

**Sneha Ravichandran (001096251)**  
**Program Structures & Algorithms INFO 6205**  
**Fall 2021**  
**Assignment 3**

**Task:**

**Part 1:**

1. Implement height-weighted Quick Union with Path Compression
2. Implement doPathCompression(), mergeComponents(), find()
3. Pass all the test cases

**Part 2:**

4. Create a main program to generate site pairs and union them if they are not connected
5. Return the number of pairs generated to bring n components to 1 component

**Part 3:**

6. Form a relationship between n(sites) and the number of pairs generated(m)

**Part 1:**

**Code for To do implementation:**

doPathCompression() method

```
private void doPathCompression(int i) {  
    // TO BE IMPLEMENTED update parent to value of grandparent  
    parent[i]=parent[parent[i]];  
}
```

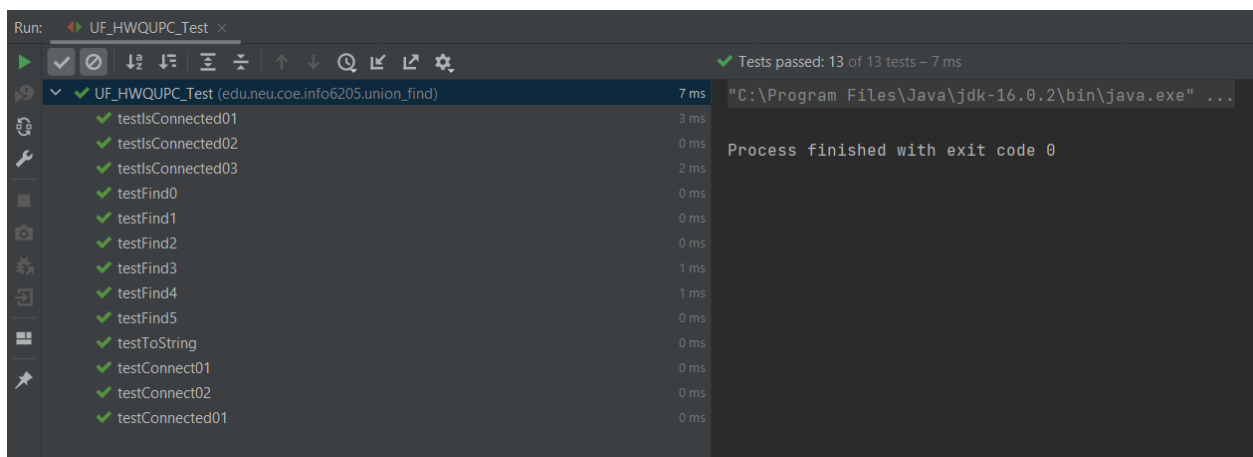
mergeComponents() method:

```
private void mergeComponents(int i, int j) {  
    // TO BE IMPLEMENTED make shorter root point to taller one  
    int findi=find(i);  
    int findj=find(j);  
    if(findi!=findj)  
    {  
        if(height[findi]>=height[findj])  
        {  
            parent[findj]=findi;  
            height[i]+=height[j];  
        }  
        else{  
            parent[findi]=findj;  
            height[j]+=height[i];  
        }  
    }  
}
```

Find() method:

```
public int find(int p) {
    validate(p);
    int root = p;
    // TO BE IMPLEMENTED
    while(parent[root]!=root)
    {
        if(pathCompression==true)
        {
            doPathCompression(root);
            //System.out.println();
        }
        root=parent[root];
    }
    return root;
}
```

Test Cases passed for Union find:



The screenshot shows the Run window of an IDE. At the top, it says "Run: UF\_HWQUPC\_Test". Below this is a toolbar with various icons. The main area displays a list of test cases, all of which are marked with a green checkmark, indicating they passed. The test cases are: testIsConnected01, testIsConnected02, testIsConnected03, testFind0, testFind1, testFind2, testFind3, testFind4, testFind5, testToString, testConnect01, testConnect02, and testConnected01. To the right of the test case names, the execution time for each is listed. The total execution time for all tests is 7 ms. At the bottom right, it says "Tests passed: 13 of 13 tests - 7 ms" and "Process finished with exit code 0".

Test Case	Execution Time
UF_HWQUPC_Test (edu.neu.coe.info6205.union_find)	7 ms
testIsConnected01	3 ms
testIsConnected02	0 ms
testIsConnected03	2 ms
testFind0	0 ms
testFind1	0 ms
testFind2	0 ms
testFind3	1 ms
testFind4	1 ms
testFind5	0 ms
testToString	0 ms
testConnect01	0 ms
testConnect02	0 ms
testConnected01	0 ms

## Part 2: Main program implementation:

```
package edu.neu.coe.info6205.union_find;
import java.util.Random;
public class UFmain {
    public static void main(String[] args)
    {
        /*arraysites has n values*/
        int arraysites[]={5000,10000,20000,40000,80000,160000,320000};
        for(int i=0;i<arraysites.length;i++) {
            int run=0;
            int n1=0,n2=0;
            UF_HWQUPC compobj = new UF_HWQUPC(arraysites[i], pathCompression: false);
            Random random = new Random();
            while (true) {
                run++;
                n1 = random.nextInt(arraysites[i]);
                //System.out.println(n1);
                n2 = random.nextInt(arraysites[i]);
                //System.out.println(n2);
                if (compobj.connected(n1, n2) == false) {
                    compobj.union(n1, n2);
                }
                if (compobj.components() == 1) {
                    System.out.println(run+" for " +arraysites[i]);
                    break;
                }
            }
        }
    }
}
```

## Output:



```
Run: UFmain x
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...
23094 pairs generated for 5000 sites
47963 pairs generated for 10000 sites
129558 pairs generated for 20000 sites
229420 pairs generated for 40000 sites
430420 pairs generated for 80000 sites
1039487 pairs generated for 160000 sites
2184390 pairs generated for 320000 sites

Process finished with exit code 0
```

## Part 3:

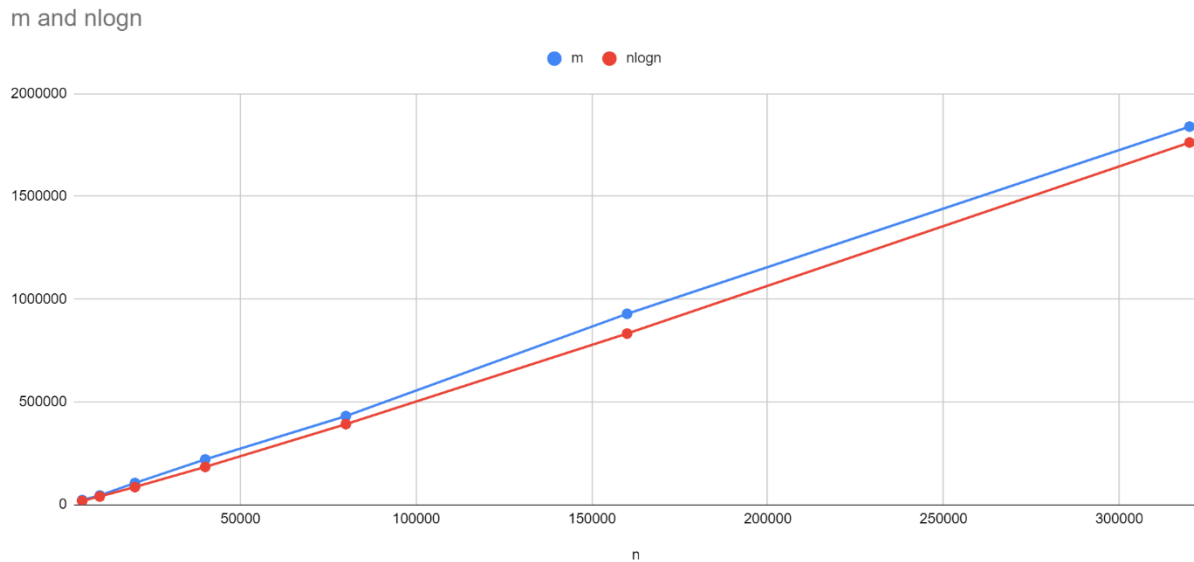
### Forming the relationship between n (number of sites) and m (number of pairs generated):

I used the doubling method and took n as 5000, 10000, 20000, 40000, 80000, 160000, 320000  
Ran the program 4 times for all the values of sites and calculated the average.

From the above calculations we can see that the m value is more closer to  $n \log(n)$  than  $n^2$  or  $n$  or  $\log(n)$

A	B	C	D	E	F	G	H
n	m	nlogn					
5000	22842	18495					
10000	45169	40000					
20000	105870	86021					
40000	220420	184082					
80000	431430	392247					
160000	928729	832659					
320000	1839052	1761648					
n=5000			n=10000				
1	22842		1	54997			
2	20689		2	46370			
3	23897		3	42286			
4	27587		4	45004			
avg	23753.75		avg	47164.25			
n=20000			n=40000			n=80000	
1	105870		1	220420		1	431430
2	88320		2	220528		2	439790
3	110156		3	210588		3	417530
4	101597		4	210345		4	448237
avg	101485.8		avg	215470.3		avg	434246.8
n=160000			n=320000				
1	928729		1	1839052			
2	1051771		2	2420919			
3	1014254		3	1989307			
4	882999		4	1890712			
avg	969438.3		avg	2034998			

## Evidence:



From the above plotted graph we can see that m is directly proportional to a constant times nlogn

## Conclusion:

Hence, we arrive at a conclusion that the total number of pairs (m) generated is directly proportional to nlog(n) times a constant(k), where n are the number of sites or objects.

$$m \propto n \log(n)$$

or

$$m = k n \log(n)$$