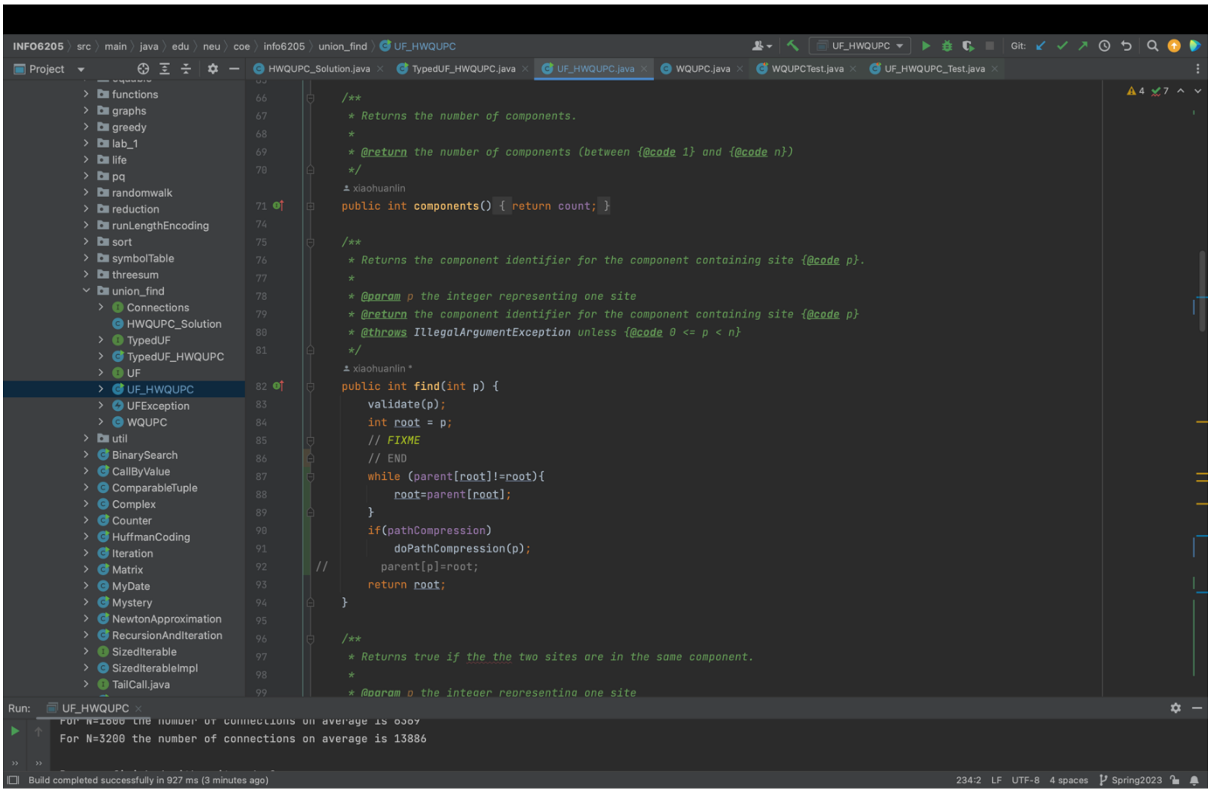
Program Structures and Algorithms
Spring 2023(SEC – 01)

NAME: NEHA RASTOGI
NUID: 002709191
ASSIGNMENT: 4

Task: The assignment has 3 steps to it
Step 1: Implement height-weighted Quick Union with Path Compression and ensure unit tests work.
Step 2: Then, develop a UF client to find number of random connections generated to create a connected graph having n objects.
Step 3: Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated.

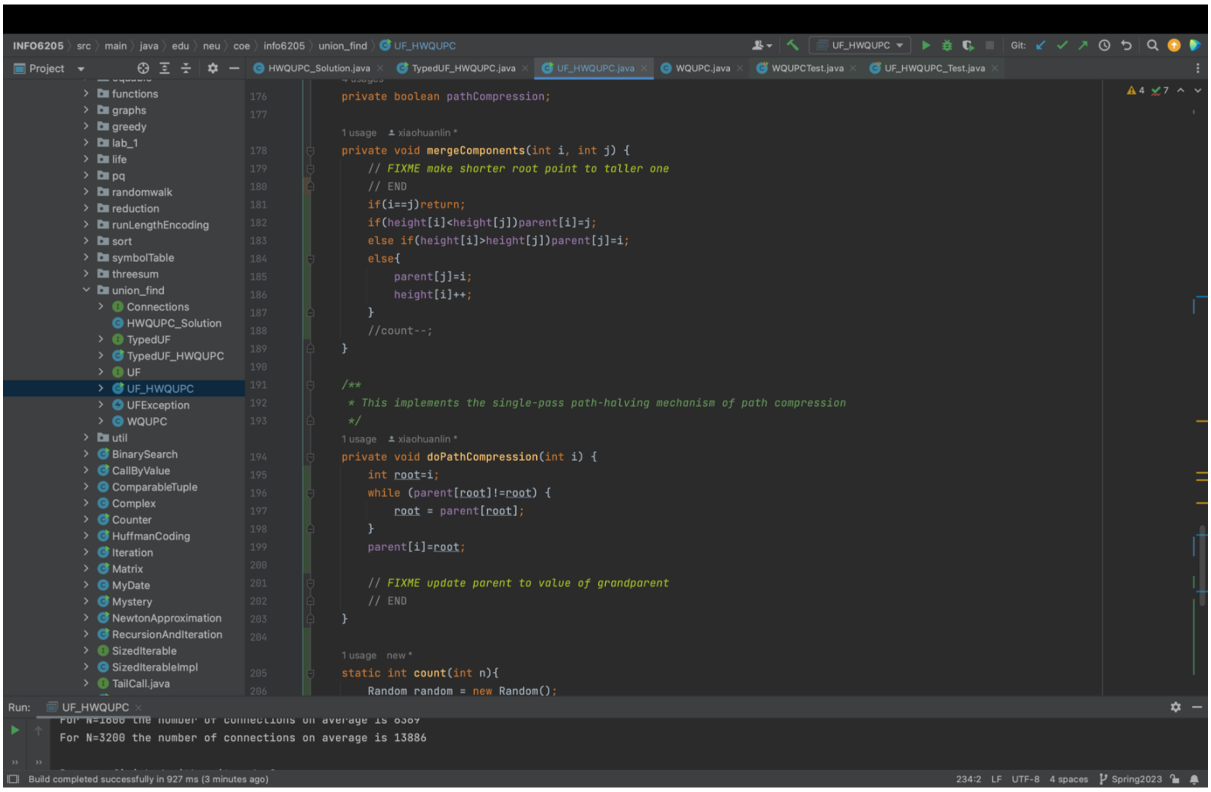
Relationship Conclusion: For this assignment the task at hand was to accept the number of sites and then generate random pairs of edges and check if they were connected or not. If not, the components must be merged, and count of components reduced. This is repeated until all sites form a connected graph, that is, 1 connected component. In this process we can establish a relation between n, the number of sites and m, the number of random pairs of edges. Without any randomness introduced in this equation, to have a connected graph the value of **m >= n-1.** Based on the linear graph obtained on plotting the values of m and n we see that it follows the equation y=mx+c and on substitution we get **m=4.35n.** The value of the slope can be taken as approximately 4 so m~4n. For any linear graph, a positive slope implies a positive relation between the two variables. **As one increases, so does the other one**. Here, as the value of n is increased, we see an increase in m almost 4 times. So, the more nodes in the graph, the greater number of random pairs we need to generate to create a connected graph.
As per the values of m that we get I observed that as the number of pairs having duplicates might increase and additionally the number of already connected pairs recurring might also increase. With more options available between 0 and n-1 to pick a random value the number of combinations made also increases and so does the number of combinations required for a connected graph. Since we are also considering duplicates this value of possible combinations would reach the order of N^2. The graph for n against m is a linear graph since both values are increasing

Evidence to support that conclusion:
Step 1:



```
    /**
     * Returns the number of components.
     *
     * @return the number of components (between {@code 1} and {@code n})
     */
    xiaohuanlin
    public int components() { return count; }

    /**
     * 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}
     */
    xiaohuanlin *
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        // END
        while (parent[root]!=root){
            root=parent[root];
        }
        if(pathCompression)
            doPathCompression(p);
//          parent[p]=root;
        return root;
    }

    /**
     * Returns true if the the two sites are in the same component.
     *
     * @param p the integer representing one site
```

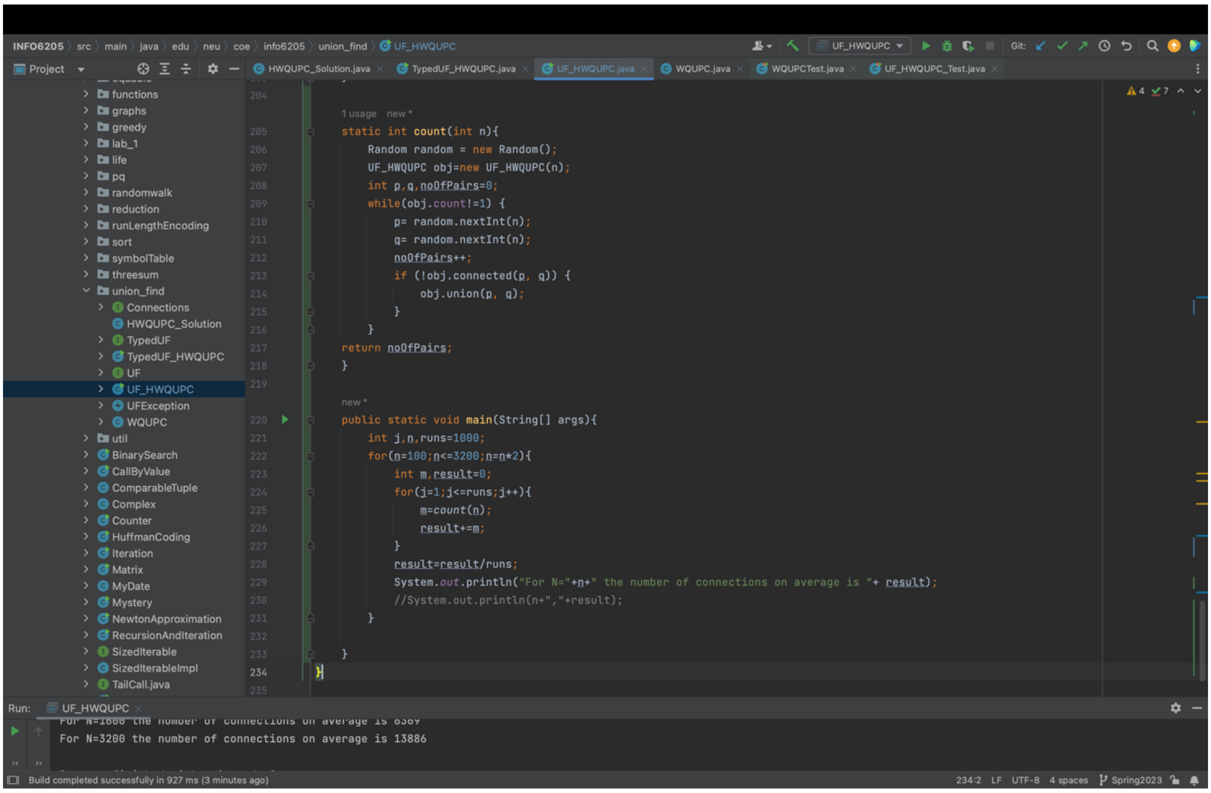
```
For N=1600 the number of connections on average is 6369
For N=3200 the number of connections on average is 13886
```



```
    private boolean pathCompression;

    1 usage  xiaohuanlin *
    private void mergeComponents(int i, int j) {
        // FIXME make shorter root point to taller one
        // END
        if(i==j)return;
        if(height[i]<height[j])parent[i]=j;
        else if(height[i]>height[j])parent[j]=i;
        else{
            parent[j]=i;
            height[i]++;
        }
        //count--;
    }

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

        // FIXME update parent to value of grandparent
        // END
    }

    1 usage  new *
    static int count(int n){
        Random random = new Random();
```

```
For N=1600 the number of connections on average is 6369
For N=3200 the number of connections on average is 13886
```

Step 2:



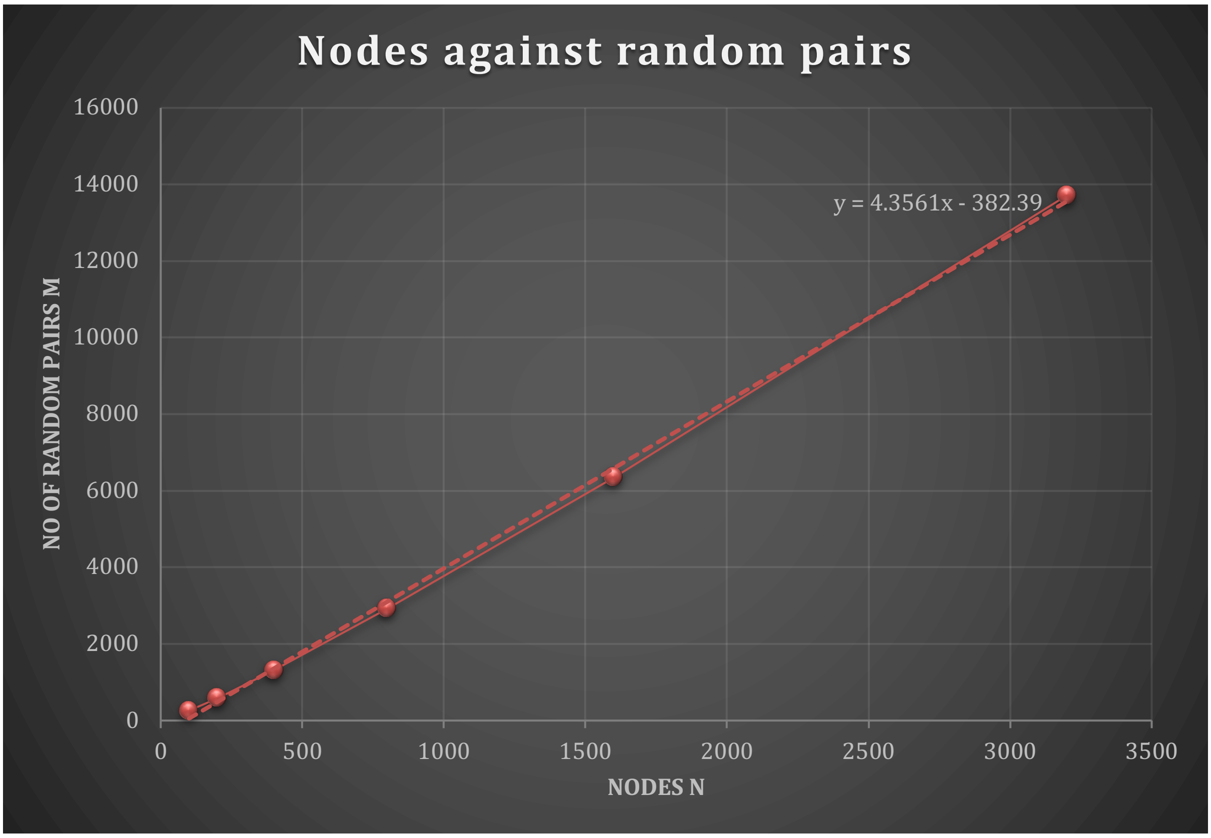
```java
static int count(int n){
    Random random = new Random();
    UF_HWQUPC obj=new UF_HWQUPC(n);
    int p,q,noOfPairs=0;
    while(obj.count!=1) {
        p= random.nextInt(n);
        q= random.nextInt(n);
        noOfPairs++;
        if (!obj.connected(p, q)) {
            obj.union(p, q);
        }
    }
return noOfPairs;
}

public static void main(String[] args){
    int j,n,runs=1000;
    for(n=100;n<=3200;n=n*2){
        int m,result=0;
        for(j=1;j<=runs;j++){
            m=count(n);
            result+=m;
        }
        result=result/runs;
        System.out.println("For N="+n+" the number of connections on average is "+ result);
        //System.out.println(n+","+result);
    }

}
}
```

```
For N=1600 the number of connections on average is 6369
For N=3200 the number of connections on average is 13886
```

Build completed successfully in 927 ms (3 minutes ago)

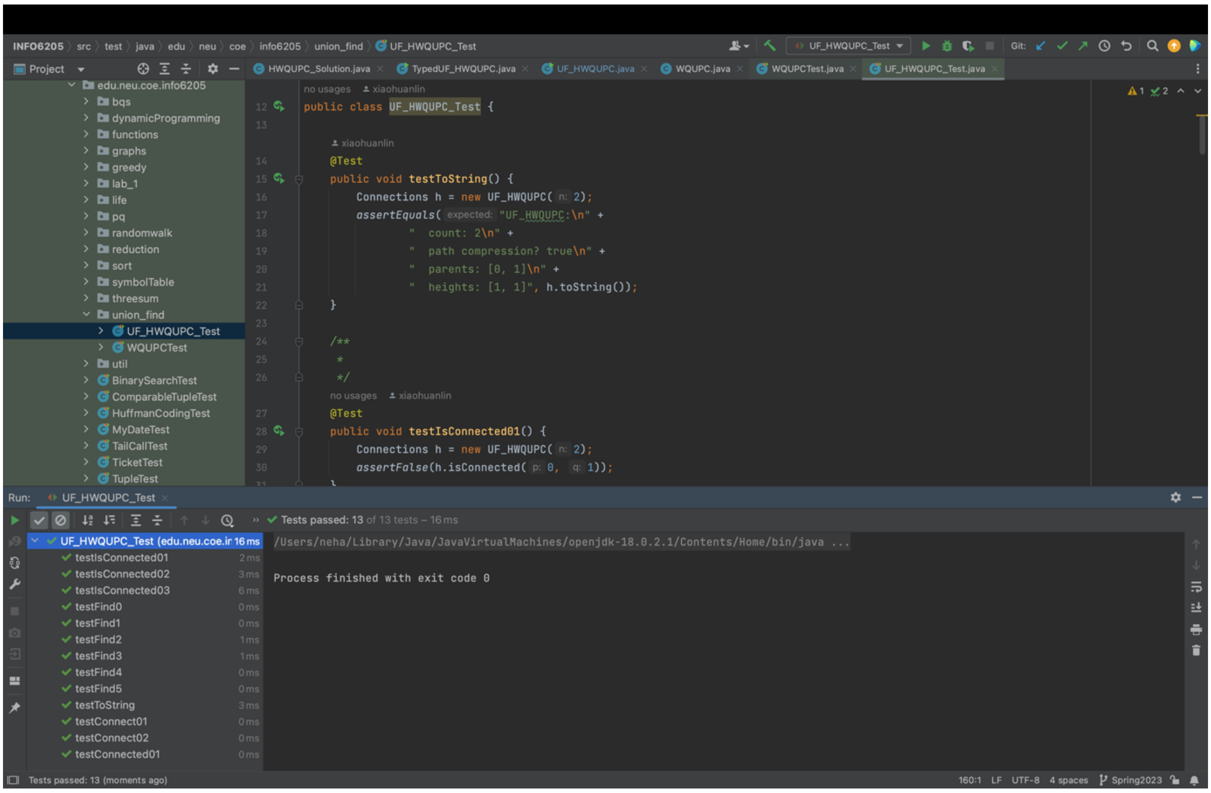
Graphical Representation:

n	m
100	261
200	587
400	1310
800	2926
1600	6353
3200	13712



In the graph shown, there is a linear graph line obtained on plotting the values of m for various n based on the doubling method. The trendline is almost superimposing on the graph line implying a perfect match. Thus, this is a **LINEAR** graph.

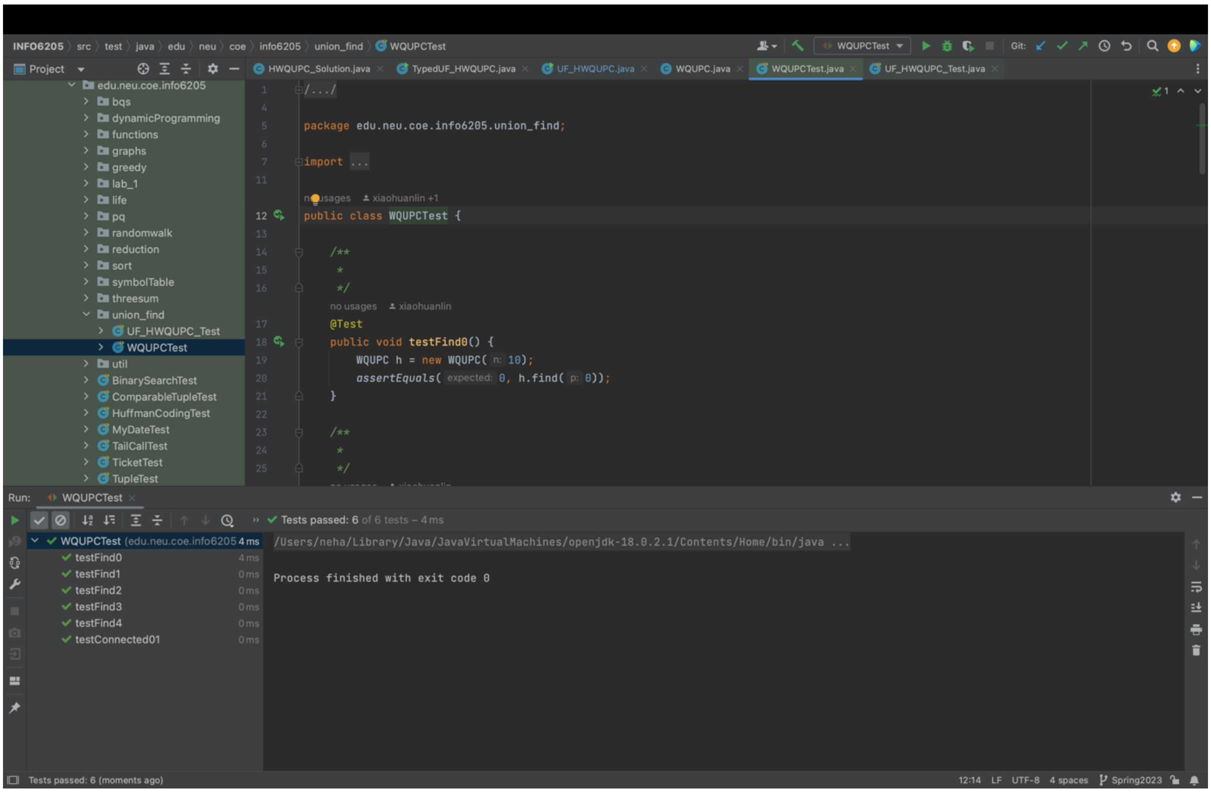
Unit Test Screenshots:
UF_HWQUPC Test



```java
public class UF_HWQUPC_Test {

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

    /**
     *
     */
    @Test
    public void testIsConnected01() {
        Connections h = new UF_HWQUPC(2);
        assertFalse(h.isConnected(0, 1));
```

```
Tests passed: 13 of 13 tests – 16 ms
UF_HWQUPC_Test (edu.neu.coe.ir 16 ms
  testIsConnected01   2 ms
  testIsConnected02   3 ms
  testIsConnected03   6 ms
  testFind0           0 ms
  testFind1           0 ms
  testFind2           1 ms
  testFind3           1 ms
  testFind4           0 ms
  testFind5           0 ms
  testToString        3 ms
  testConnect01       0 ms
  testConnect02       0 ms
  testConnected01     0 ms

/Users/neha/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java ...

Process finished with exit code 0
```

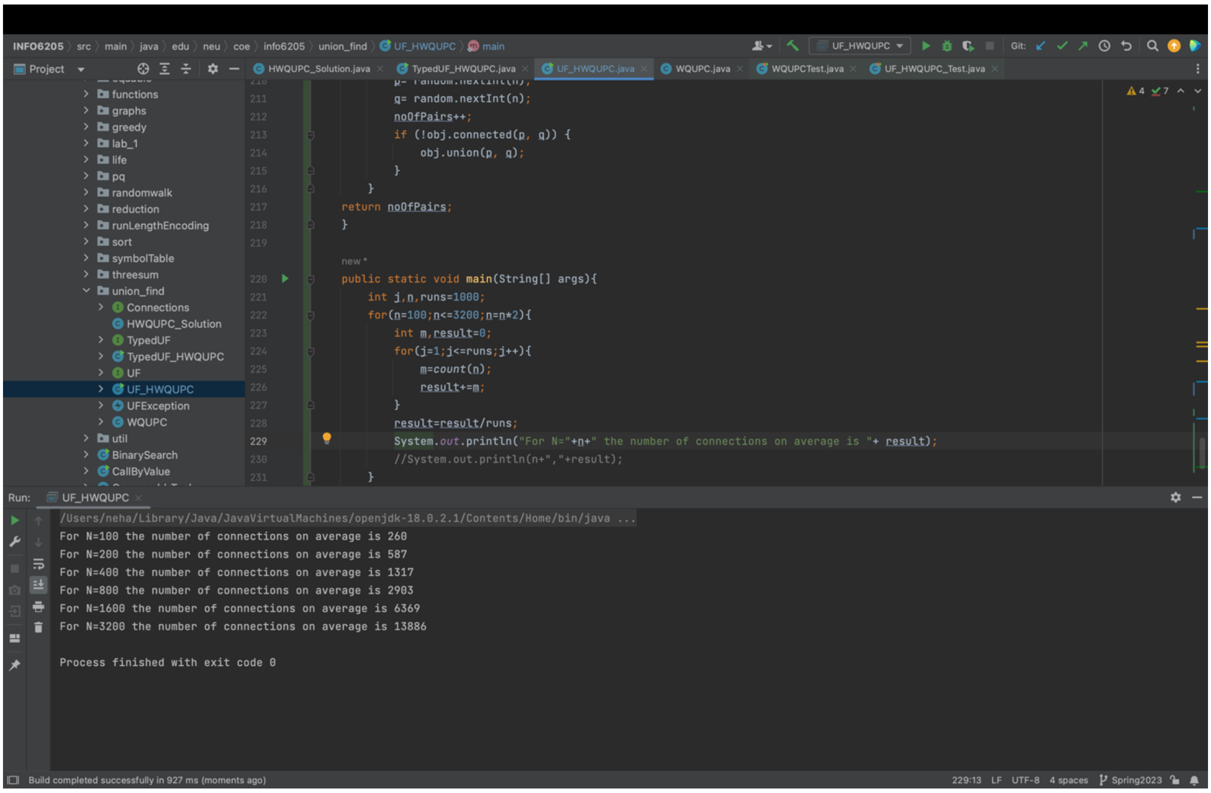
WQUPC Test



```java
/.../

package edu.neu.coe.info6205.union_find;

import ...

public class WQUPCTest {

    /**
     *
     */
    @Test
    public void testFind0() {
        WQUPC h = new WQUPC(10);
        assertEquals(0, h.find(0));
    }

    /**
     *
     */
```

```
Tests passed: 6 of 6 tests – 4 ms
WQUPCTest (edu.neu.coe.info6205 4 ms
  testFind0         4 ms
  testFind1         0 ms
  testFind2         0 ms
  testFind3         0 ms
  testFind4         0 ms
  testConnected01   0 ms

/Users/neha/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java ...

Process finished with exit code 0
```

Test for m and n



```java
                p= random.nextInt(n);
                q= random.nextInt(n);
                noOfPairs++;
                if (!obj.connected(p, q)) {
                    obj.union(p, q);
                }
            }
        return noOfPairs;
        }

        public static void main(String[] args){
            int j,n,runs=1000;
            for(n=100;n<=3200;n=n*2){
                int m,result=0;
                for(j=1;j<=runs;j++){
                    m=count(n);
                    result+=m;
                }
                result=result/runs;
                System.out.println("For N="+n+" the number of connections on average is "+ result);
                //System.out.println(n+","+result);
            }
```

```
/Users/neha/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java ...
For N=100 the number of connections on average is 260
For N=200 the number of connections on average is 587
For N=400 the number of connections on average is 1317
For N=800 the number of connections on average is 2903
For N=1600 the number of connections on average is 6369
For N=3200 the number of connections on average is 13886

Process finished with exit code 0
```