Program Structures & Algorithms

Spring 2022

Assignment No. 03

Name : Akshay Kumthi Matad
(NUID) : 002928833

Task : UF_HWQUPC

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).

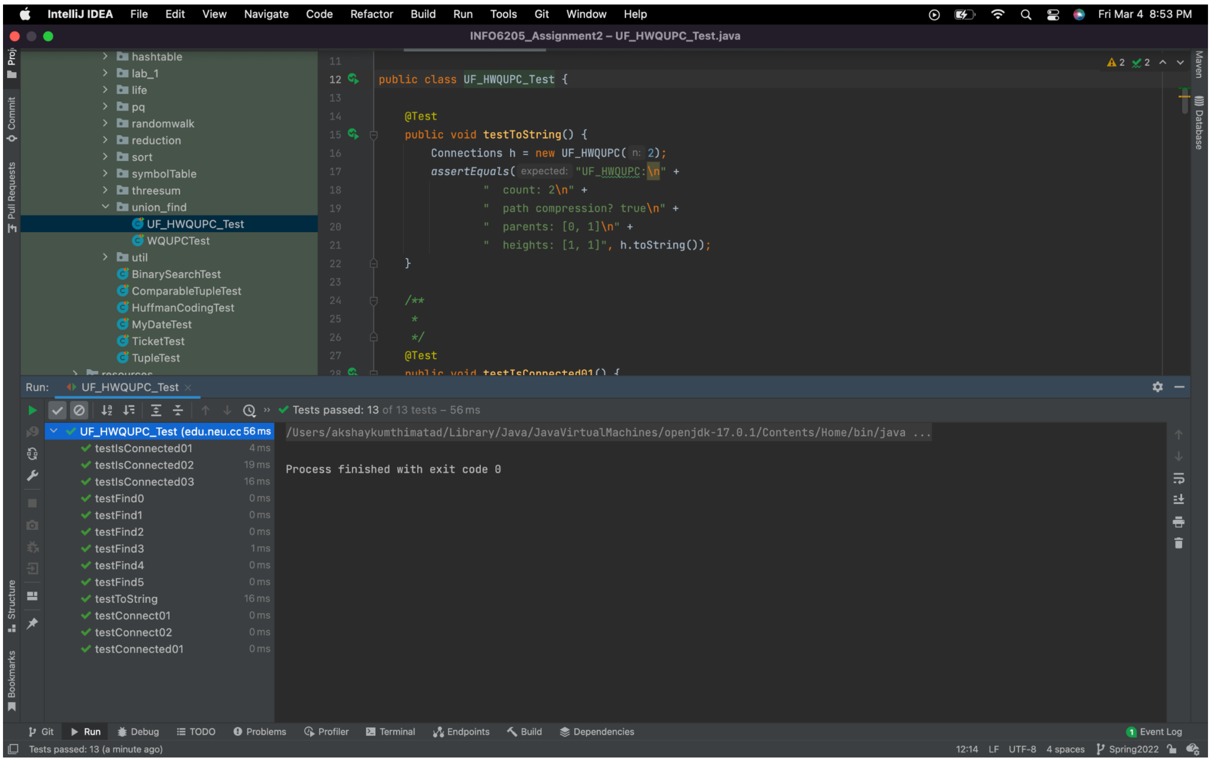
Step 3:

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.

Output screenshot :

Step 1 OUTPUT :

UF_HWQUPC_Test.java - All test Cases PASSED



IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Fri Mar 4 8:53 PM
hashtable
lab_1
life
pq
randomwalk
reduction
sort
symbolTable
threesum
union_find
UF_HWQUPC_Test
WQUPCTest
util
BinarySearchTest
ComparableTupleTest
HuffmanCodingTest
MyDateTest
TicketTest
TupleTest
Git Run Debug TODO Problems Profiler Terminal Endpoints Build Dependencies Event Log
12:14 LF UTF-8 4 spaces Spring2022

INFO6205_Assignment2 – UF_HWQUPC_Test.java

```java
public class UF_HWQUPC_Test {

    @Test
    public void testToString() {
        Connections h = new UF_HWQUPC( n: 2);
        assertEquals( expected: "UF_HWQUPC:\n" +
                "  count: 2\n" +
                "  path compression? true\n" +
                "  parents: [0, 1]\n" +
                "  heights: [1, 1]", h.toString());
    }

    /**
     *
     */
    @Test
    public void testIsConnected01() {
```

Run: UF_HWQUPC_Test

✓ Tests passed: 13 of 13 tests – 56 ms

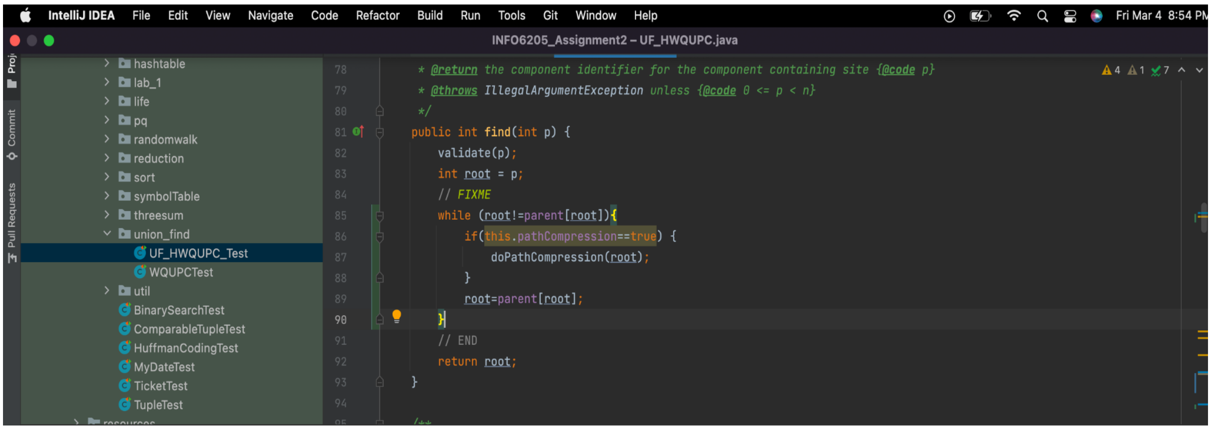
```
✓ UF_HWQUPC_Test (edu.neu.cc 56 ms
    ✓ testIsConnected01     4 ms
    ✓ testIsConnected02    19 ms
    ✓ testIsConnected03    16 ms
    ✓ testFind0             0 ms
    ✓ testFind1             0 ms
    ✓ testFind2             0 ms
    ✓ testFind3             1 ms
    ✓ testFind4             0 ms
    ✓ testFind5             0 ms
    ✓ testToString         16 ms
    ✓ testConnect01         0 ms
    ✓ testConnect02         0 ms
    ✓ testConnected01       0 ms
```

```
/Users/akshaykumthimatad/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Contents/Home/bin/java ...

Process finished with exit code 0
```

Tests passed: 13 (a minute ago)

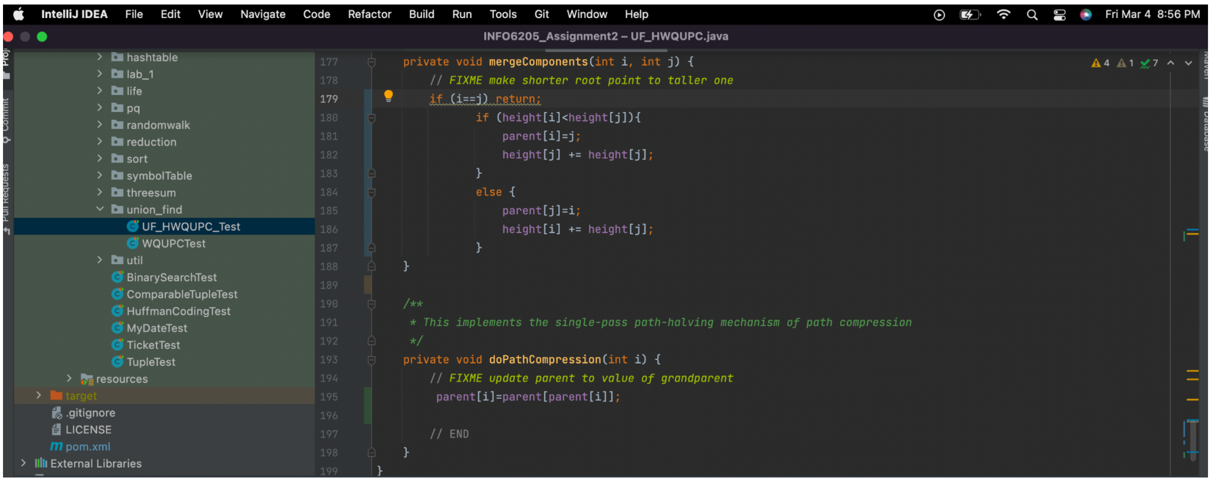
UF_HWQUPC code:



INFO6205_Assignment2 – UF_HWQUPC.java

```java
     * @return the component identifier for the component containing site {@code p}
     * @throws IllegalArgumentException unless {@code 0 <= p < n}
     */
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        while (root!=parent[root]){
            if(this.pathCompression==true) {
                doPathCompression(root);
            }
            root=parent[root];
        }
        // END
        return root;
    }

    /**
```



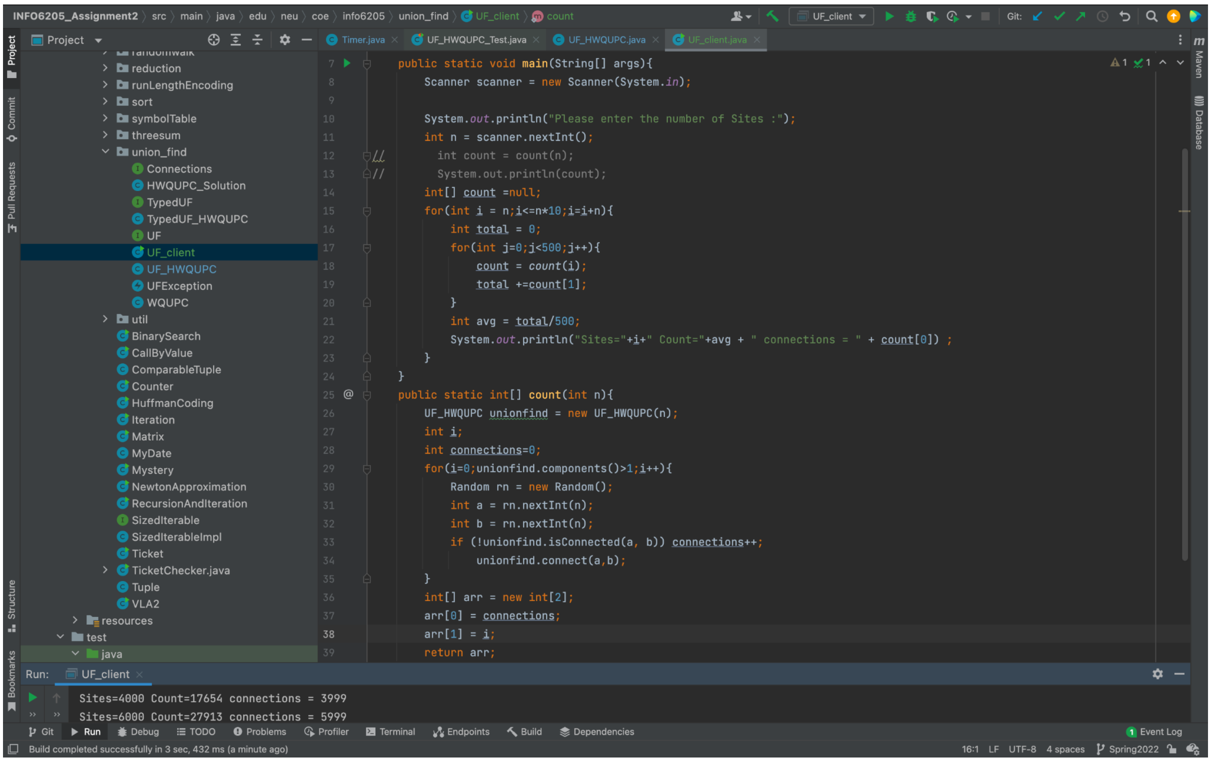
```java
    private void mergeComponents(int i, int j) {
        // FIXME make shorter root point to taller one
        if (i==j) return;
            if (height[i]<height[j]){
                parent[i]=j;
                height[j] += height[j];
            }
            else {
                parent[j]=i;
                height[i] += height[j];
            }
    }

    /**
     * This implements the single-pass path-halving mechanism of path compression
     */
    private void doPathCompression(int i) {
        // FIXME update parent to value of grandparent
         parent[i]=parent[parent[i]];

        // END
    }
}
```

STEP 2 OUTPUT :

UF_client.java code :


INFO6205_Assignment2 〉 src 〉 main 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 union_find 〉 UF_client 〉 count
Timer.java
UF_HWQUPC_Test.java
UF_HWQUPC.java
UF_client.java
Run: UF_client

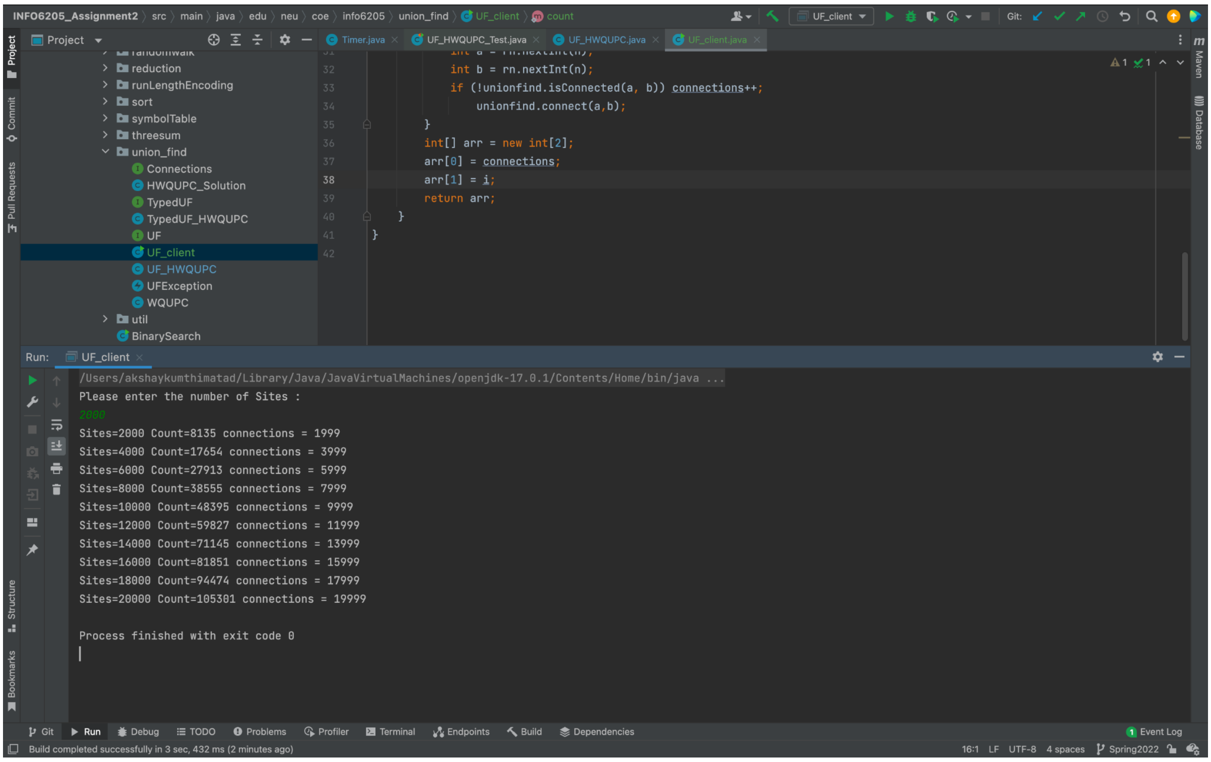
```java
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);

        System.out.println("Please enter the number of Sites :");
        int n = scanner.nextInt();
//        int count = count(n);
//        System.out.println(count);
        int[] count =null;
        for(int i = n;i<=n*10;i=i+n){
            int total = 0;
            for(int j=0;j<500;j++){
                count = count(i);
                total +=count[1];
            }
            int avg = total/500;
            System.out.println("Sites="+i+" Count="+avg + " connections = " + count[0]) ;
        }
    }
    public static int[] count(int n){
        UF_HWQUPC unionfind = new UF_HWQUPC(n);
        int i;
        int connections=0;
        for(i=0;unionfind.components()>1;i++){
            Random rn = new Random();
            int a = rn.nextInt(n);
            int b = rn.nextInt(n);
            if (!unionfind.isConnected(a, b)) connections++;
                unionfind.connect(a,b);
        }
        int[] arr = new int[2];
        arr[0] = connections;
        arr[1] = i;
        return arr;
```

```
Sites=4000 Count=17654 connections = 3999
Sites=6000 Count=27913 connections = 5999
```

Build completed successfully in 3 sec, 432 ms (a minute ago)

Output of client.java file :



```
                int a = rn.nextInt(n);
                int b = rn.nextInt(n);
                if (!unionfind.isConnected(a, b)) connections++;
                    unionfind.connect(a,b);
            }
            int[] arr = new int[2];
            arr[0] = connections;
            arr[1] = i;
            return arr;
        }
    }
```

```
/Users/akshaykumthimatad/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Contents/Home/bin/java ...
Please enter the number of Sites :
2000
Sites=2000 Count=8135 connections = 1999
Sites=4000 Count=17654 connections = 3999
Sites=6000 Count=27913 connections = 5999
Sites=8000 Count=38555 connections = 7999
Sites=10000 Count=48395 connections = 9999
Sites=12000 Count=59827 connections = 11999
Sites=14000 Count=71145 connections = 13999
Sites=16000 Count=81851 connections = 15999
Sites=18000 Count=94474 connections = 17999
Sites=20000 Count=105301 connections = 19999

Process finished with exit code 0
```

Build completed successfully in 3 sec, 432 ms (2 minutes ago)

Step 3

<u>Relationship Conclusion :</u>

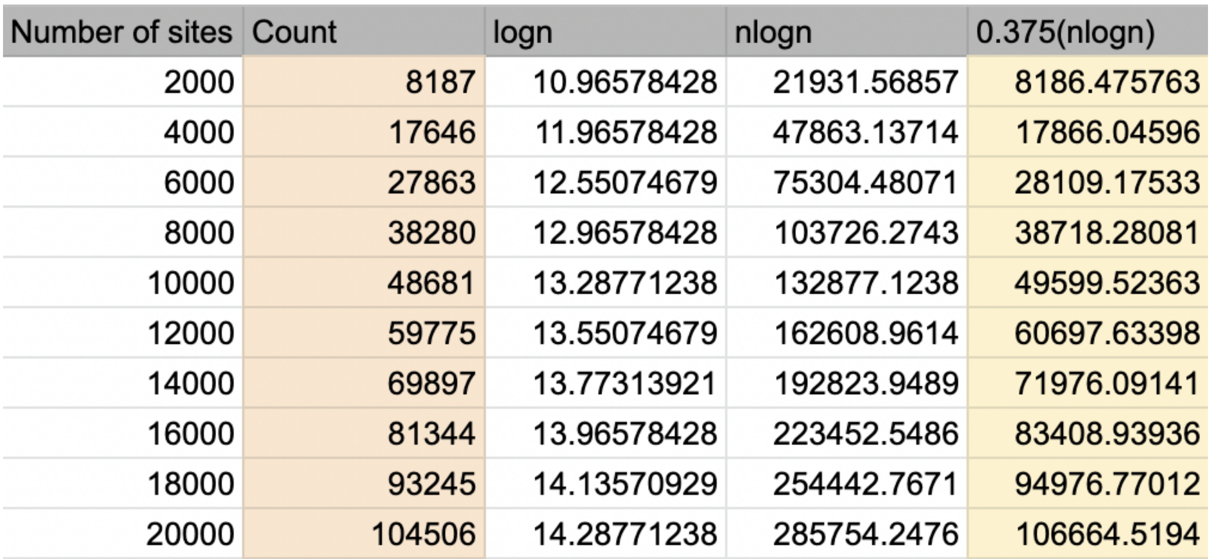
I believe the relationship between Number of sites and Number of pairs(Count/m) is linear.

N is directly proportional to M

But taking the theoretical time complexity into consideration the relationship can be defined as
M = c(Nlogn)
Where c is the constant which varies from machine to machine.
According to my output and graph,

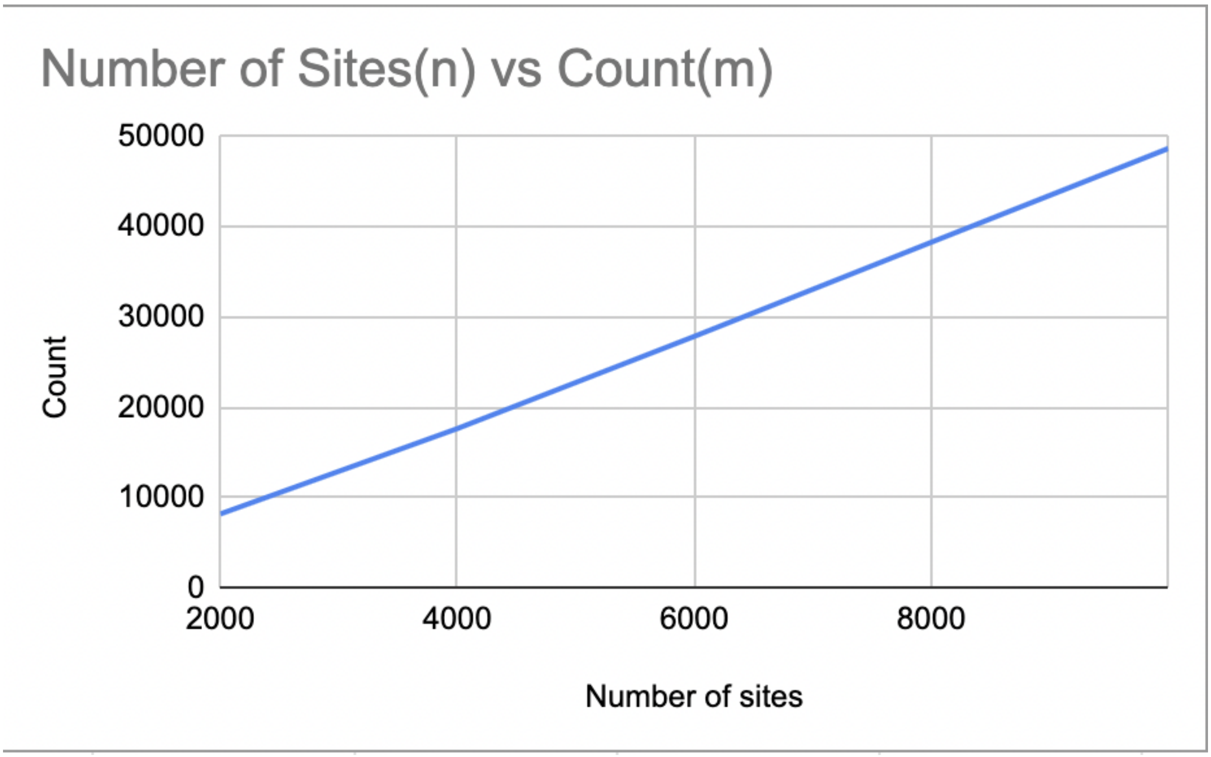
<u>M = 0.375(nlogn) (approx) | Number of connections = N - 1 (deduced from the graph and values) and number of connections is 1 less than the number of objects.</u>

Evidence / Graph :

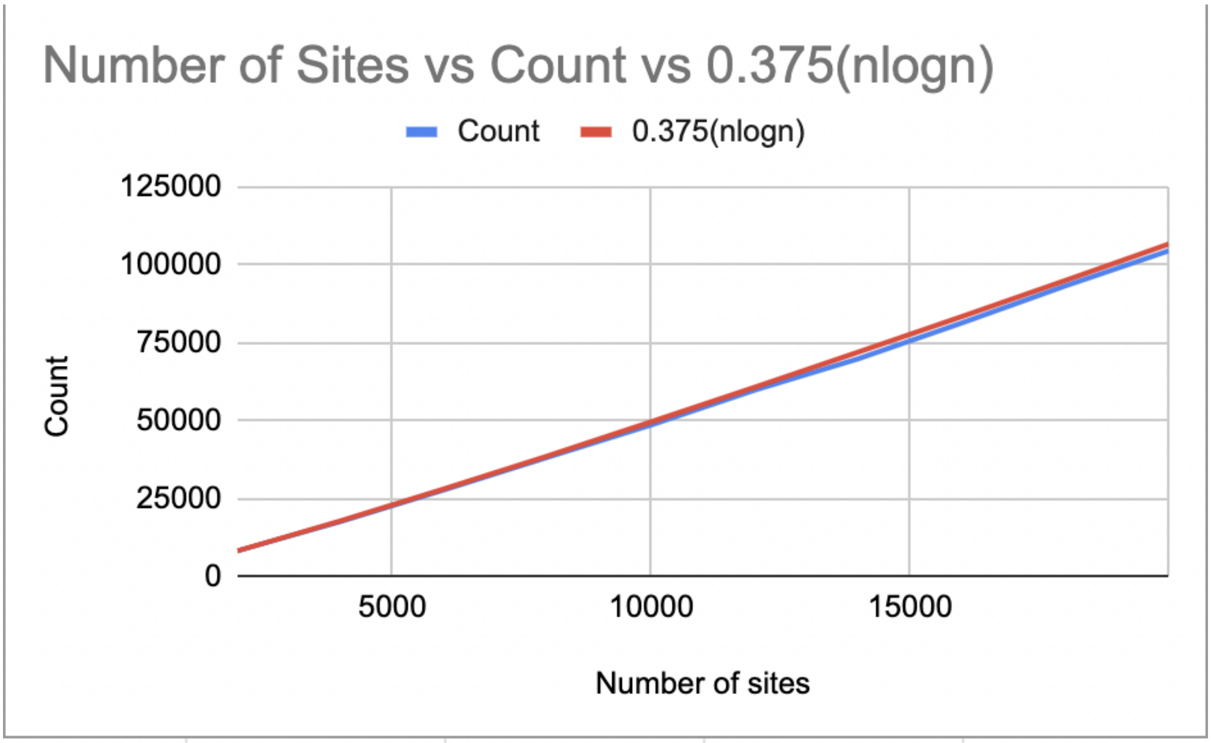


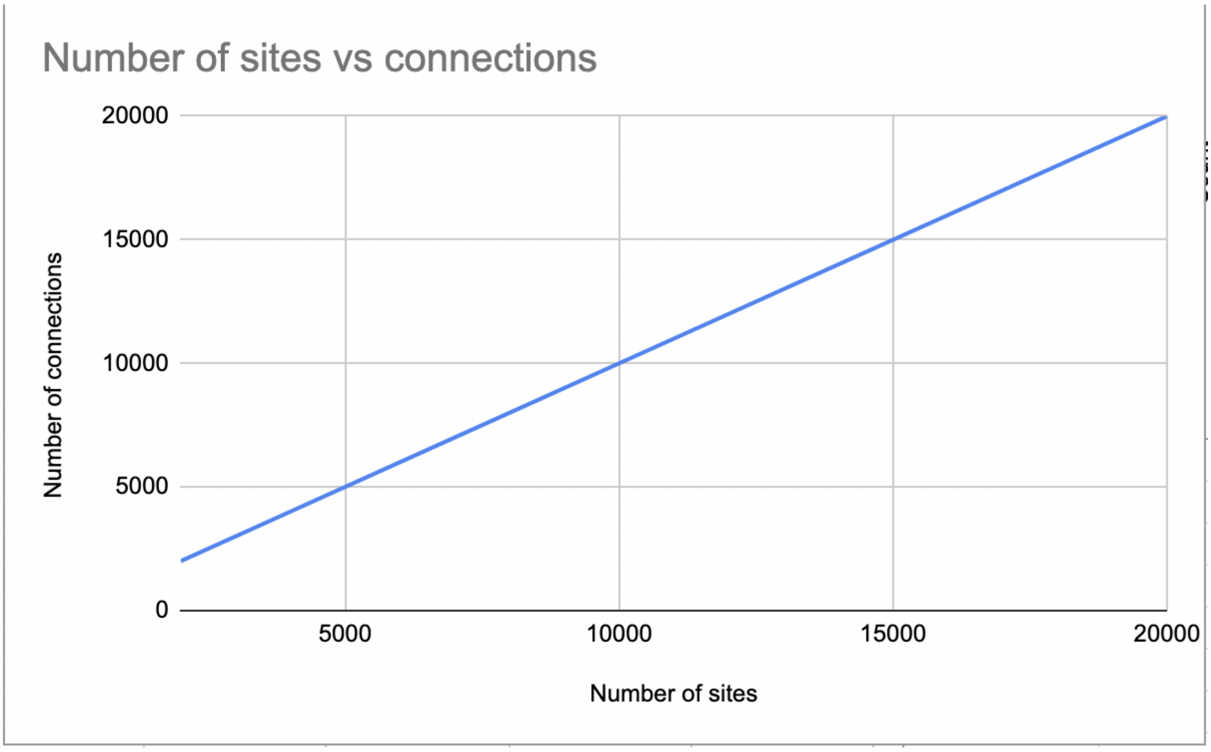
Number of sites	Count	logn	nlogn	0.375(nlogn)
2000	8187	10.96578428	21931.56857	8186.475763
4000	17646	11.96578428	47863.13714	17866.04596
6000	27863	12.55074679	75304.48071	28109.17533
8000	38280	12.96578428	103726.2743	38718.28081
10000	48681	13.28771238	132877.1238	49599.52363
12000	59775	13.55074679	162608.9614	60697.63398
14000	69897	13.77313921	192823.9489	71976.09141
16000	81344	13.96578428	223452.5486	83408.93936
18000	93245	14.13570929	254442.7671	94976.77012
20000	104506	14.28771238	285754.2476	106664.5194

Graph which says Number of sites (n) is linear to Number of pairs(count/m)



Graph which says Number of pairs(count/m) and 0.375(nlogn) are almost equal:


Number of Sites vs Count vs 0.375(nlogn)
Count
0.375(nlogn)
Count
Number of sites


Number of sites vs connections
Number of connections
Number of sites