Vedant Jain
Nuid - 001305281

INFO 6205
Program Structures & Algorithms Fall 2020
Assignment No 4

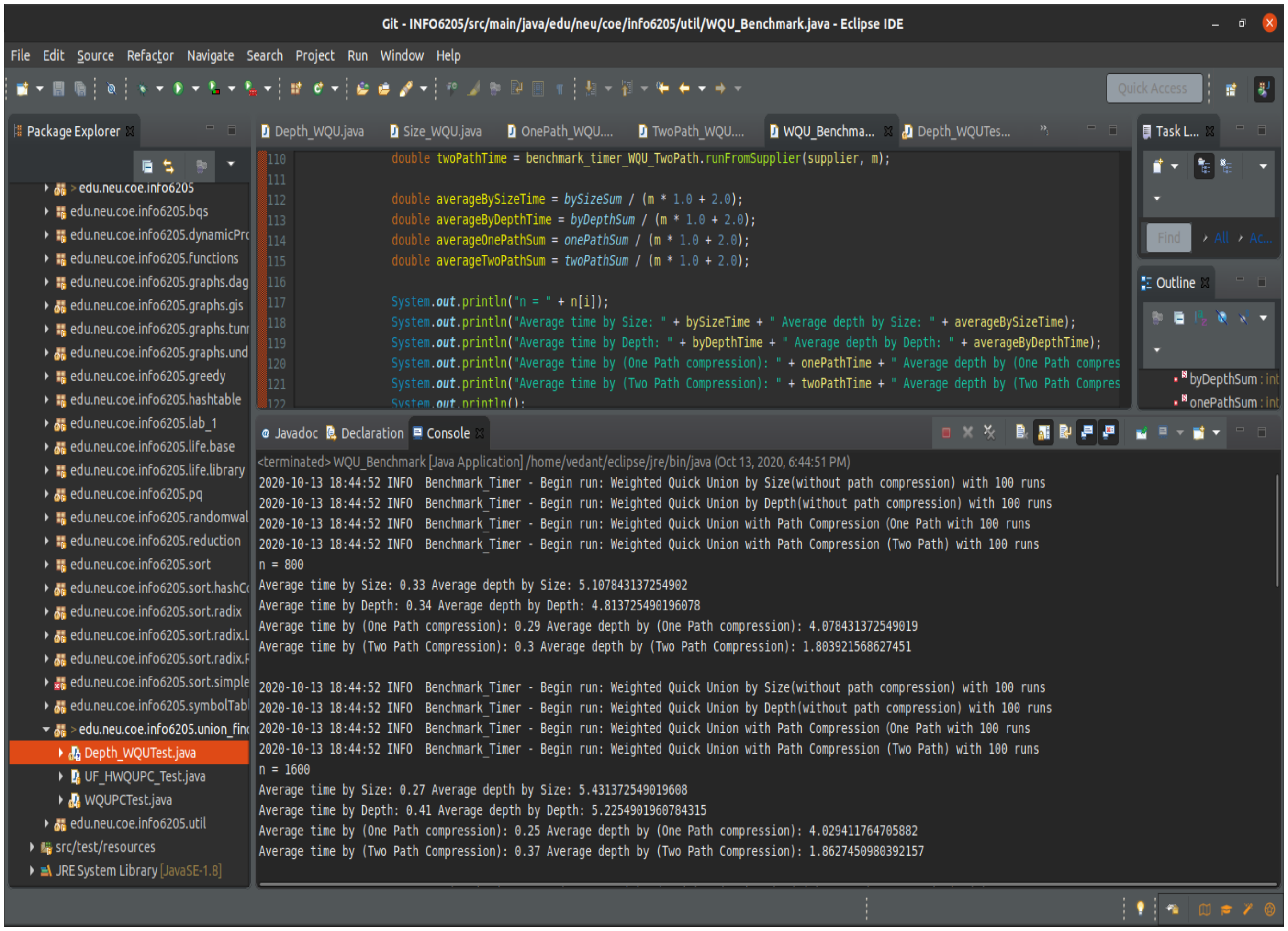
- ## Task

  Perform benchmarking for alternatives for weighted quick union and weighted quick union with path compression for:

  1. Weighted quick union by Size
  2. Weighted quick union by Depth
  3. Weighted quick union with One Pass Path compression
  4. Weighted quick union with two Pass Path compression

- ## Output:

  - For each class I used 7 different lengths:800,1600,3200,6400,128000,25600,51200 and ran it over for 100 times to get the average time of each class and average depth of each tree. The results are shown below:



```
            double twoPathTime = benchmark_timer_WQU_TwoPath.runFromSupplier(supplier, m);

            double averageBySizeTime = bySizeSum / (m * 1.0 + 2.0);
            double averageByDepthTime = byDepthSum / (m * 1.0 + 2.0);
            double averageOnePathSum = onePathSum / (m * 1.0 + 2.0);
            double averageTwoPathSum = twoPathSum / (m * 1.0 + 2.0);

            System.out.println("n = " + n[i]);
            System.out.println("Average time by Size: " + bySizeTime + " Average depth by Size: " + averageBySizeTime);
            System.out.println("Average time by Depth: " + byDepthTime + " Average depth by Depth: " + averageByDepthTime);
            System.out.println("Average time by (One Path compression): " + onePathTime + " Average depth by (One Path compres
            System.out.println("Average time by (Two Path Compression): " + twoPathTime + " Average depth by (Two Path Compres
            System.out.println();
```

```
<terminated> WQU_Benchmark [Java Application] /home/vedant/eclipse/jre/bin/java (Oct 13, 2020, 6:44:51 PM)
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (One Path with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 800
Average time by Size: 0.33 Average depth by Size: 5.107843137254902
Average time by Depth: 0.34 Average depth by Depth: 4.813725490196078
Average time by (One Path compression): 0.29 Average depth by (One Path compression): 4.078431372549019
Average time by (Two Path Compression): 0.3 Average depth by (Two Path Compression): 1.803921568627451

2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (One Path with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 1600
Average time by Size: 0.27 Average depth by Size: 5.431372549019608
Average time by Depth: 0.41 Average depth by Depth: 5.2254901960784315
Average time by (One Path compression): 0.25 Average depth by (One Path compression): 4.029411764705882
Average time by (Two Path Compression): 0.37 Average depth by (Two Path Compression): 1.8627450980392157
```

Vedant Jain
Nuid - 001305281

Console Output for Benchmarking :

```
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 800
Average time by Size: 0.33 Average depth by Size: 5.107843137254902
Average time by Depth: 0.34 Average depth by Depth: 4.813725490196078
Average time by (One Path compression): 0.29 Average depth by (One Path compression): 4.078431372549019
Average time by (Two Path Compression): 0.3 Average depth by (Two Path Compression): 1.803921568627451

2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 1600
Average time by Size: 0.27 Average depth by Size: 5.431372549019608
Average time by Depth: 0.41 Average depth by Depth: 5.2254901960784315
Average time by (One Path compression): 0.25 Average depth by (One Path compression): 4.029411764705882
Average time by (Two Path Compression): 0.37 Average depth by (Two Path Compression): 1.8627450980392157

2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 3200
Average time by Size: 0.79 Average depth by Size: 5.921568627450981
Average time by Depth: 0.65 Average depth by Depth: 5.509803921568627
Average time by (One Path compression): 0.71 Average depth by (One Path compression): 4.205882352941177
Average time by (Two Path Compression): 0.51 Average depth by (Two Path Compression): 1.8529411764705883

2020-10-13 18:44:52 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:53 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:53 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:53 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 6400
Average time by Size: 1.39 Average depth by Size: 6.235294117647059
Average time by Depth: 1.59 Average depth by Depth: 5.931372549019608
Average time by (One Path compression): 1.23 Average depth by (One Path compression): 4.205882352941177
Average time by (Two Path Compression): 1.05 Average depth by (Two Path Compression): 1.892156862745098

2020-10-13 18:44:53 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:53 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:54 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:54 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 12800
Average time by Size: 3.24 Average depth by Size: 6.715686274509804
Average time by Depth: 2.88 Average depth by Depth: 6.382352941176471
Average time by (One Path compression): 2.53 Average depth by (One Path compression): 4.2254901960784315
Average time by (Two Path Compression): 2.15 Average depth by (Two Path Compression): 1.8823529411764706

2020-10-13 18:44:54 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:55 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
2020-10-13 18:44:56 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:44:56 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 25600
Average time by Size: 7.56 Average depth by Size: 7.0588235294117645
Average time by Depth: 7.01 Average depth by Depth: 6.666666666666667
Average time by (One Path compression): 5.62 Average depth by (One Path compression): 4.372549019607843
Average time by (Two Path Compression): 4.72 Average depth by (Two Path Compression): 1.803921568627451

2020-10-13 18:44:57 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Size(without path compression) with 100 runs
2020-10-13 18:44:59 INFO  Benchmark_Timer - Begin run: Weighted Quick Union by Depth(without path compression) with 100 runs
```

Vedant Jain
Nuid - 001305281

```
2020-10-13 18:45:00 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression（One Path with 100 runs
2020-10-13 18:45:02 INFO  Benchmark_Timer - Begin run: Weighted Quick Union with Path Compression (Two Path) with 100 runs
n = 51200
Average time by Size: 15.01 Average depth by Size: 7.519607843137255
Average time by Depth: 15.72 Average depth by Depth: 7.019607843137255
Average time by (One Path compression): 13.31 Average depth by (One Path compression): 4.2745098039215685
Average time by (Two Path Compression): 10.54 Average depth by (Two Path Compression): 1.892156862745098
```

I have summarized the benchmarking results in the table below:

Average Depth:

n	Weight quick union (store the size)	Weight quick union (store the depth)	WQU path compression (one path)	WQU path compression(two path)
800	5.10	4.81	4.07	1.80
1600	5.4	5.22	4.02	1.86
3200	5.92	5.50	4.20	1.85
6400	6.23	5.93	4.20	1.89
12800	6.71	6.38	4.22	1.88
25600	7.05	6.66	4.37	1.80
51200	7.51	7.01	4.27	1.89

Vedant Jain
Nuid - 001305281

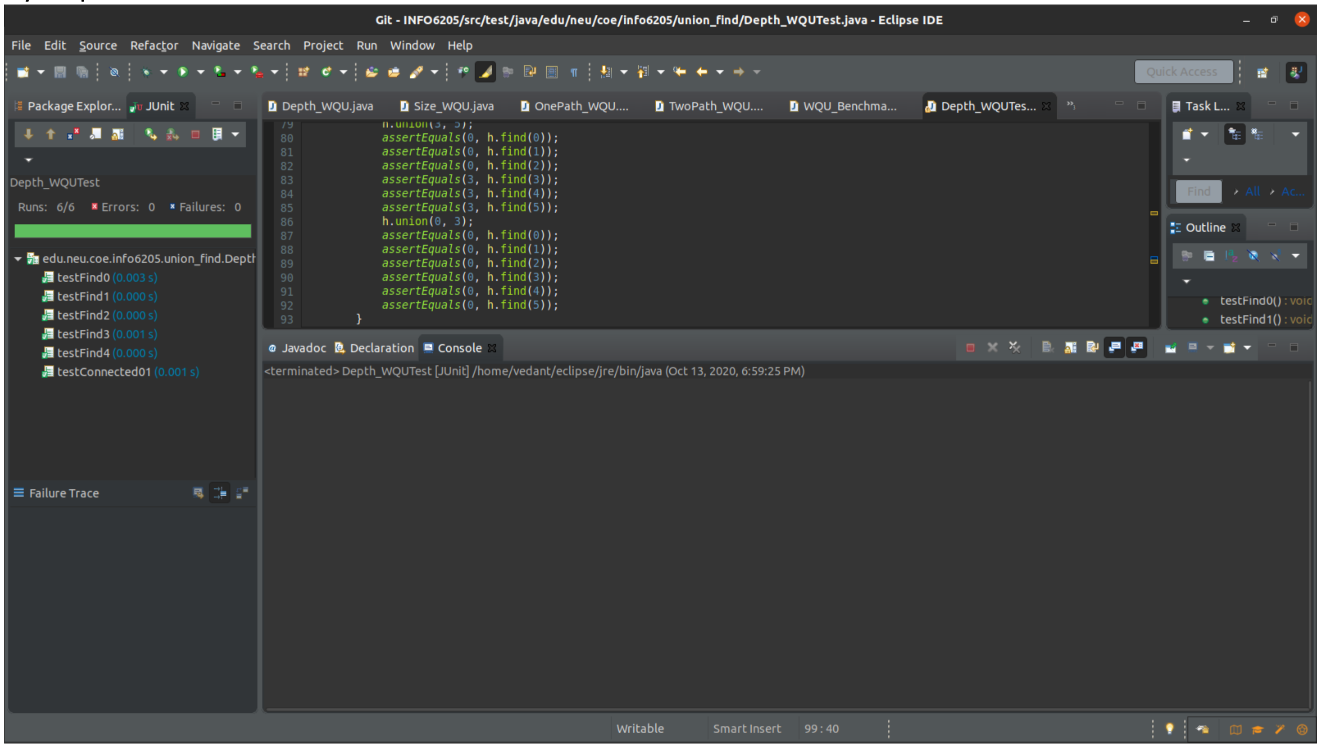
Average Running Time:

n	Weight quick union (store the size)	Weight quick union (store the depth)	WQU path compression (one path)	WQU path compression(two path)
800	0.33	0.34	0.29	0.30
1600	0.27	0.41	0.25	0.37
3200	0.79	0.65	0.71	0.51
6400	1.39	1.59	1.23	1.05
12800	3.24	2.88	2.53	2.15
25600	7.56	7.01	5.62	4.72
51200	15.01	15.72	13.31	10.54

```
Vedant Jain
Nuid - 001305281
```

Test Screenshots:

1) By Depth:-



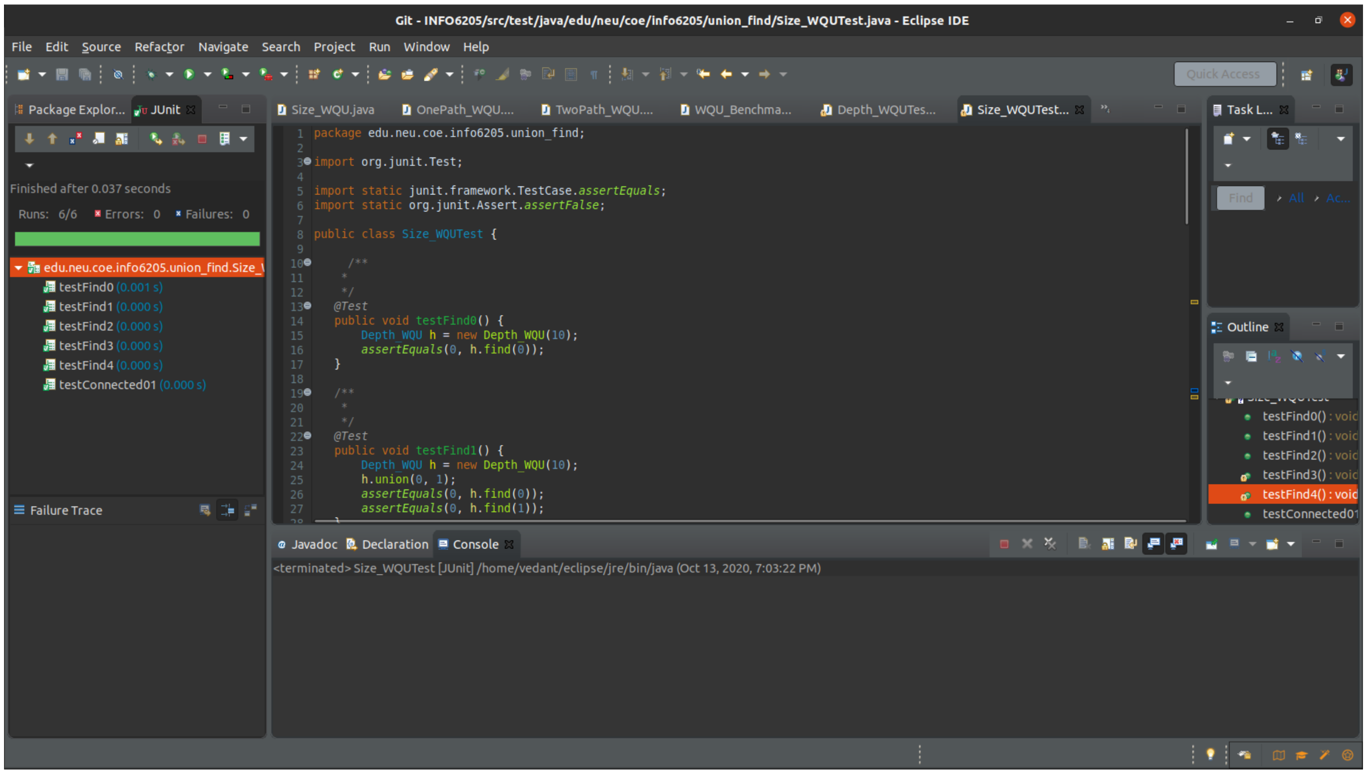
Git - INFO6205/src/test/java/edu/neu/coe/info6205/union_find/Depth_WQUTest.java - Eclipse IDE

```
Depth_WQUTest

Runs:  6/6    Errors:  0    Failures:  0

edu.neu.coe.info6205.union_find.Deptl
    testFind0 (0.003 s)
    testFind1 (0.000 s)
    testFind2 (0.000 s)
    testFind3 (0.001 s)
    testFind4 (0.000 s)
    testConnected01 (0.001 s)
```

```java
                h.union(3, 5);
                assertEquals(0, h.find(0));
                assertEquals(0, h.find(1));
                assertEquals(0, h.find(2));
                assertEquals(3, h.find(3));
                assertEquals(3, h.find(4));
                assertEquals(3, h.find(5));
                h.union(0, 3);
                assertEquals(0, h.find(0));
                assertEquals(0, h.find(1));
                assertEquals(0, h.find(2));
                assertEquals(0, h.find(3));
                assertEquals(0, h.find(4));
                assertEquals(0, h.find(5));
        }
```

```
<terminated> Depth_WQUTest [JUnit] /home/vedant/eclipse/jre/bin/java (Oct 13, 2020, 6:59:25 PM)
```

2) By Size:



Git - INFO6205/src/test/java/edu/neu/coe/info6205/union_find/Size_WQUTest.java - Eclipse IDE

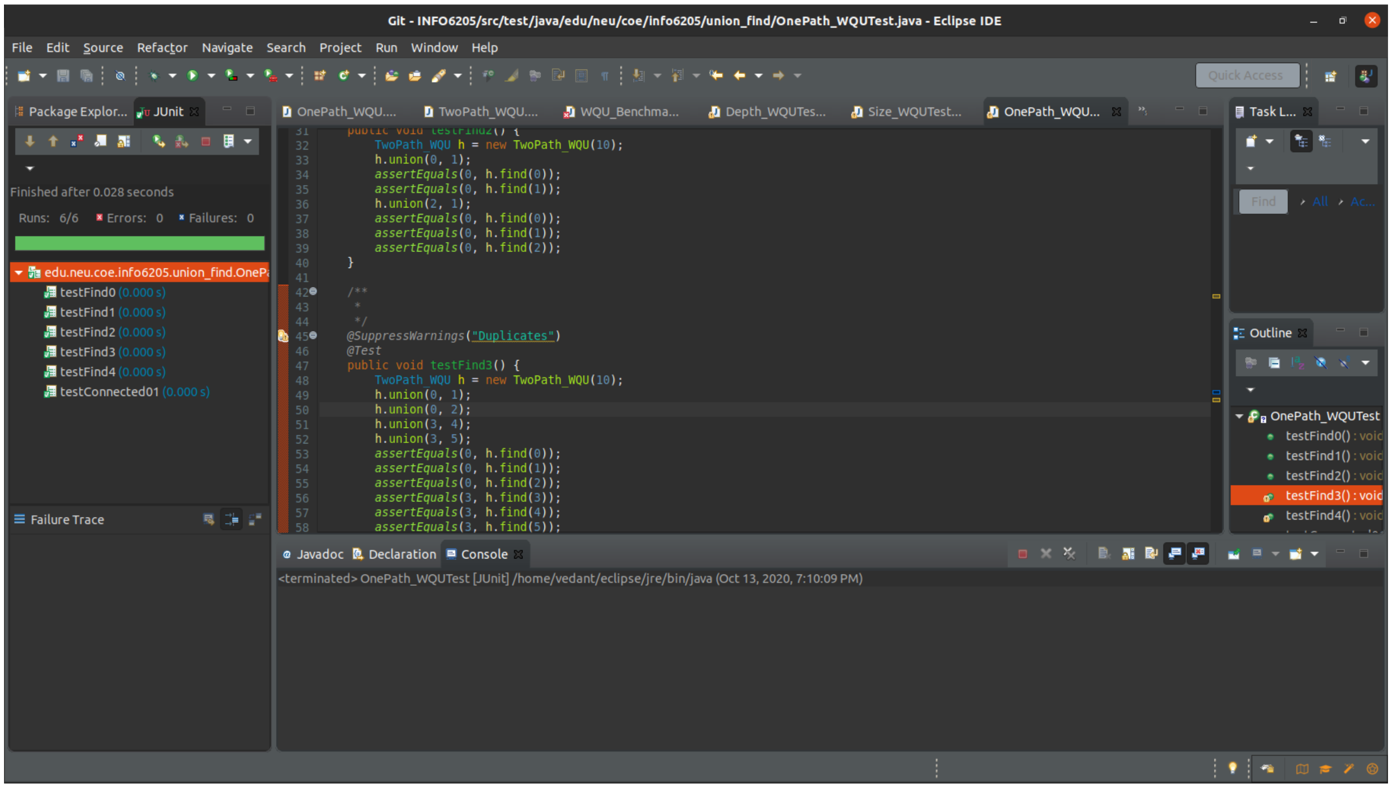
```
Finished after 0.037 seconds

Runs:  6/6    Errors:  0    Failures:  0

edu.neu.coe.info6205.union_find.Size_\
    testFind0 (0.001 s)
    testFind1 (0.000 s)
    testFind2 (0.000 s)
    testFind3 (0.000 s)
    testFind4 (0.000 s)
    testConnected01 (0.000 s)
```

```java
package edu.neu.coe.info6205.union_find;

import org.junit.Test;

import static junit.framework.TestCase.assertEquals;
import static org.junit.Assert.assertFalse;

public class Size_WQUTest {

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

    /**
     *
     */
    @Test
    public void testFind1() {
        Depth_WQU h = new Depth_WQU(10);
        h.union(0, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
```

```
<terminated> Size_WQUTest [JUnit] /home/vedant/eclipse/jre/bin/java (Oct 13, 2020, 7:03:22 PM)
```

Vedant Jain
Nuid - 001305281

3) OnePath:-



Git - INFO6205/src/test/java/edu/neu/coe/info6205/union_find/OnePath_WQUTest.java - Eclipse IDE

Finished after 0.028 seconds

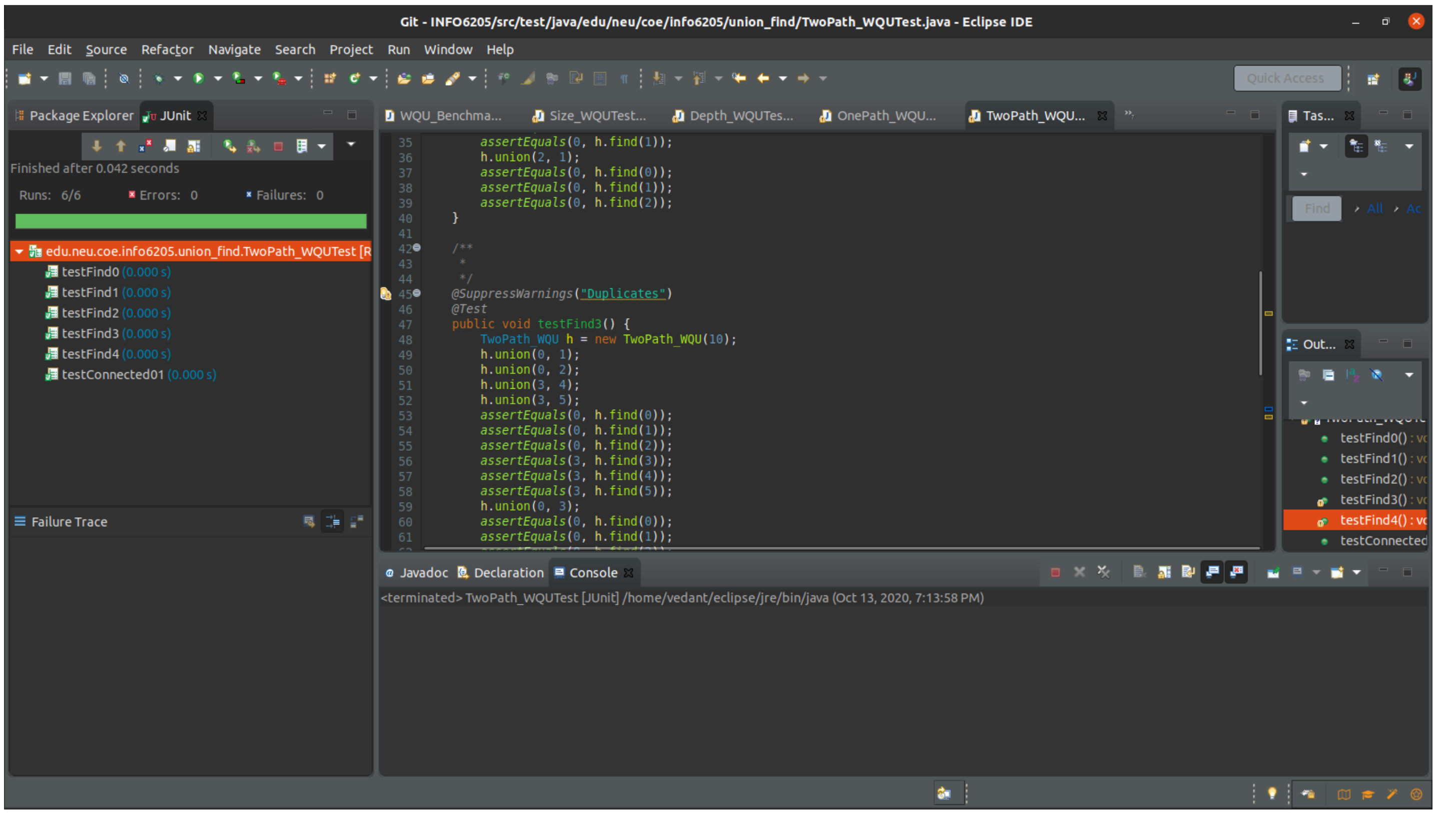
Runs: 6/6 Errors: 0 Failures: 0

edu.neu.coe.info6205.union_find.OneP
- testFind0 (0.000 s)
- testFind1 (0.000 s)
- testFind2 (0.000 s)
- testFind3 (0.000 s)
- testFind4 (0.000 s)
- testConnected01 (0.000 s)

```
        TwoPath_WQU h = new TwoPath_WQU(10);
        h.union(0, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        h.union(2, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
    }

    /**
     *
     */
    @SuppressWarnings("Duplicates")
    @Test
    public void testFind3() {
        TwoPath_WQU h = new TwoPath_WQU(10);
        h.union(0, 1);
        h.union(0, 2);
        h.union(3, 4);
        h.union(3, 5);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(3, h.find(3));
        assertEquals(3, h.find(4));
        assertEquals(3, h.find(5));
```

```
<terminated> OnePath_WQUTest [JUnit] /home/vedant/eclipse/jre/bin/java (Oct 13, 2020, 7:10:09 PM)
```

4) TwoPath:-



Git - INFO6205/src/test/java/edu/neu/coe/info6205/union_find/TwoPath_WQUTest.java - Eclipse IDE

Finished after 0.042 seconds

Runs: 6/6 Errors: 0 Failures: 0

edu.neu.coe.info6205.union_find.TwoPath_WQUTest [R
- testFind0 (0.000 s)
- testFind1 (0.000 s)
- testFind2 (0.000 s)
- testFind3 (0.000 s)
- testFind4 (0.000 s)
- testConnected01 (0.000 s)

```
        assertEquals(0, h.find(1));
        h.union(2, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
    }

    /**
     *
     */
    @SuppressWarnings("Duplicates")
    @Test
    public void testFind3() {
        TwoPath_WQU h = new TwoPath_WQU(10);
        h.union(0, 1);
        h.union(0, 2);
        h.union(3, 4);
        h.union(3, 5);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(3, h.find(3));
        assertEquals(3, h.find(4));
        assertEquals(3, h.find(5));
        h.union(0, 3);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
```

```
<terminated> TwoPath_WQUTest [JUnit] /home/vedant/eclipse/jre/bin/java (Oct 13, 2020, 7:13:58 PM)
```

Vedant Jain
Nuid - 001305281

- **Relationship Observation and Conclusion**

  For Weight Quick Union (Store the size) and Weight Quick Union (Store the depth), their average running times and average depth are almost the same. Since they have the same method of Find() and Union(). The only difference is the way they store the tree. The running time is O(N + N lg(N)).

  For Weight Quick Union with path compression, we observe that compared with Weight Quick Union(Without Path Compression), both One Pass and Two Path have a significant improvement. In the One Pass Compression, we set the parent of the node to its grandparent, and in the Two Path Compression, we set the parent of the node directly to its root. So, we can observe that the Two Path Compression performance is better than the One Path Compression. But the running time of them are O(N + N lg*(N)).

- **File Description:**

1) **Depth_WQU.java:** Weight Quick Union, store the depth.
2) **Size_WQU.java:** Weight Quick Union, store the size.
3) **OnePath_WQU.java:** Weight Quick Union with path compression, store the size
4) **TwoPath_WQU.java:** Weight Quick Union with path compression, store the size
5) **WQU_Benchmark.java:** run the 4 types of alternatives, calculate the running time and depth