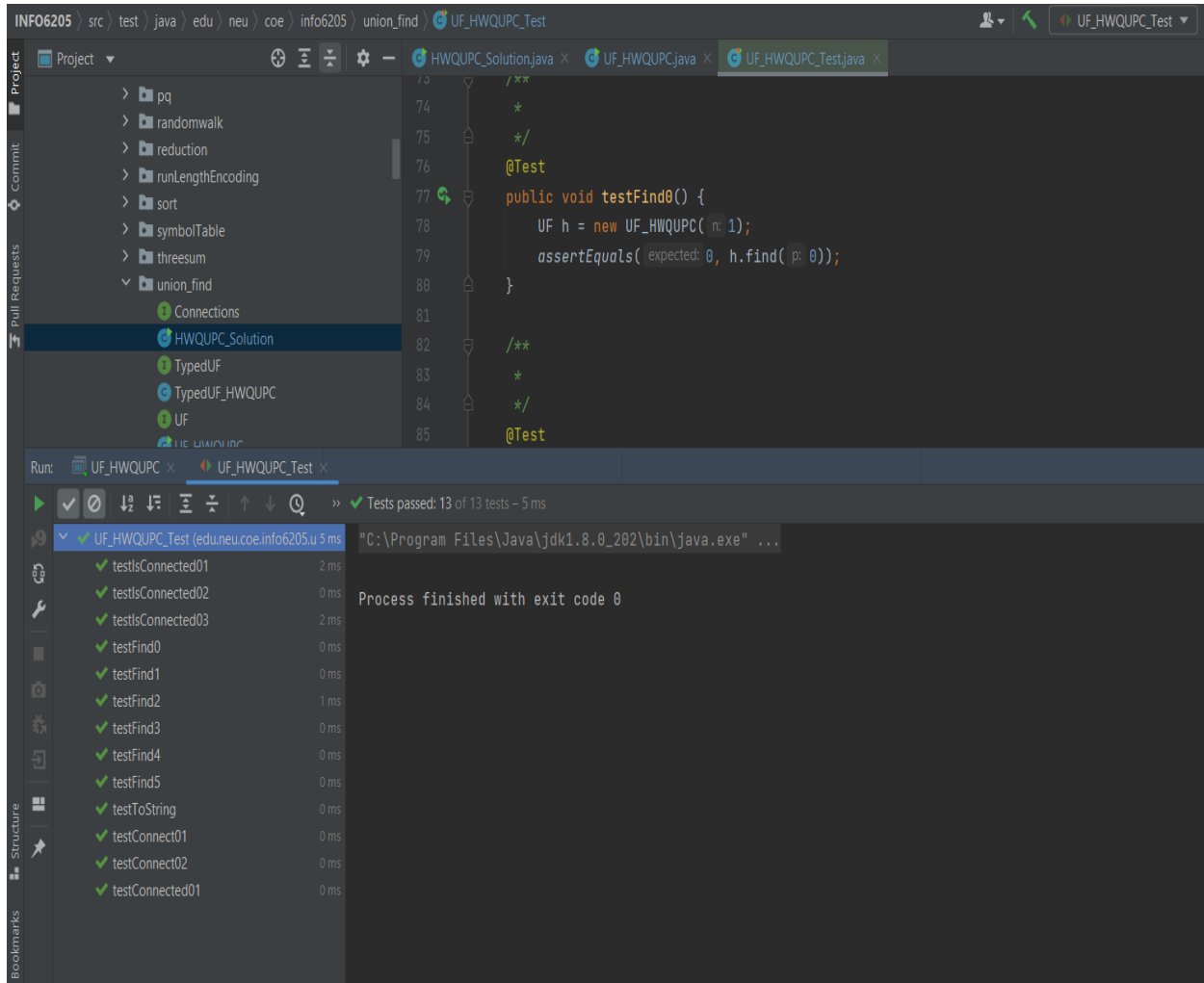
```
HWQUPC_Solution.java x UF_HWQUPC.java x UF_HWQUPC_Test.java x
Q find x ↻ Cc W .* 11/11 ↑ ↓ ↵ +II -II ☑ II ≡ I Y
176
177     private void mergeComponents(int i, int j) {
178         // FIXME make shorter root point to taller one
179         if(i==j) return;
180         if(height[i] < height[j]){
181             updateHeight(j,i);
182             updateParent(i,j);}
183
184         else if(height[i] > height[j]) {
185             updateHeight(i,j);
186             updateParent(j,i);
187         }
188         else{
189             updateParent(j,i);
190             height[i]++;
191         }
192     }
193     // END |
194
```



```
HWQUPC_Solution.java x UF_HWQUPC.java x UF_HWQUPC_Test.java x
Q 0 results ↻ Cc W .* 0 results ↑ ↓ ↵ +II -II ☑ II ≡ I Y
83     public int find(int p) {
84         validate(p);
85         int root = p;
86         // FIXME
87         while (root != parent[root]) {
88             doPathCompression(root);
89             root = parent[root];
90         }
91         // END
92         return root;
93     }
94
```

**(b) Check that the unit tests for this class all work**

The test cases have been passed successfully as mentioned below:



The screenshot displays an IDE interface with a project named 'INFO6205'. The project structure on the left includes folders like 'pq', 'randomwalk', 'reduction', 'runLengthEncoding', 'sort', 'symbolTable', 'threesum', and 'union\_find'. The 'union\_find' folder is expanded, showing files: 'Connections', 'HWQUPC\_Solution', 'TypedUF', 'TypedUF\_HWQUPC', 'UF', and 'UF\_HWQUPC'. The 'UF\_HWQUPC\_Test' file is selected, showing a Java test class with the following code:

```
73  /**
74  *
75  */
76  @Test
77  public void testFind0() {
78      UF h = new UF_HWQUPC( n: 1);
79      assertEquals( expected: 0, h.find( p: 0));
80  }
81
82  /**
83  *
84  */
85  @Test
```

The 'Run' tab at the bottom shows the execution results for 'UF\_HWQUPC\_Test'. It indicates that 13 of 13 tests passed in 5 ms. The command used for running the tests is: "C:\Program Files\Java\jdk1.8.0\_202\bin\java.exe" ...

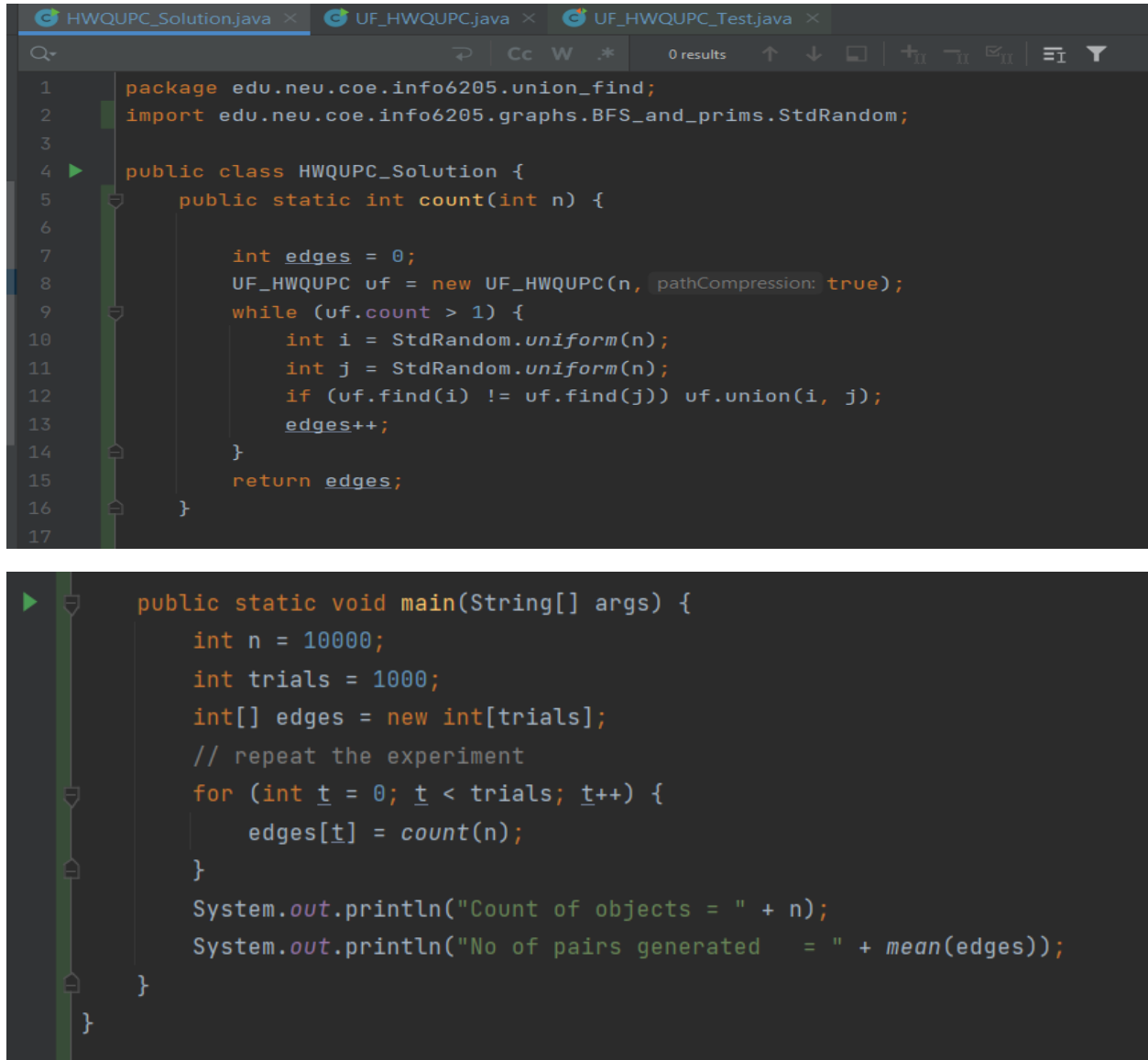
Test Case	Duration
testIsConnected01	2 ms
testIsConnected02	0 ms
testIsConnected03	2 ms
testFind0	0 ms
testFind1	0 ms
testFind2	1 ms
testFind3	0 ms
testFind4	0 ms
testFind5	0 ms
testToString	0 ms
testConnect01	0 ms
testConnect02	0 ms
testConnected01	0 ms

Process finished with exit code 0

## Step 2:

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.

Done the code and created `count()` which takes `n` as the argument and returns the number of connections. And prints the returned value.



```
HWQUPC_Solution.java  UF_HWQUPC.java  UF_HWQUPC_Test.java
1 package edu.neu.coe.info6205.union_find;
2 import edu.neu.coe.info6205.graphs.BFS_and_prims.StdRandom;
3
4 public class HWQUPC_Solution {
5     public static int count(int n) {
6
7         int edges = 0;
8         UF_HWQUPC uf = new UF_HWQUPC(n, pathCompression: true);
9         while (uf.count > 1) {
10             int i = StdRandom.uniform(n);
11             int j = StdRandom.uniform(n);
12             if (uf.find(i) != uf.find(j)) uf.union(i, j);
13             edges++;
14         }
15         return edges;
16     }
17
18     public static void main(String[] args) {
19         int n = 10000;
20         int trials = 1000;
21         int[] edges = new int[trials];
22         // repeat the experiment
23         for (int t = 0; t < trials; t++) {
24             edges[t] = count(n);
25         }
26         System.out.println("Count of objects = " + n);
27         System.out.println("No of pairs generated = " + mean(edges));
28     }
29 }
```

```

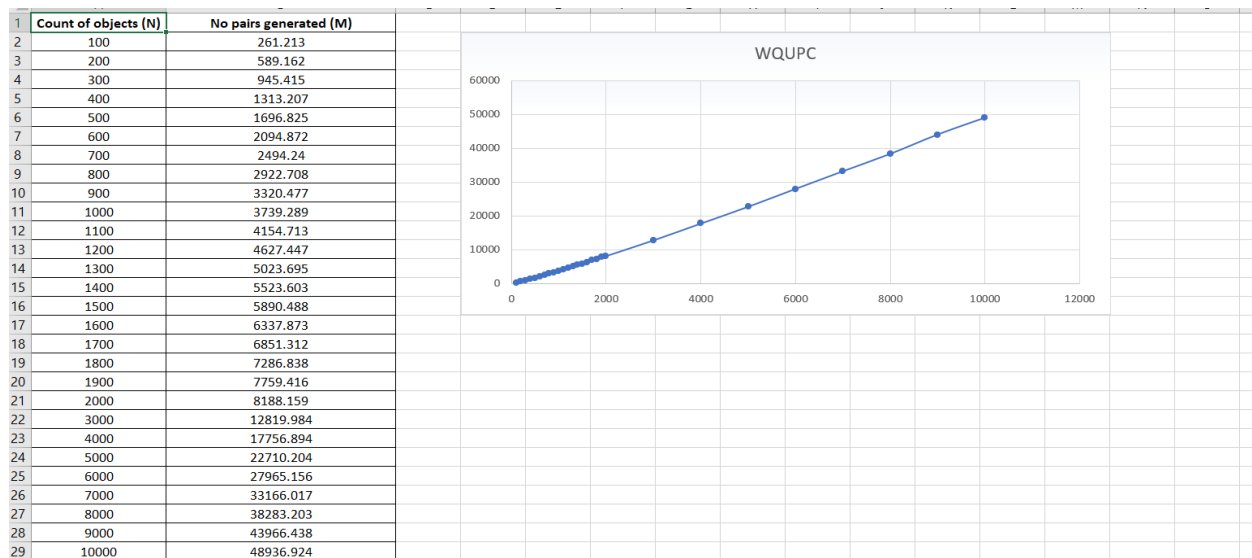
public static void main(String[] args) {
    Scanner StdIn = new Scanner(System.in);
    int n = StdIn.nextInt();
    UF_HWQUPC uf = new UF_HWQUPC(n, pathCompression: true);
    while (StdIn.hasNext()) {
        int p = (StdIn.nextInt());
        int q = (StdIn.nextInt());
        if (uf.find(p) == uf.find(q)) continue;
        uf.union(p, q);
        System.out.println(p + " " + q);
    }
    System.out.println(uf.count + " connections");
}

```

### Step 3:

Determine the relationship between the number of objects ( $n$ ) and the number of pairs ( $m$ ).

For the number of objects ( $n$ ) and the number of pairs( $m$ ) the relationship has been generated as shown in the below graph.



### Conclusion:

With the derived graph with the plotting points. The final conclusion regarding the relation is as getting as below.

$$\text{Number of Pairs} = (N \cdot \log N) / 2, \text{ where } N = \text{Number of Objects}$$

