Program Structures and Algorithms

Spring 2023(SEC – 8)

NAME: Natarajan Lekshmi Narayana Pillai

NUID: 002766033

Task:

Step 1:A

(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.

Relationship Conclusion:

We can conclude that in order to reduce the number of objects from n to 1, which means that all the pairs are connected, the average value of m over 100 runs for each value of “n”, is equal to 0.368237.

Therefore the relationship between the number of objects (n) and the number of randomly generated pairs (m) can be expressed as follows:

$m = c * n * \log_2(n)$

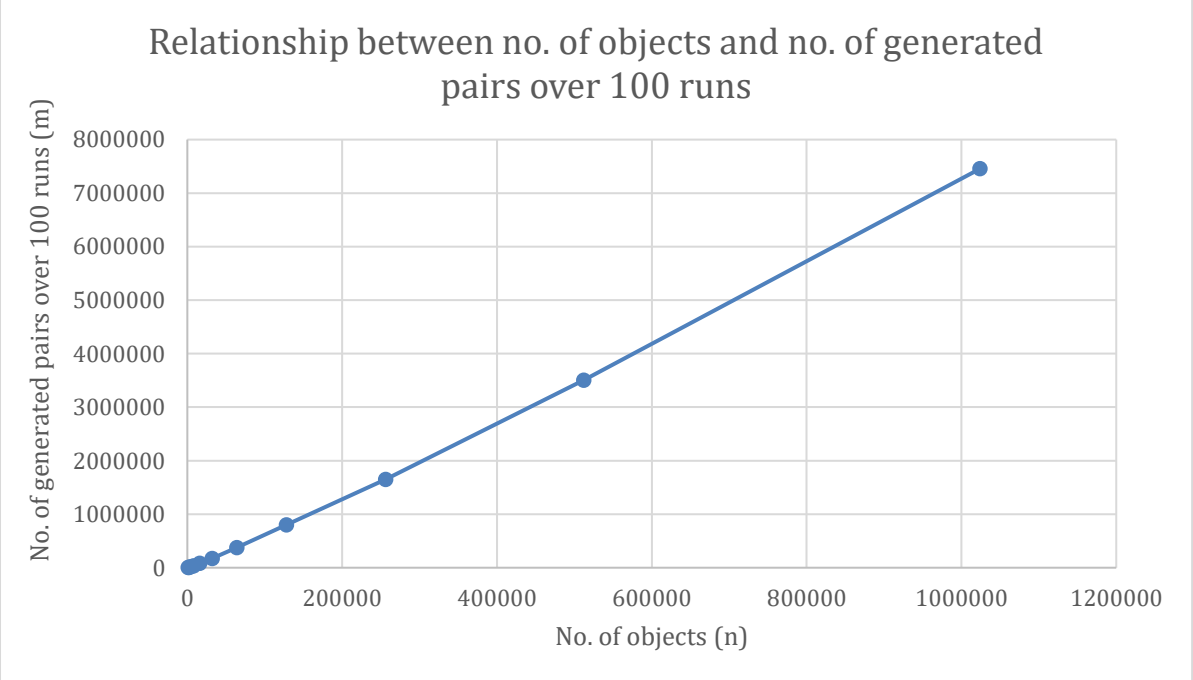
Where, $c = m / (n * \log_2(n))$

Therefore, $m \alpha n * \log_2(n)$

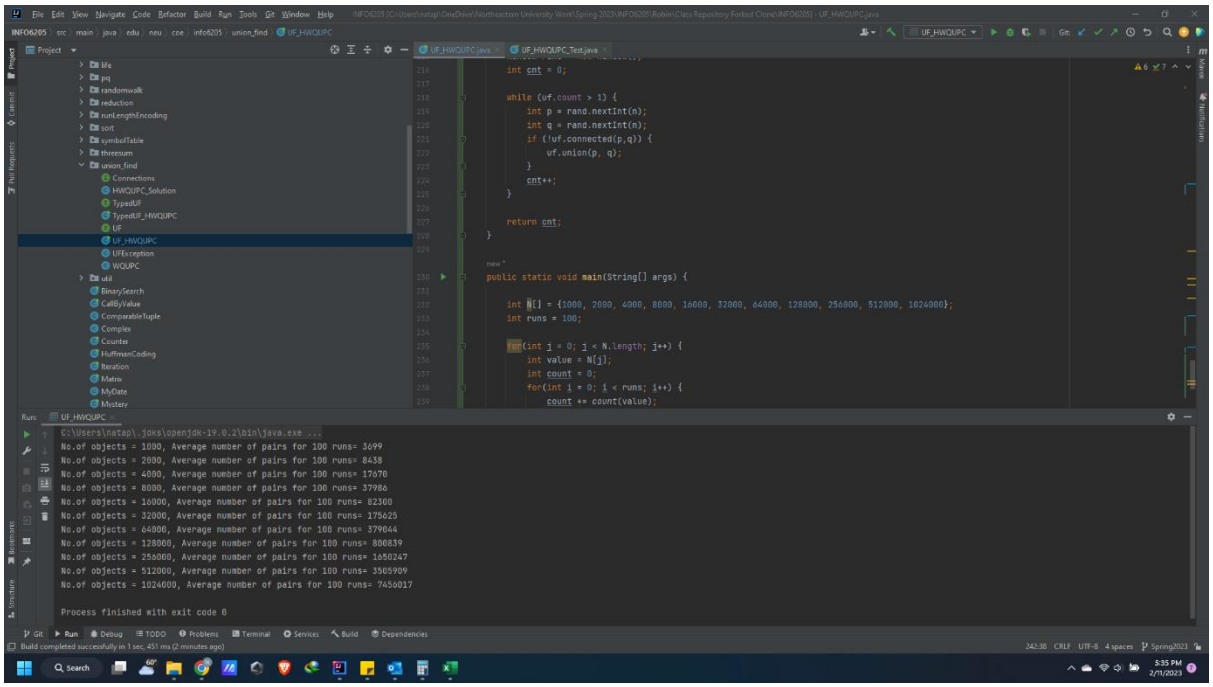
where c is a constant and can be roughly equated to 0.368237, establishing the relationship between “n” and “m”.

Evidence to support that conclusion:

n (No. of objects)	m (No. of generated pairs over 100 runs)	$n*\log_2(n)$	$m/(n*\log_2(n))$
1000	3699	9965.784285	0.371169985
2000	8438	21931.56857	0.384742203
4000	17670	47863.13714	0.36917764
8000	37986	103726.2743	0.366213867
16000	82300	223452.5486	0.368310859
32000	175625	478905.0971	0.366721927
64000	379044	1021810.194	0.370953434
128000	800839	2171620.388	0.368774858
256000	1650247	4599240.777	0.358808569
512000	3505909	9710481.554	0.361043784
1024000	7456017	20444963.11	0.364687232



Output Screenshot:



Unit Test Screenshots: