

# Program Structures & Algorithms

## Spring 2022

### Assignment No. 3

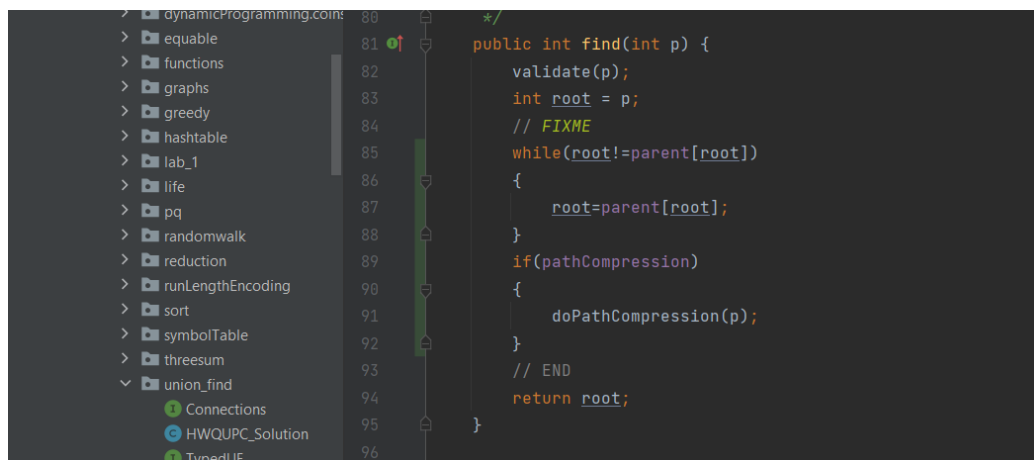
Name: Shashank Siripragada  
NUID: 002193773

#### Task

#### Step 1:

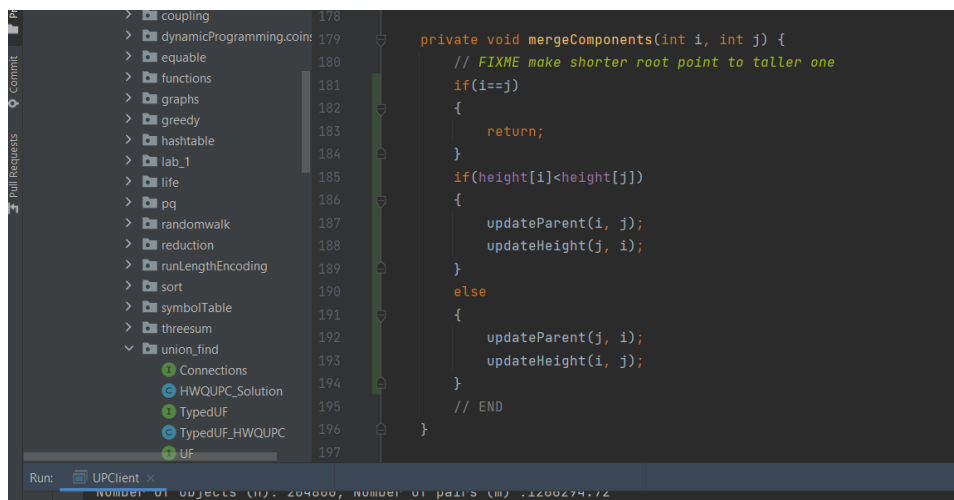
(a) Implement height-weighted Quick Union with Path Compression.

#### Find:



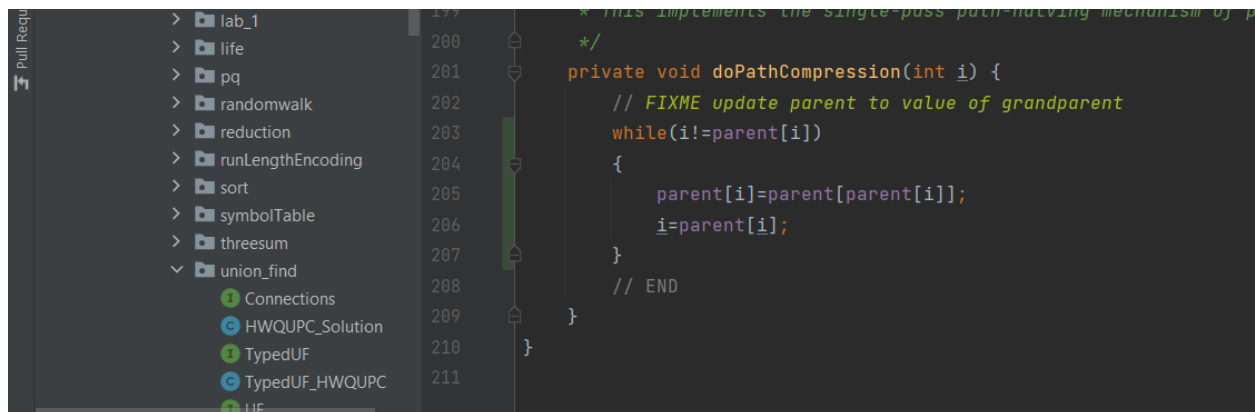
```
80  */
81  public int find(int p) {
82      validate(p);
83      int root = p;
84      // FIXME
85      while(root != parent[root])
86      {
87          root = parent[root];
88      }
89      if(pathCompression)
90      {
91          doPathCompression(p);
92      }
93      // END
94      return root;
95  }
96  }
```

#### mergeComponents:



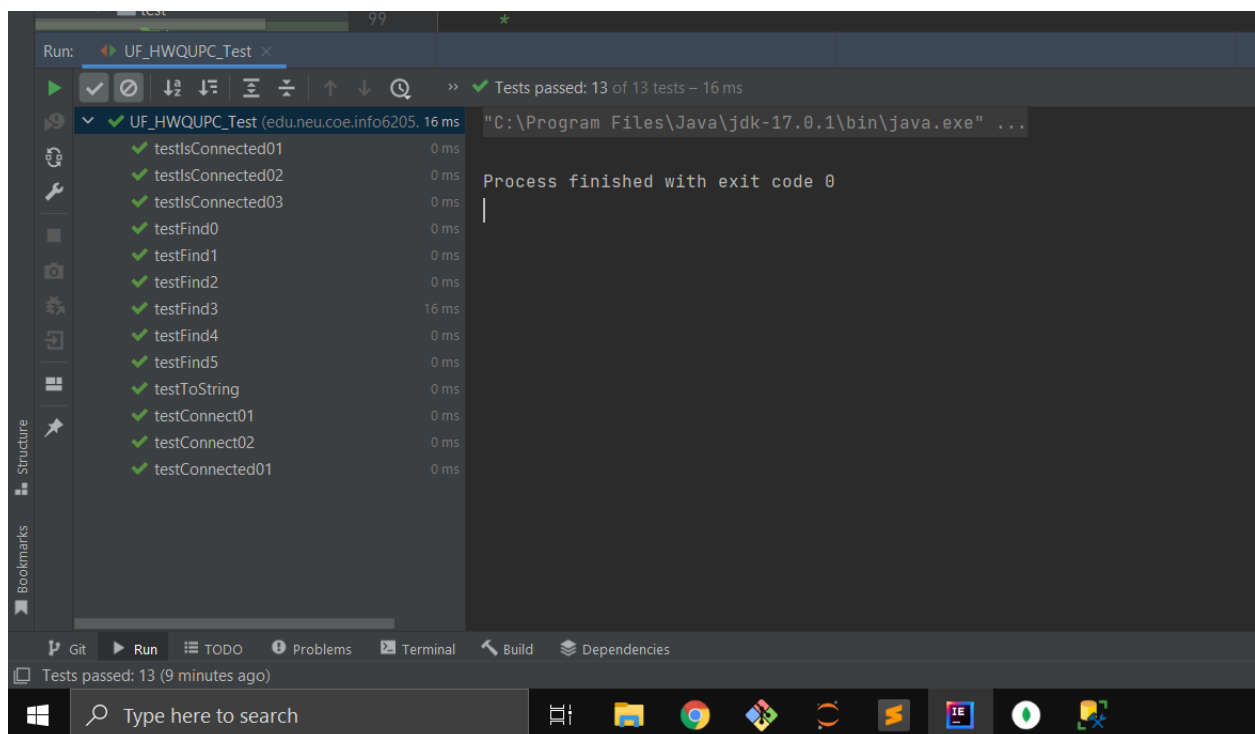
```
178
179  private void mergeComponents(int i, int j) {
180      // FIXME make shorter root point to taller one
181      if(i == j)
182      {
183          return;
184      }
185      if(height[i] < height[j])
186      {
187          updateParent(i, j);
188          updateHeight(j, i);
189      }
190      else
191      {
192          updateParent(j, i);
193          updateHeight(i, j);
194      }
195      // END
196  }
197  }
```

## pathCompression:



(b) Check that the unit tests for this class all work.

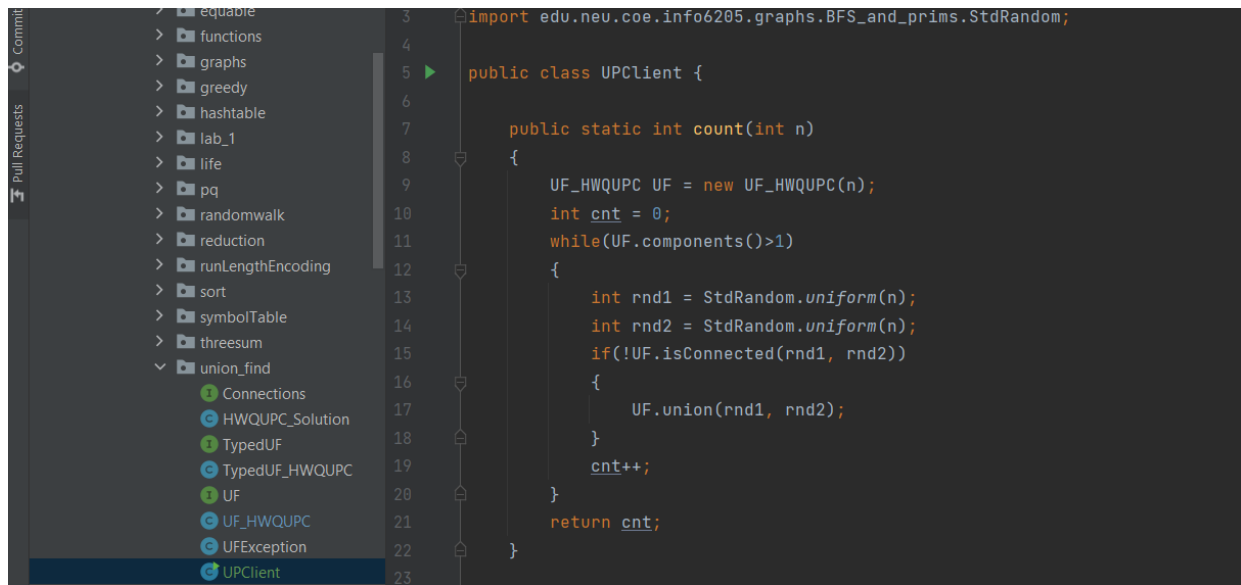
## UF\_HWQUPC\_Test



## Step 2:

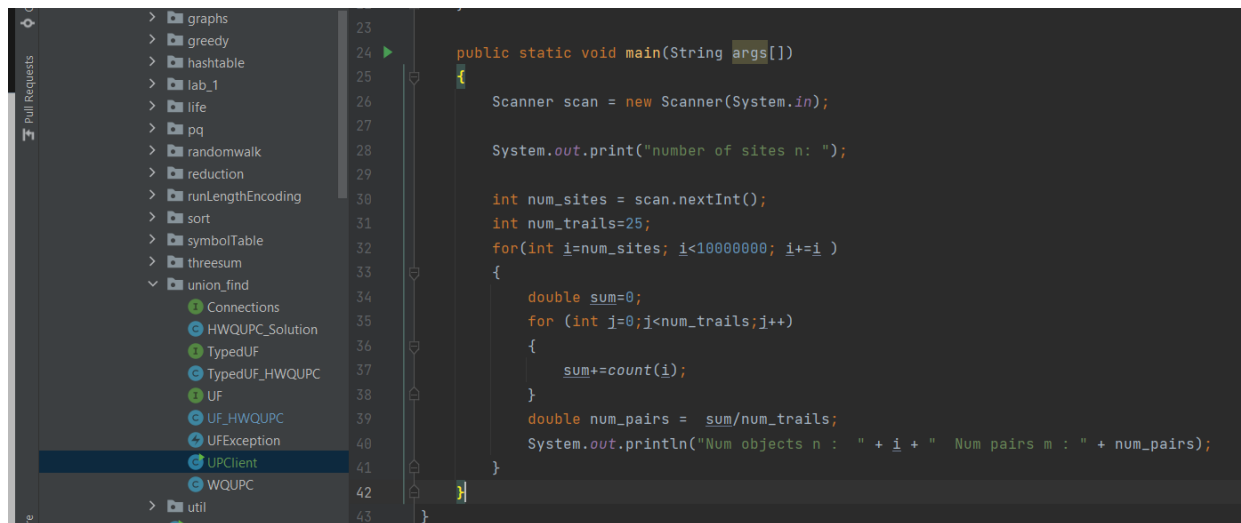
Using your implementation of UF\_HWQUPC, develop a UF ("union-find") client that takes an integer value *n* from the command line to determine the number of "sites."

## Implemented UPClient



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a directory structure with folders like 'equable', 'functions', 'graphs', 'greedy', 'hashtable', 'lab\_1', 'life', 'pq', 'randomwalk', 'reduction', 'runLengthEncoding', 'sort', 'symbolTable', 'threesum', and 'union\_find'. The 'union\_find' folder is expanded, showing files like 'Connections', 'HWQUPC\_Solution', 'TypedUF', 'TypedUF\_HWQUPC', 'UF', 'UF\_HWQUPC', 'UFException', and 'UPClient'. The 'UPClient' file is selected. The code editor shows the implementation of the 'UPClient' class. It starts with an import statement for 'StdRandom' from 'edu.neu.coe.info6205.graphs.BFS\_and\_prims'. The class 'UPClient' has a public static method 'count(int n)' which returns an integer. The method initializes a 'UF\_HWQUPC' object 'UF' with 'n' sites, sets a counter 'cnt' to 0, and enters a while loop that continues as long as 'UF.components()' is greater than 1. Inside the loop, it generates two random integers 'rnd1' and 'rnd2' using 'StdRandom.uniform(n)', checks if they are connected using 'UF.isConnected(rnd1, rnd2)', and if not, it unions them using 'UF.union(rnd1, rnd2)'. The counter 'cnt' is incremented each time a union is performed. Finally, the method returns 'cnt'.

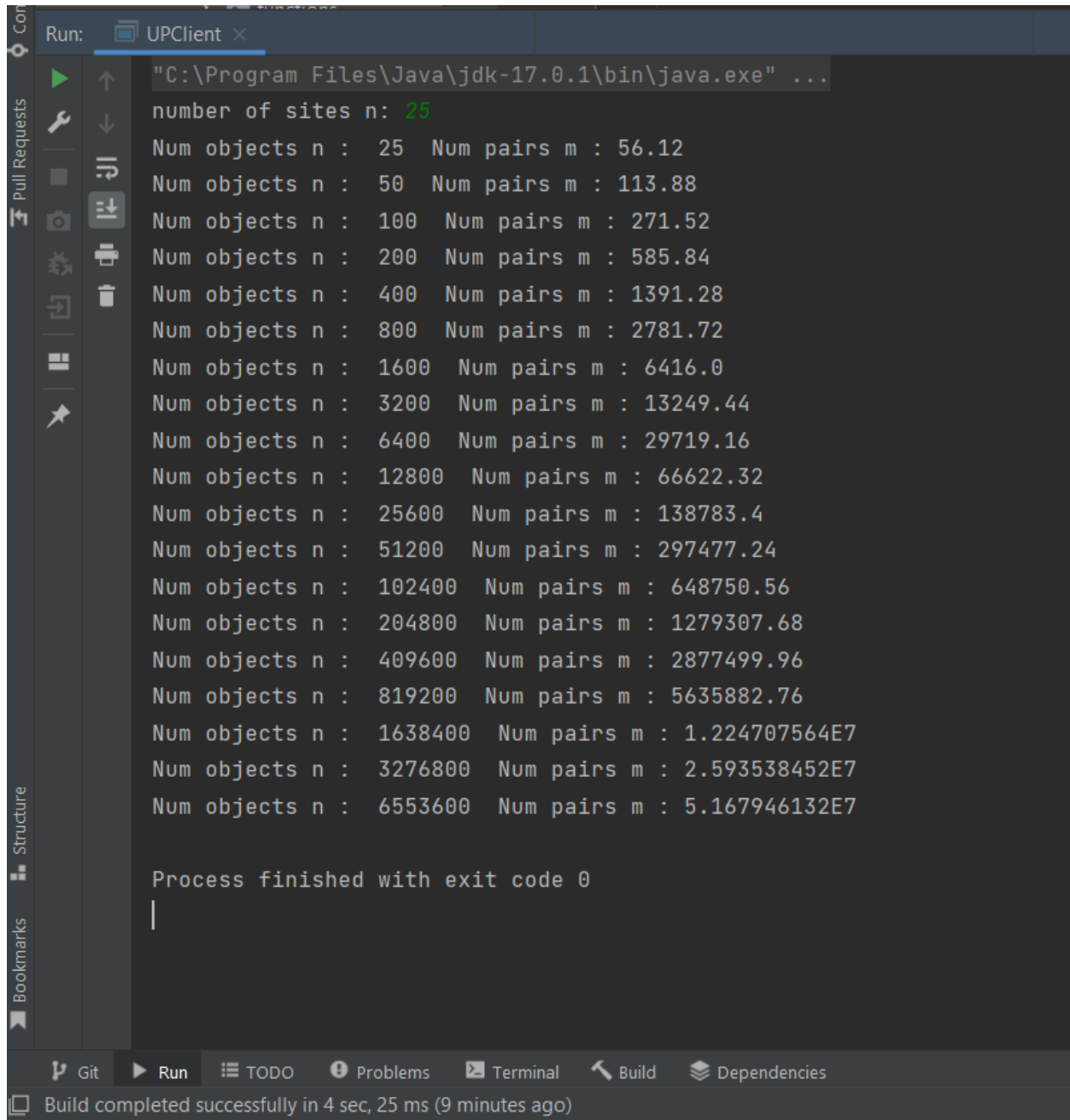
```
3 import edu.neu.coe.info6205.graphs.BFS_and_prims.StdRandom;
4
5 public class UPClient {
6
7     public static int count(int n)
8     {
9         UF_HWQUPC UF = new UF_HWQUPC(n);
10        int cnt = 0;
11        while(UF.components()>1)
12        {
13            int rnd1 = StdRandom.uniform(n);
14            int rnd2 = StdRandom.uniform(n);
15            if(!UF.isConnected(rnd1, rnd2))
16            {
17                UF.union(rnd1, rnd2);
18            }
19            cnt++;
20        }
21        return cnt;
22    }
23 }
```



The screenshot shows the same IDE as the previous one, but now the 'main' method of the 'UPClient' class is visible. The 'main' method takes an array of strings 'args' as input. It creates a 'Scanner' object 'scan' from 'System.in'. It prints a prompt 'number of sites n: ' and reads an integer 'num\_sites' from the scanner. It sets 'num\_trails' to 25. It then enters a for loop that iterates from 'num\_sites' to 100000000, incrementing by 1. Inside the loop, it initializes a 'double sum' to 0 and enters a for loop that iterates from 0 to 'num\_trails', incrementing by 1. Inside this inner loop, it calls 'count(i)' and adds the result to 'sum'. After the inner loop, it calculates 'num\_pairs' as 'sum/num\_trails' and prints the result: 'Num objects n : ' + i + ' Num pairs m : ' + num\_pairs. The 'main' method ends with a closing brace.

```
23
24 public static void main(String args[])
25 {
26     Scanner scan = new Scanner(System.in);
27
28     System.out.print("number of sites n: ");
29
30     int num_sites = scan.nextInt();
31     int num_trails=25;
32     for(int i=num_sites; i<100000000; i+=1 )
33     {
34         double sum=0;
35         for (int j=0;j<num_trails;j++)
36         {
37             sum+=count(i);
38         }
39         double num_pairs = sum/num_trails;
40         System.out.println("Num objects n : " + i + " Num pairs m : " + num_pairs);
41     }
42 }
43 }
```

## Run Results:



```
Run: UPClient x
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" ...
number of sites n: 25
Num objects n : 25  Num pairs m : 56.12
Num objects n : 50  Num pairs m : 113.88
Num objects n : 100  Num pairs m : 271.52
Num objects n : 200  Num pairs m : 585.84
Num objects n : 400  Num pairs m : 1391.28
Num objects n : 800  Num pairs m : 2781.72
Num objects n : 1600  Num pairs m : 6416.0
Num objects n : 3200  Num pairs m : 13249.44
Num objects n : 6400  Num pairs m : 29719.16
Num objects n : 12800  Num pairs m : 66622.32
Num objects n : 25600  Num pairs m : 138783.4
Num objects n : 51200  Num pairs m : 297477.24
Num objects n : 102400  Num pairs m : 648750.56
Num objects n : 204800  Num pairs m : 1279307.68
Num objects n : 409600  Num pairs m : 2877499.96
Num objects n : 819200  Num pairs m : 5635882.76
Num objects n : 1638400  Num pairs m : 1.224707564E7
Num objects n : 3276800  Num pairs m : 2.593538452E7
Num objects n : 6553600  Num pairs m : 5.167946132E7

Process finished with exit code 0
|
```

Build completed successfully in 4 sec, 25 ms (9 minutes ago)

### Step 3:

Determine the relationship between the number of objects ( $n$ ) and the number of pairs ( $m$ ).

**Relationship:**

The relationship between the number of objects (n) and the number of pairs (m) generated to reduce the number of components from  $n$  to 1 is:

$$m = f(n) = 1/2 \times n \times \ln(n)$$

Where,

m = number of pairs

n = number of objects

**Evidence:**

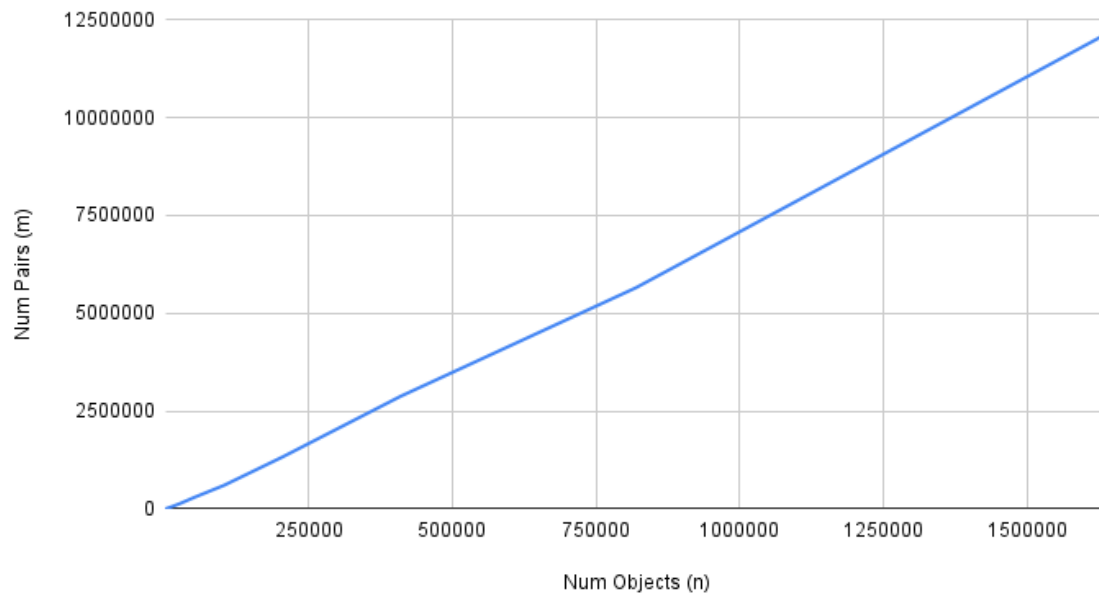
With the initial value of n as 25 and using the doubling method, we can calculate the num pairs (m) generated to reduce the number of components from n to 1.

Below is the data from the run.

Num Objects (n)	Num Pairs (m)	$0.5 \times n \times \ln(n)$
25	57.2	40.23594781
50	116.16	97.80057514
100	272.28	230.2585093
200	550.92	529.8317367
400	1323.92	1198.292909
800	2671.76	2673.844691
1600	6121.36	5902.207127
3200	14588.72	12913.44974
6400	31000.96	28044.97046
12800	63623.04	60526.08288
25600	134619.16	129924.4497
51200	297053.76	277593.4672
102400	601920.52	590676.07
204800	1331145.48	1252330.411
409600	2877144.36	2646617.365
819200	5652306.36	5577147.815
1638400	1.21E+07	11722121.8
3276800	2.59E+07	24579895.94
6553600	5.17E+07	51431096.57

The below diagrams show the result of plotting the above table data, on a standard scale, with the number of objects ( $n$ ) on the Xaxis and number of pairs ( $m$ ) generated to reduce the number of components from  $n$  to 1 on the Yaxis.

Num Pairs ( $m$ ) vs. Num Objects ( $n$ )



Num Objects vs.  $0.5 \cdot n \cdot \ln(n)$

