

Program Structures and Algorithms
Fall 2022(SEC 06)

NAME: Mahathi Siddavatam
NUID: 002198134

Task:

Step 1:

(a) Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF_HWQUPC. All you have to do is to fill in the sections marked with `// TO BE IMPLEMENTED ... // ...END IMPLEMENTATION`.

(b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

Step 2:

Using your implementation of UF_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and $n-1$, calling `connected()` to determine if they are connected and `union()` if not. Loop until all sites are connected then print the number of connections generated. 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. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).

Step 3:

Determine the relationship between the number of objects (n) and the number of pairs (m) generated to accomplish this (i.e. to reduce the number of components from n to 1). Justify your conclusion in terms of your observations and what you think might be going on.

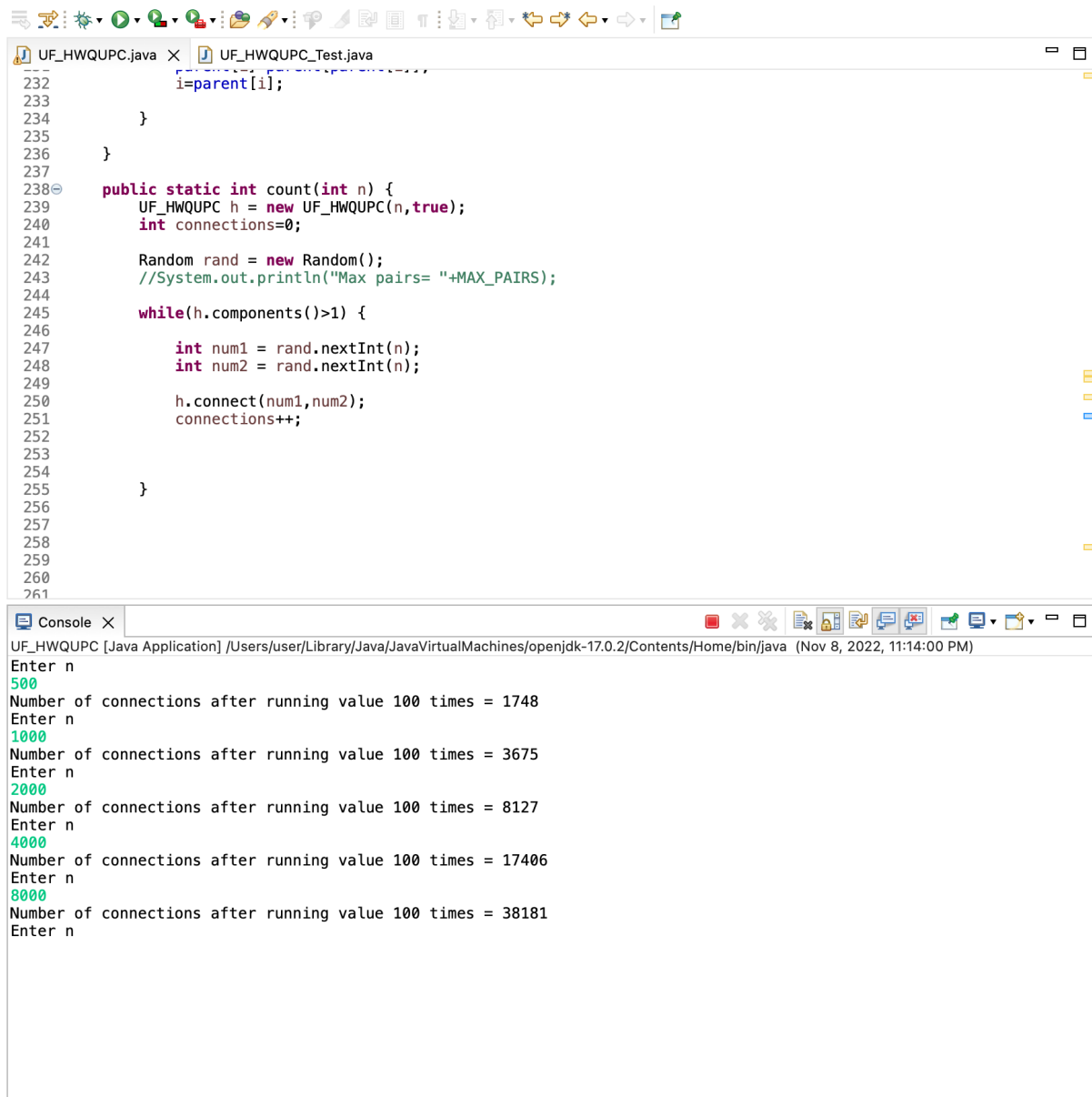
NOTE: although I'm not going to tell you in advance what the relationship is, I can assure you that it is a *simple* relationship.

Don't forget to follow the submission guidelines. And to use sufficient (and sufficiently large) different values of n .

Relationship Conclusion: The relationship between the number of objects (n) and number of pairs generated is approximately:

number of pairs generated = $n + n \log(\text{base } 10) n$. Many of these connections, while redundant, still count in the total.

Evidence to support that conclusion:



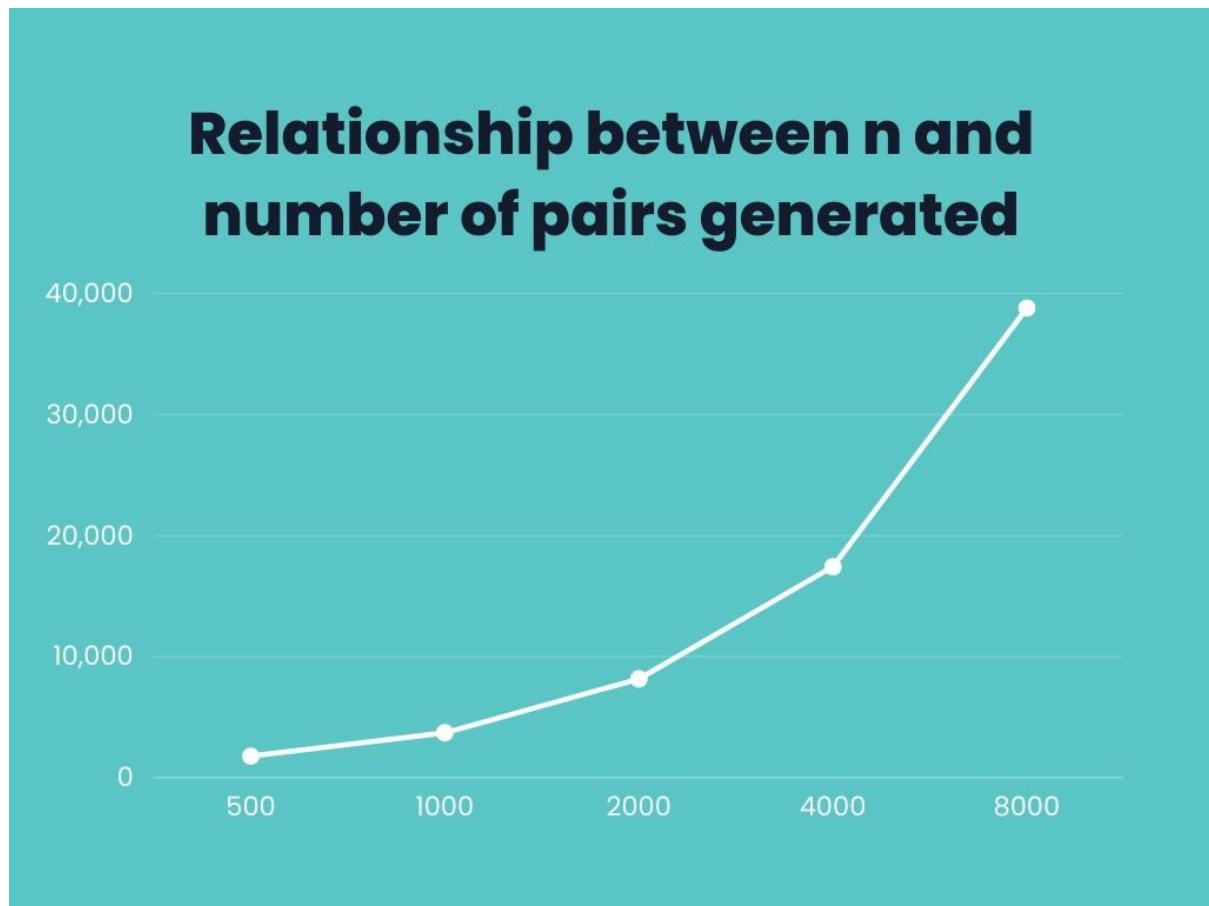
The screenshot shows an IDE with two tabs: `UF_HWQUPC.java` and `UF_HWQUPC_Test.java`. The `UF_HWQUPC_Test.java` tab is active, displaying the following Java code:

```
232     i=parent[i];
233
234     }
235
236     }
237
238     public static int count(int n) {
239         UF_HWQUPC h = new UF_HWQUPC(n,true);
240         int connections=0;
241
242         Random rand = new Random();
243         //System.out.println("Max pairs= "+MAX_PAIRS);
244
245         while(h.components()>1) {
246
247             int num1 = rand.nextInt(n);
248             int num2 = rand.nextInt(n);
249
250             h.connect(num1,num2);
251             connections++;
252
253
254
255         }
256
257
258
259
260
261
```

The console output shows the results of running the program for different values of `n`:

```
UF_HWQUPC [Java Application] /Users/user/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java (Nov 8, 2022, 11:14:00 PM)
Enter n
500
Number of connections after running value 100 times = 1748
Enter n
1000
Number of connections after running value 100 times = 3675
Enter n
2000
Number of connections after running value 100 times = 8127
Enter n
4000
Number of connections after running value 100 times = 17406
Enter n
8000
Number of connections after running value 100 times = 38181
Enter n
```

Graphical Representation:



Unit Test Screenshots:

The screenshot shows a unit test runner interface. On the left, a sidebar displays test results: "Runs: 13/13", "Errors: 0", and "Failures: 0". The main area shows the source code of the test class, `UF_HWQUPC_Test.java`, which includes several test methods. The code is as follows:

```
2 * Copyright (c) 2017. Phasmid Software
4
5 package edu.neu.coe.info6205.union_find;
6
7 import edu.neu.coe.info6205.util.PrivateMethodTester;
8
9 public class UF_HWQUPC_Test {
10
11     @Test
12     public void testToString() {
13         Connections h = new UF_HWQUPC(2);
14         assertEquals("UF_HWQUPC:\n" +
15             "  count: 2\n" +
16             "  path compression? true\n" +
17             "  parents: [0, 1]\n" +
18             "  heights: [1, 1]", h.toString());
19     }
20
21     /**
22     */
23
24     @Test
25     public void testIsConnected01() {
26         Connections h = new UF_HWQUPC(2);
27         assertFalse(h.isConnected(0, 1));
28     }
29
30     /**
31     */
32
33     @Test(expected = IllegalArgumentException.class)
34     public void testIsConnected02() {
35         Connections h = new UF_HWQUPC(1);
36         assertTrue(h.isConnected(0, 1));
37     }
38
39     /**
40     */
41
42     @Test
43     public void testIsConnected03() {
44         Connections h = new UF_HWQUPC(2);
45         final PrivateMethodTester tester = new PrivateMethodTester(h);
46         assertNull(tester.invokePrivate("updateParent", 0, 1));
47     }
48 }
```

