

Program Structures and Algorithms Spring 2024

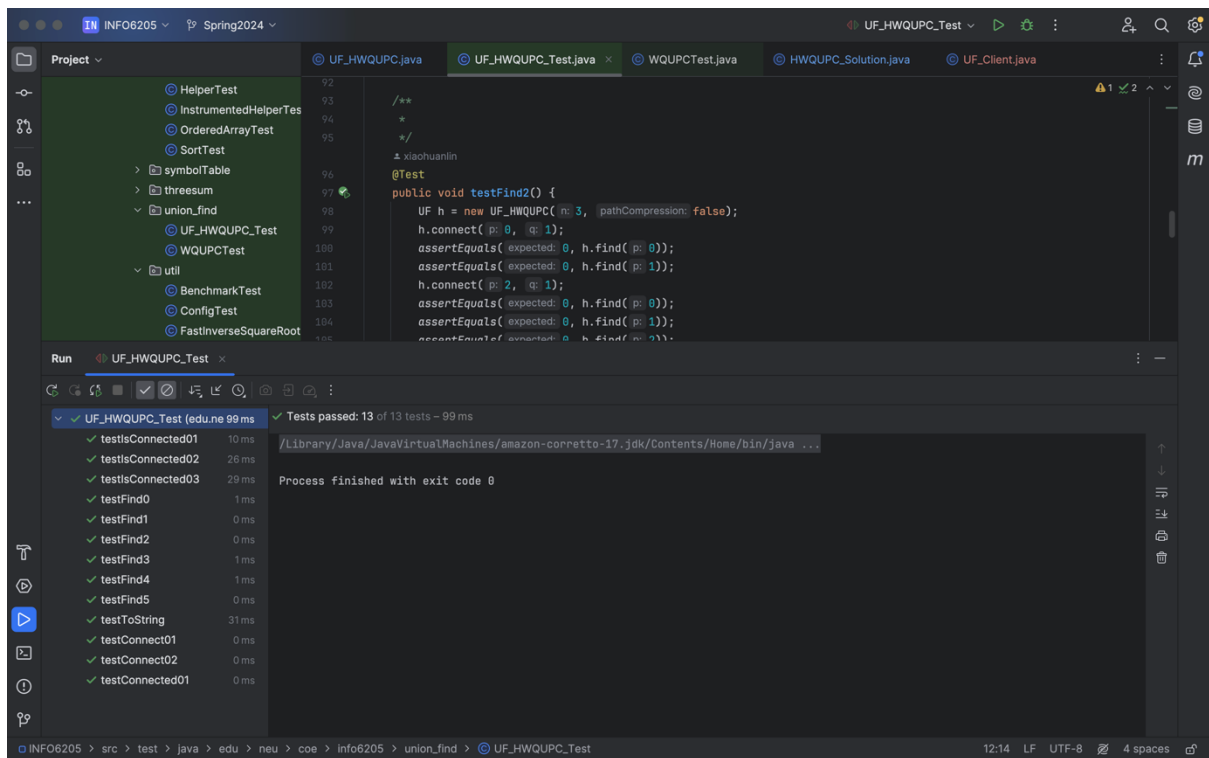
NAME: Saurabh Srivastava

NUID: 002895225

GITHUB LINK: <https://github.com/ssaurabh760/INFO6205>

Assignment 4:

Unit Test Screenshots



Screenshot for UF_Client.java

The screenshot shows an IDE with the following components:

- Project Explorer:** Displays a project structure with packages like `symbolTable`, `threesum`, `union_find`, and `util`. The `UF_Client.java` file is selected.
- Editor:** Shows the code for `UF_Client.java`. It includes a `for` loop that calls `countConnectionsAndPairs(n)` and prints the results. A `static class Result` is defined with `final int pairsGenerated` and `final int connections`.
- Run Console:** Shows the output of the program. It displays the results for `n` values from 100 to 6400 in increments of 200. The output is as follows:

n	pairs generated	connections required
100	246	99
200	657	199
400	1164	399
800	2251	799
1600	5659	1599
3200	11635	3199
6400	30011	6399

The console also shows the command used to run the program: `/Library/Java/JavaVirtualMachines/amazon-corretto-17.jdk/Contents/Home/bin/java ...` and the message "Process finished with exit code 0".

Observations:

For $n = 100$, pairs generated: 246, connections required: 99

For $n = 200$, pairs generated: 657, connections required: 199

For $n = 400$, pairs generated: 1164, connections required: 399

For $n = 800$, pairs generated: 2251, connections required: 799

For $n = 1600$, pairs generated: 5659, connections required: 1599

For $n = 3200$, pairs generated: 11635, connections required: 3199

For $n = 6400$, pairs generated: 30011, connections required: 6399

The observations depict an almost linear relationship between m and n generated to reduce the components from n to 1. The difference between the number of generated pairs and the actual connections ($n-1$) showcases the algorithm's efficiency in handling redundant checks without affecting the final outcome. The union-find algorithm, especially with path compression and height-weighting optimizations, efficiently manages to minimize the work done during each union operation.

