

Program Structures & Algorithms

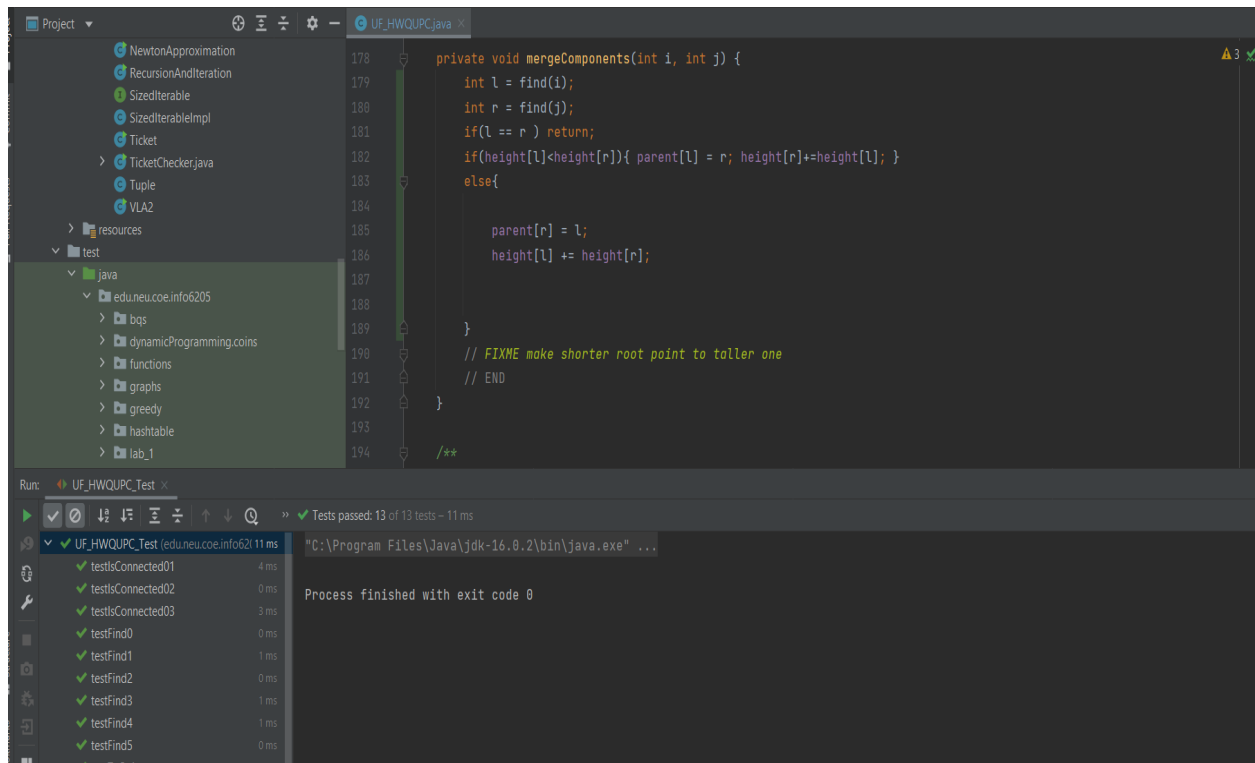
Assignment No. 3

Name: Varun Vuppala
(NUID): 002193132

Tasks:

- 1) (a) Implement height-weighted Quick Union with Path Compression.
(b) Check that the unit tests for this class all work.
- 2) Create a main program that takes n from the command line, calls `count()` and prints the returned value.
- 3) Determine the relationship between the number of objects (n) and the number of pairs (m)

Part I : All Test cases Passed

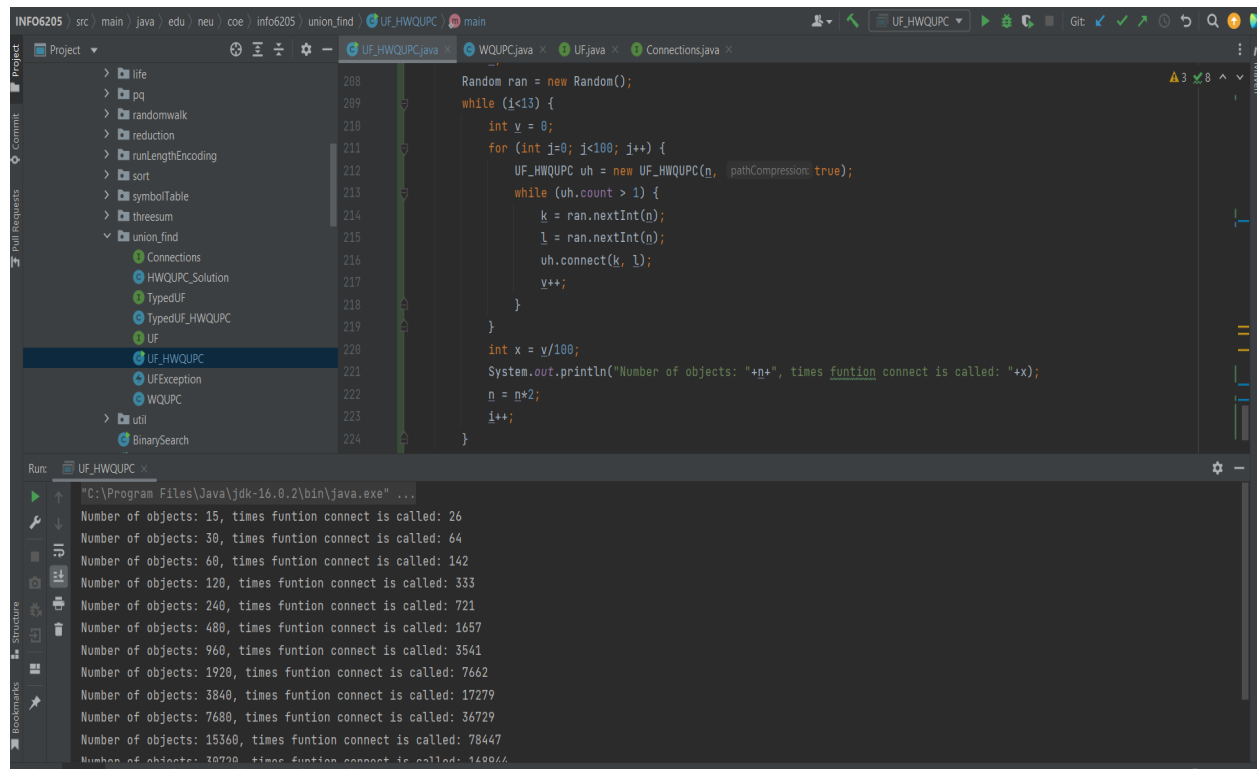


The screenshot displays an IDE with the following components:

- Project Explorer:** Shows a project structure with folders like `resources`, `test`, and `java`. The `java` folder contains `edu.neu.coe.info6205`, which includes subfolders like `bqs`, `dynamicProgramming.coins`, `functions`, `graphs`, `greedy`, `hashtable`, and `lab_1`.
- Code Editor:** Displays the `UF_HWQUPC.java` file. The code includes a `mergeComponents` method that uses `find` to locate roots and updates heights. Comments indicate a fix for root height and the end of the method. Line numbers 178 to 194 are visible.
- Run Console:** Shows the execution of `UF_HWQUPC_Test`. It reports "Tests passed: 13 of 13 tests - 11 ms" and "Process finished with exit code 0". A list of tests is shown with their durations:

Test Name	Duration
testIsConnected01	4 ms
testIsConnected02	0 ms
testIsConnected03	3 ms
testFind0	0 ms
testFind1	1 ms
testFind2	0 ms
testFind3	1 ms
testFind4	1 ms
testFind5	0 ms
testToString	0 ms

Output Screen Shot:



The screenshot shows an IDE with a project named 'info6205'. The left sidebar displays a file tree with a folder 'union_find' containing several files, including 'UF_HWQUPC'. The main editor window shows the code for 'UF_HWQUPC.java'. The code is a Java program that generates a random graph and uses a Union-Find data structure to track the number of objects and the number of times the 'funtion connect' method is called. The output window at the bottom shows the results of the program's execution for various values of 'n'.

```
Random ran = new Random();
while (i<13) {
    int v = 0;
    for (int j=0; j<100; j++) {
        UF_HWQUPC uh = new UF_HWQUPC(n, pathCompression: true);
        while (uh.count > 1) {
            k = ran.nextInt(n);
            l = ran.nextInt(n);
            uh.connect(k, l);
            v++;
        }
    }
    int x = v/100;
    System.out.println("Number of objects: "+n+", times funtion connect is called: "+x);
    n = n*2;
    i++;
}
```

Run: UF_HWQUPC x

```
C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...
Number of objects: 15, times funtion connect is called: 26
Number of objects: 30, times funtion connect is called: 64
Number of objects: 60, times funtion connect is called: 142
Number of objects: 120, times funtion connect is called: 333
Number of objects: 240, times funtion connect is called: 721
Number of objects: 480, times funtion connect is called: 1657
Number of objects: 960, times funtion connect is called: 3541
Number of objects: 1920, times funtion connect is called: 7662
Number of objects: 3840, times funtion connect is called: 17279
Number of objects: 7680, times funtion connect is called: 36729
Number of objects: 15360, times funtion connect is called: 78447
Number of objects: 30720, times funtion connect is called: 169046
```

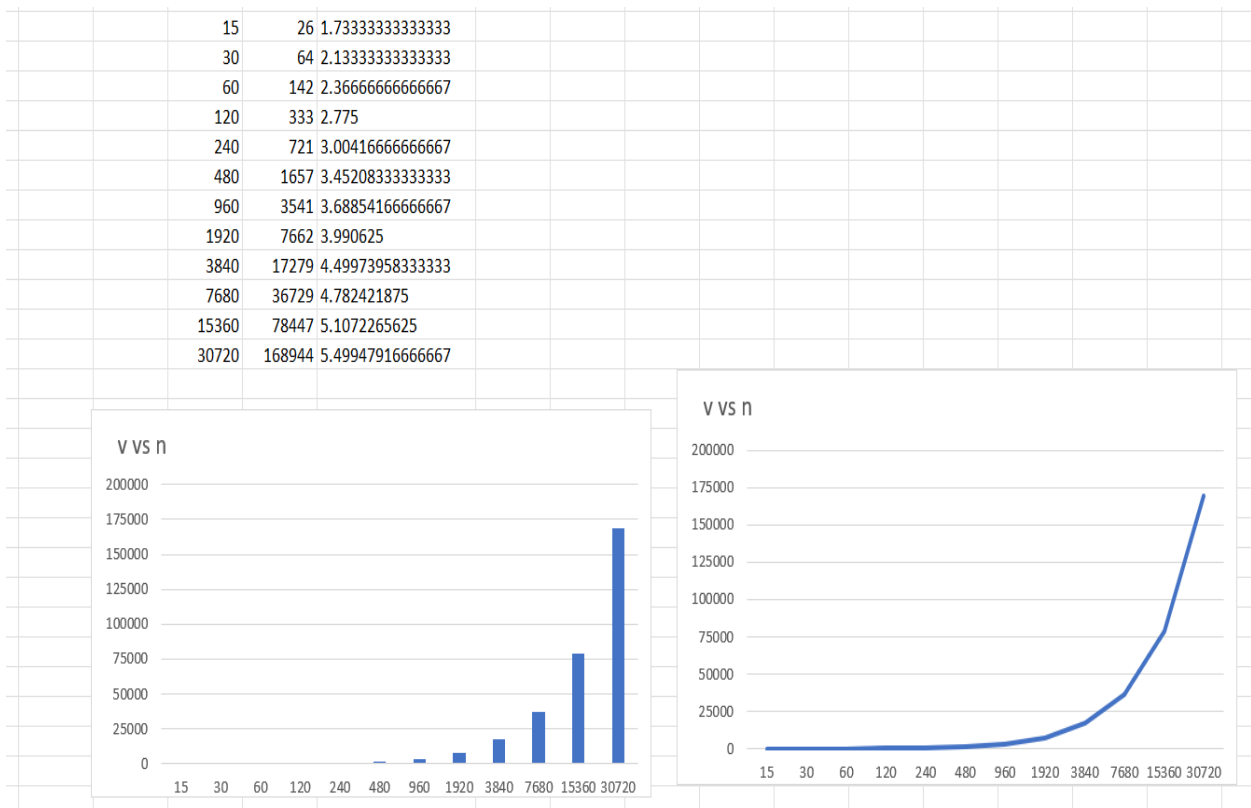
CONCLUSION:

As the number of objects(n) are increasing the number of pairs(v) changes linearithmic with n. To be more precise m is increasing n times the log of n to the base 5 which can be approximated to below expression.

$$m = n \times \log (n)$$

For larger values of n which is around 5000 to 20000, m value is nearly 5 times of n which can be seen in the evidence graph.

CONCLUSION GRAPH:



m	n	m/n
15	26	1.73333333333333
30	64	2.13333333333333
60	142	2.36666666666667
120	333	2.775
240	721	3.00416666666667
480	1657	3.45208333333333
960	3541	3.68854166666667
1920	7662	3.990625
3840	17279	4.49973958333333
7680	36729	4.782421875
15360	78447	5.1072265625
30720	168944	5.49947916666667