

# SAI KUMAR GANGA (002191288)

## INFO 6205 PROGRAM STRUCTURES AND ALGORITHMS

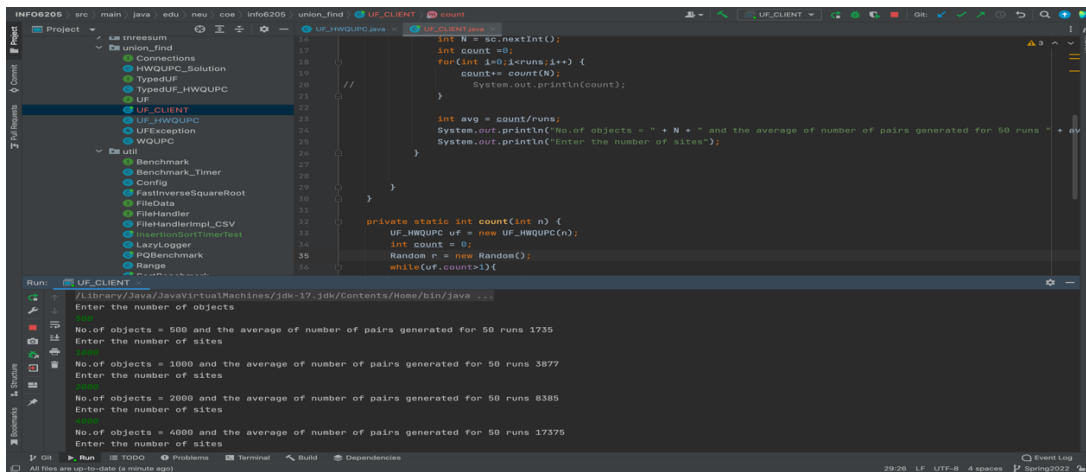
### Assignment-3(WQUPC)

#### 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 fill in the sections marked with // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION.

#### Solution:

The solution can be accessed through the UF\_HWQUPC.java.



The screenshot shows an IDE with a project named 'INFO6205'. The file explorer on the left shows a package 'union\_find' containing several files, including 'UF\_HWQUPC.java'. The main editor displays the code for 'UF\_CLIENT.java', which includes a 'count' method that runs 50 trials for different numbers of objects and sites. The output window at the bottom shows the results of these trials.

```
14  int N = SC.nextInt();
15
16  int count = 0;
17  for (int i=0; i<runs; i++) {
18      count += count(N);
19  }
20  System.out.println(count);
21
22  // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION
23
24  int avg = count/runs;
25  System.out.println("No. of objects = " + N + " and the average of number of pairs generated for 50 runs " + avg);
26  System.out.println("Enter the number of sites");
27
28  }
29
30  }
31
32  private static int count(int n) {
33      UF_HWQUPC uf = new UF_HWQUPC(n);
34      int count = 0;
35      Random r = new Random();
36      while (uf.count > 0) {
37          count++;
38          uf.union(r.nextInt(n), r.nextInt(n));
39      }
40  }
```

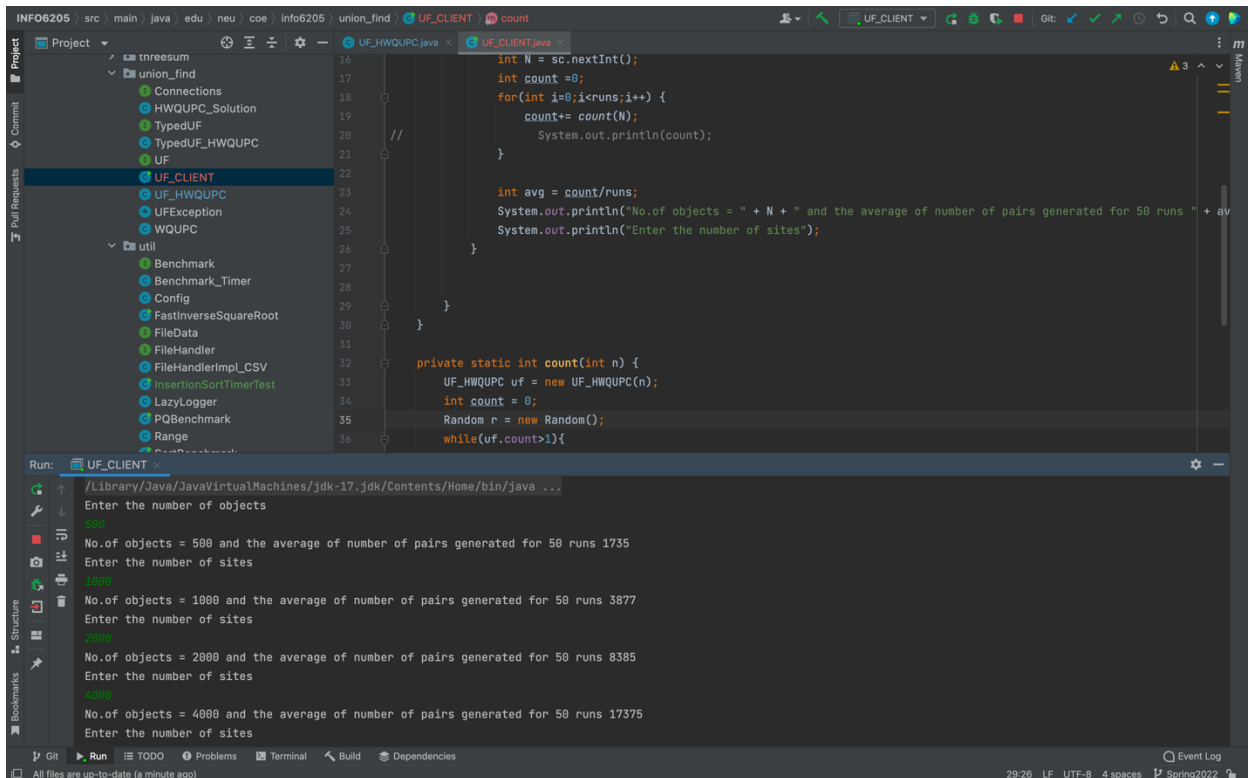
Run: UF\_CLIENT

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
Enter the number of objects
No. of objects = 500 and the average of number of pairs generated for 50 runs 1735
Enter the number of sites
No. of objects = 1000 and the average of number of pairs generated for 50 runs 3877
Enter the number of sites
No. of objects = 2000 and the average of number of pairs generated for 50 runs 8385
Enter the number of sites
No. of objects = 4000 and the average of number of pairs generated for 50 runs 17375
Enter the number of sites
```

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

### Proof:

UF\_HWQUPC\_Test.java is the code for all the unit test cases, and the following screenshot is proof of the test cases being passed



The screenshot shows an IDE with the following components:

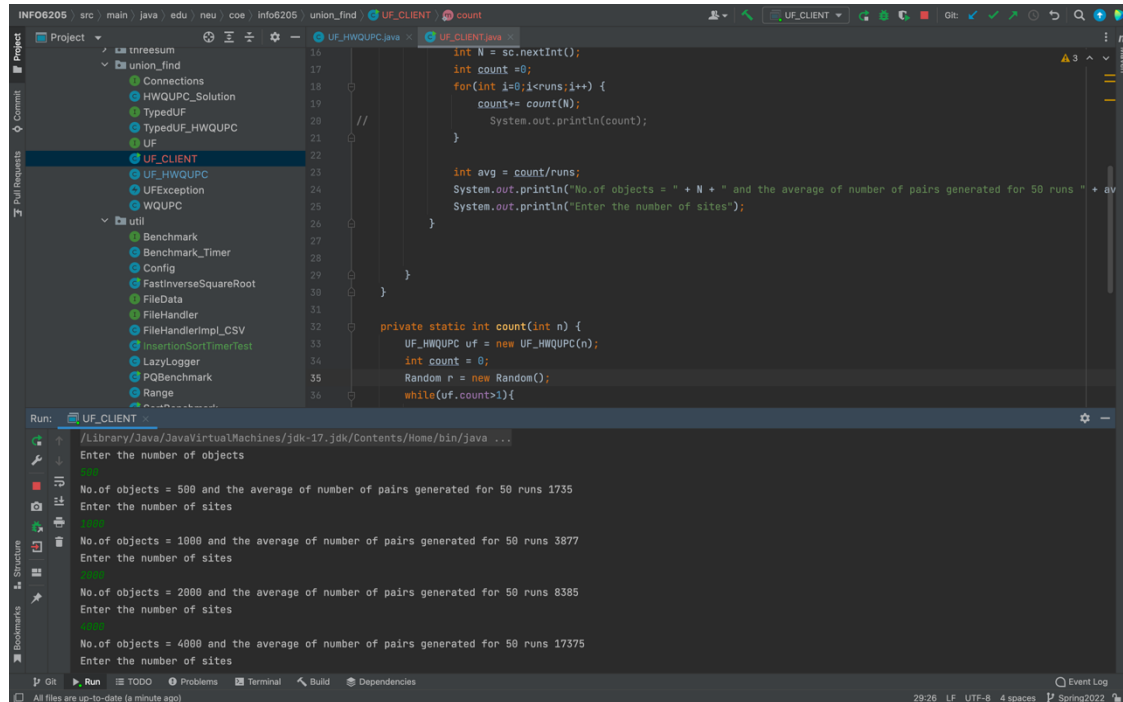
- Project Explorer:** Shows a project named 'INFO6205' with a package 'union\_find' containing files like 'Connections', 'HWQUPC\_Solution', 'TypedUF', 'TypedUF\_HWQUPC', 'UF', 'UF\_CLIENT', 'UF\_HWQUPC', 'UFException', 'WQUPC', and a 'util' package with various utility classes.
- Editor:** Displays the code for 'UF\_CLIENT.java'. The code includes a main method that takes an integer 'N' from the command line, generates random pairs of integers, and prints the number of objects and the average number of pairs generated for 50 runs. It also includes a static method 'count' that takes an integer 'n' and returns the number of connections generated.
- Run Console:** Shows the output of the program. It prompts the user to 'Enter the number of objects' and 'Enter the number of sites'. The output shows the results for 500, 1000, 2000, and 4000 objects, with the average number of pairs generated for 50 runs being 1735, 3977, 8385, and 17375 respectively.

### 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).

## Solution:

UF\_CLIENT.java. is the code and following screenshot is the evidence that the given input and the code is executing properly and with the outputs.



The screenshot shows an IDE with the following components:

- Project Explorer:** A tree view on the left showing a project named 'union\_find' with sub-packages like 'Connections', 'HWQUPC\_Solution', 'TypedUF', 'TypedUF\_HWQUPC', 'UF', 'UF\_CLIENT', 'UF\_HWQUPC', 'UFException', 'WQUPC', and 'util'. The 'UF\_CLIENT' package is selected.
- Code Editor:** The main area displays the code for 'UF\_CLIENT.java'. It includes a main method that takes an integer 'N' and a 'count' method that takes an integer 'n'. The main method generates 'N' objects and calls 'count' 50 times, printing the results. The 'count' method uses a 'UF\_HWQUPC' object to count the number of pairs generated for 'n' objects.
- Run Console:** The bottom panel shows the output of the program. It displays the results of 50 runs for different values of 'N' (500, 1000, 2000, 4000). For each run, it prints the number of objects, the average number of pairs generated, and the number of sites.

```
int N = sc.nextInt();
int count = 0;
for(int i=0; i<runs; i++) {
    count += count(N);
    System.out.println(count);
}

int avg = count/runs;
System.out.println("No. of objects = " + N + " and the average of number of pairs generated for 50 runs " + avg);
System.out.println("Enter the number of sites");

private static int count(int n) {
    UF_HWQUPC uf = new UF_HWQUPC(n);
    int count = 0;
    Random r = new Random();
    while(uf.count > 1) {
        int i = r.nextInt(n);
        int j = r.nextInt(n);
        uf.union(i, j);
    }
    return count;
}
```

Run: UF\_CLIENT

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Enter the number of objects

500

No. of objects = 500 and the average of number of pairs generated for 50 runs 1735

Enter the number of sites

1000

No. of objects = 1000 and the average of number of pairs generated for 50 runs 3877

Enter the number of sites

2000

No. of objects = 2000 and the average of number of pairs generated for 50 runs 8385

Enter the number of sites

4000

No. of objects = 4000 and the average of number of pairs generated for 50 runs 17375

Enter the number of sites

## 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.

## Conclusion and Relationship:

N values ranging from 500 to 256000(doubling every time), for each value of n the program runs 50 times and the avg value of m is noted down. The constant can be approximated, and relationship can be determined as

$$m = \frac{1}{2} * n * \log_2 n$$

Evidence & Graph:

