Name: Namrata Ruchandani
NU ID : 002125637

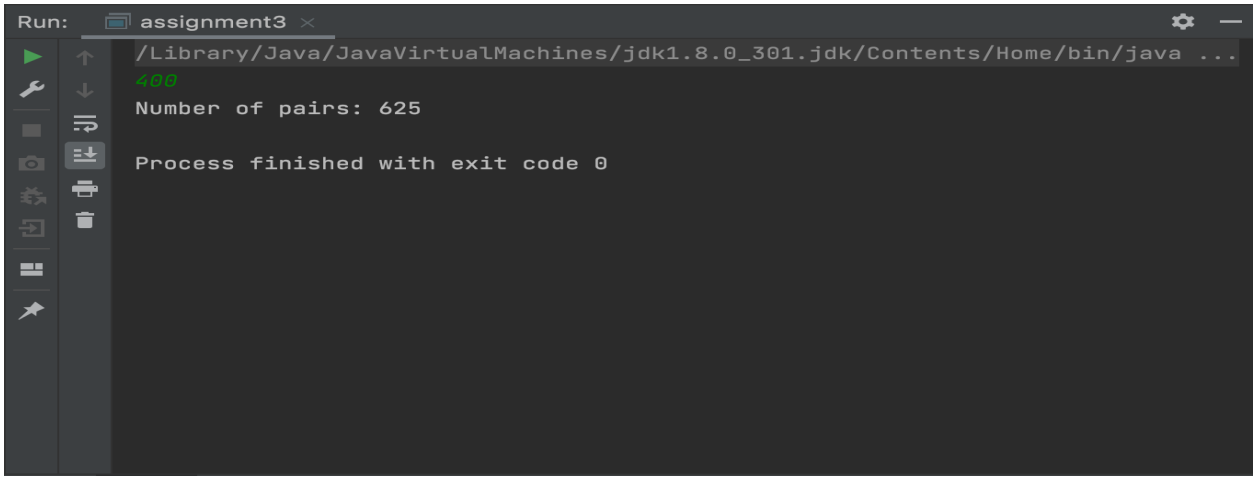
➔ Task done in this assignment:

1. In the find method, I'm checking pathCompression first. If yes then I'm calling dopathCompression by passing root, and updating parent to grandparent of root. If there is no pathCompression, I'm just pointing to the parent of root not grandparent. Finally, returning root.
2. In the mergeComponents method, I'm comparing the height of two roots to be merged. Making root with larger size as parent of root with smaller size. If the sizes are similar, attach them and increase height by 1.
3. In the doPathCompression method, I'm pointing to grandparent instead of parent, and thus doing pathCompression.
4. In the file name assignment3.java, I took input by using scanner command and then made an object **uf** of UF_HWQUPC class. I stored the value of the number of components after each union in temp. I made a while loop with a condition when temp is not equal to 1, as when all components are connected, the number of components will be one. Generated p and q random numbers and then checked if connected or not. If not, I did union. Number of times random pairs are generated is counted by count. Hence, printing count as the number of pairs generated.

➔ The screenshot of main method (or part 2 of assignment) assignment3.java code compilation and output

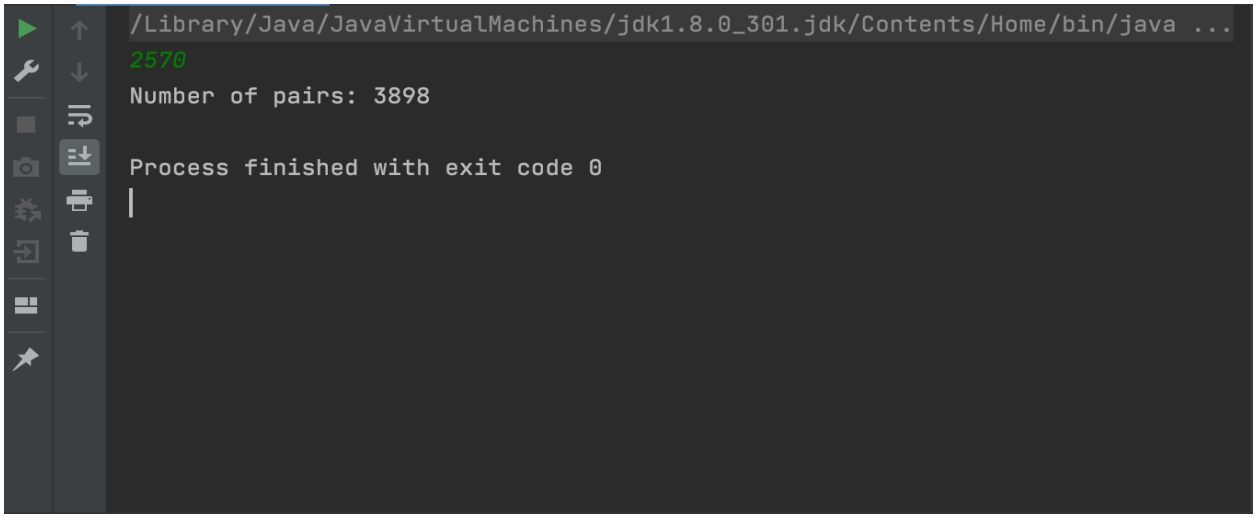
assignment3.java

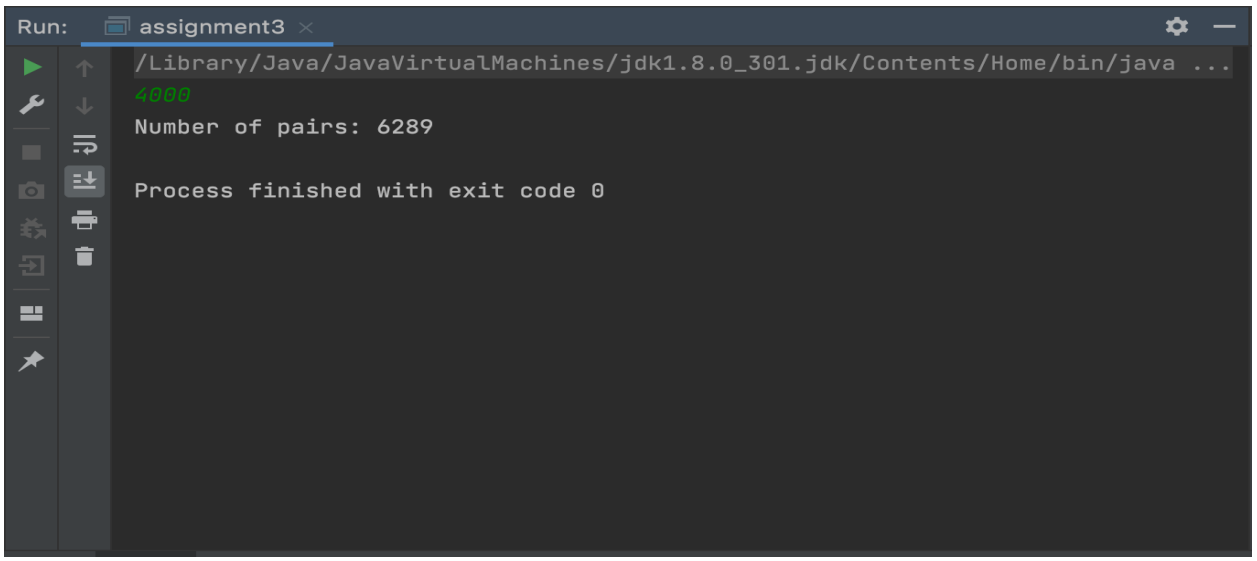
```java
package edu.neu.coe.info6205.union_find;
import java.util.Scanner;
public class assignment3
{
    public static void main(String[] args)
    {
        int p, q;
        Scanner ip = new Scanner(System.in);
        long N = ip.nextLong();
        UF_HWQUPC uf = new UF_HWQUPC((int) N);
        int count=0;
        int temp= uf.components();

        while(temp!=1)
        {
            p = (int) (Math.random() * (N - 1));
            q = (int) (Math.random() * (N - 1));

            if (!uf.connected(p, q))
            {
                uf.union(p, q);
            }
            count++;
            temp= uf.components();
        }
        System.out.println("Number of pairs: "+ count);
    }
}
```


Run:
assignment3
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
400
Number of pairs: 625
Process finished with exit code 0

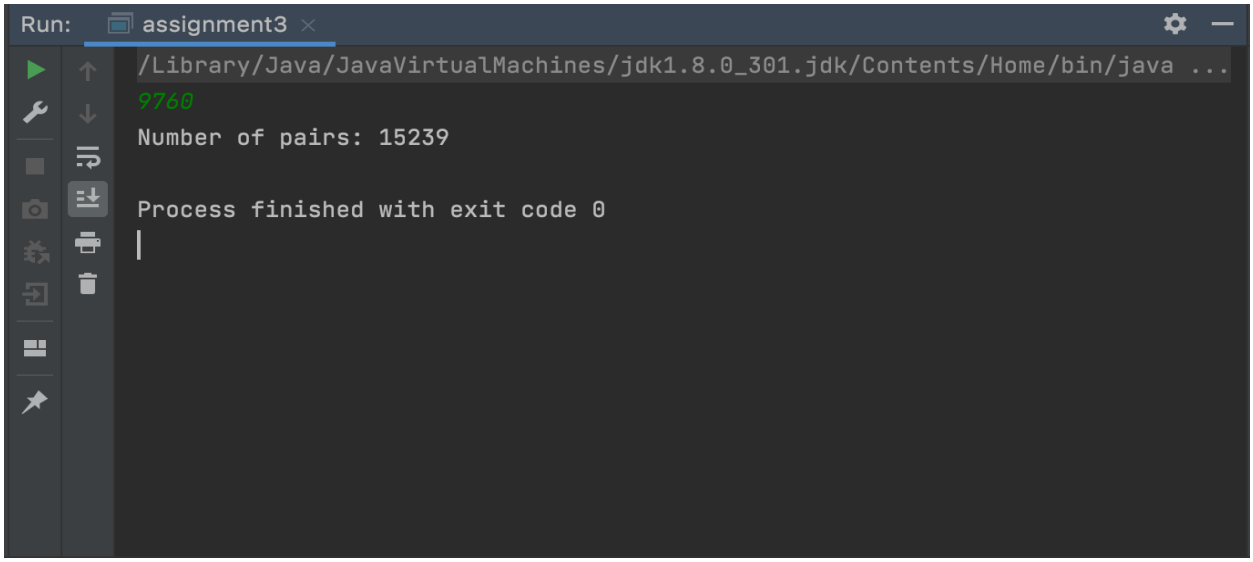

Run:
assignment3
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
750
Number of pairs: 1135
Process finished with exit code 0


/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
1500
Number of pairs: 2290
Process finished with exit code 0


/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
2570
Number of pairs: 3898
Process finished with exit code 0


Run:
assignment3
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
4000
Number of pairs: 6289
Process finished with exit code 0


Run:
assignment3
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
8300
Number of pairs: 12871
Process finished with exit code 0

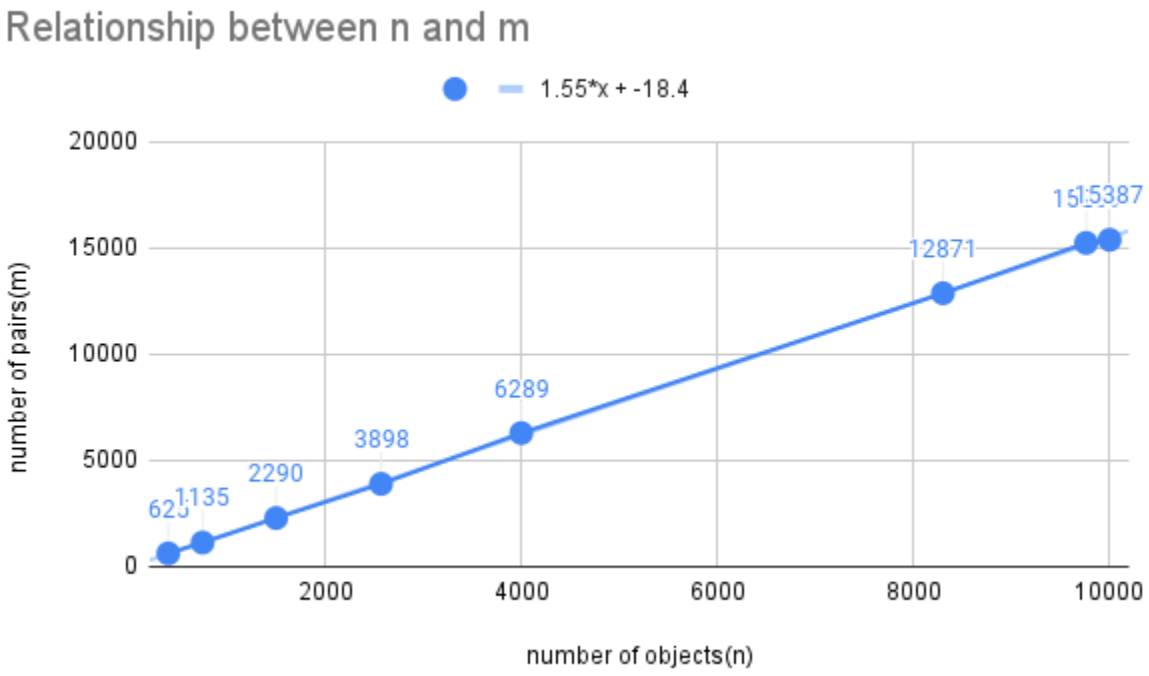


➔ **The link for file assignment3.java :**
https://github.com/Namrata2108/INFO6205/blob/Fall2021/src/main/java/edu/neu/coe/info6205/union_find/assignment3.java

➔ **Conclusion about number of objects(n) and number of pairs(m)**

Ideally, there should be n-1 connections to make one component in the end. But, as we are generating random pairs then repeated cases can occur. This is justified after seeing that the output of the number of pairs is more than n-1. The graph is linear showing the relation is (**slope*n + c),** where c is constant,i.e, -18.4 and slope is 1.55(from the equation generated by excel). I think, constant is added because of repeated pairs generated.

number of objects(n)	number of pairs (m)
400	625
750	1135
1500	2290
2570	3898
4000	6289
8300	12871
9760	15239
10000	15387



➔ **I pushed the commands in the git and attached the link for the repository.**
https://github.com/Namrata2108/INFO6205/tree/Fall2021/src/main/java/edu/neu/coe/info6205/union_find

➔ **Below is the screenshot of successful test cases run**