Program Structures & Algorithms
Spring 2022
Assignment No. 4

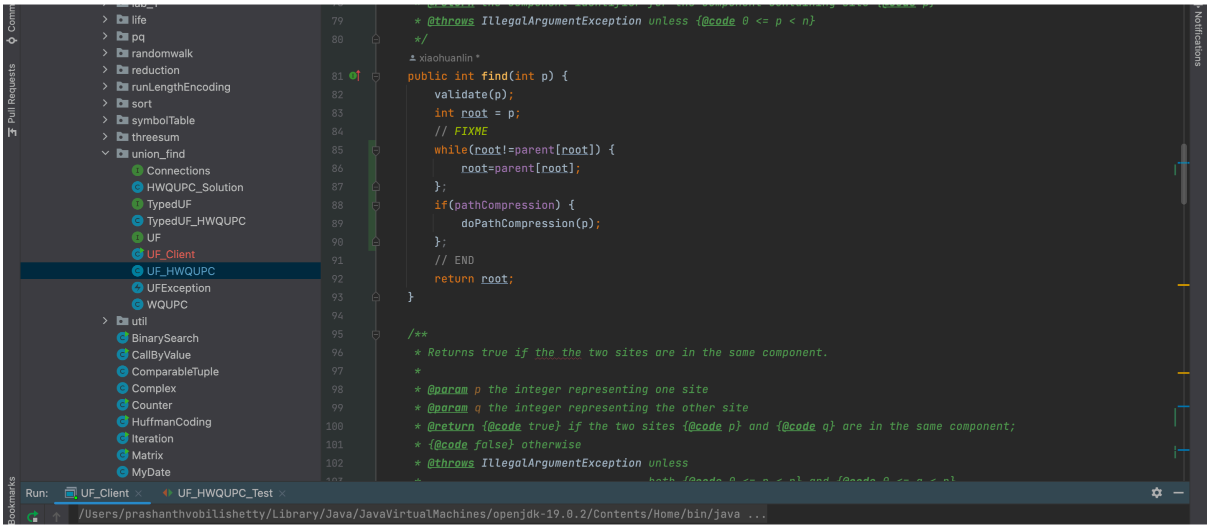
Name : V.Prashanth
NUID : 002707220

Tasks Performed:

Implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

Step 1:

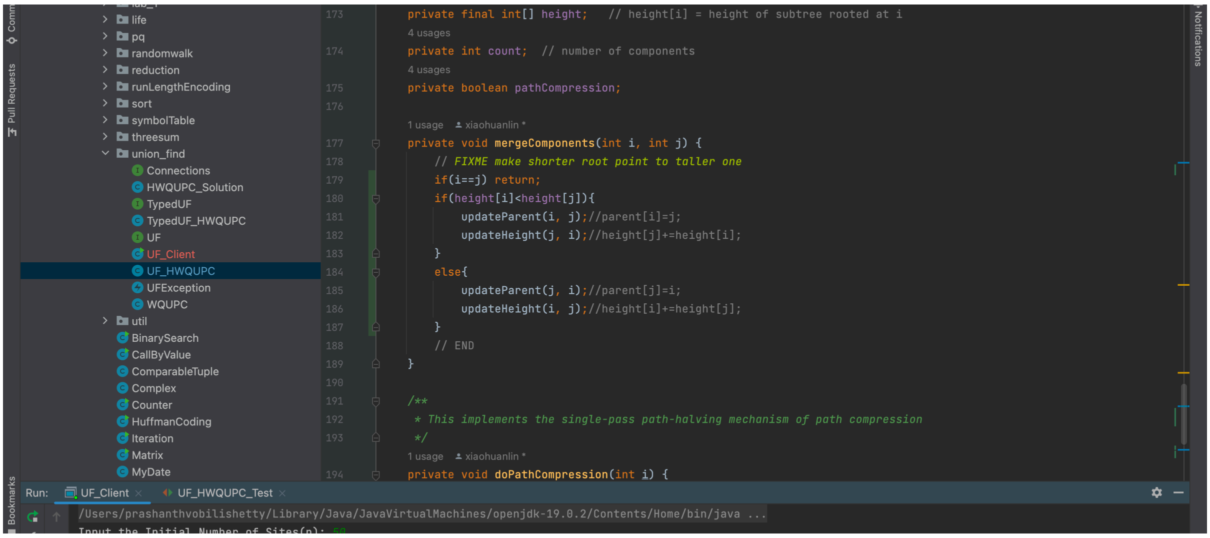
Implement a height-weighted Quick Union with Path Compression



```java
     * @throws IllegalArgumentException unless {@code 0 <= p < n}
     */
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        while(root!=parent[root]) {
            root=parent[root];
        };
        if(pathCompression) {
            doPathCompression(p);
        };
        // END
        return root;
    }

    /**
     * Returns true if the the two sites are in the same component.
     *
     * @param p the integer representing one site
     * @param q the integer representing the other site
     * @return {@code true} if the two sites {@code p} and {@code q} are in the same component;
     * {@code false} otherwise
     * @throws IllegalArgumentException unless
```

```
/Users/prashanthvobilishetty/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java ...
```

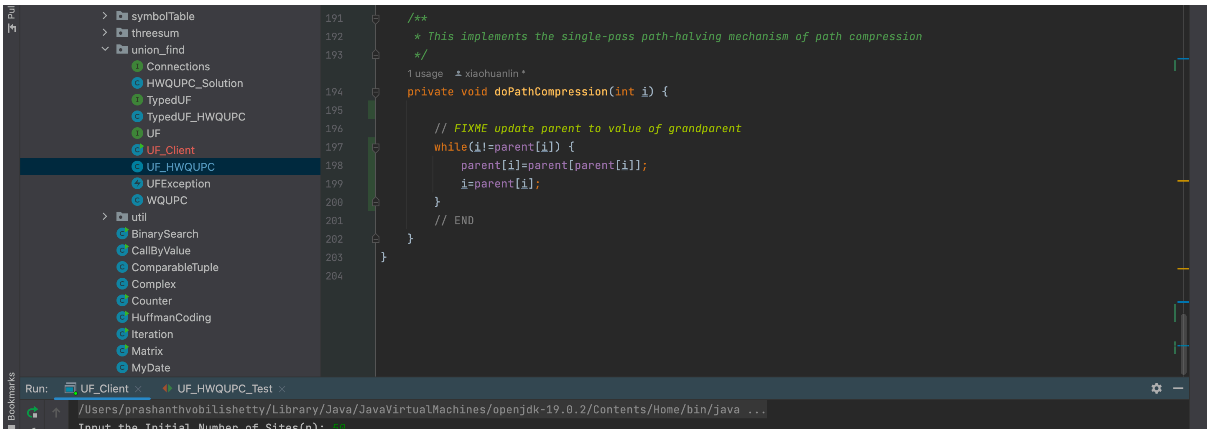
Merge Components Methods:



```
    private final int[] height;   // height[i] = height of subtree rooted at i
    4 usages
    private int count;  // number of components
    4 usages
    private boolean pathCompression;

    1 usage  xiaohuanlin *
    private void mergeComponents(int i, int j) {
        // FIXME make shorter root point to taller one
        if(i==j) return;
        if(height[i]<height[j]){
            updateParent(i, j);//parent[i]=j;
            updateHeight(j, i);//height[j]+=height[i];
        }
        else{
            updateParent(j, i);//parent[j]=i;
            updateHeight(i, j);//height[i]+=height[j];
        }
        // END
    }

    /**
     * This implements the single-pass path-halving mechanism of path compression
     */
    1 usage  xiaohuanlin *
    private void doPathCompression(int i) {
```

```
/Users/prashanthvobilishetty/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java ...
Input the Initial Number of Sites(n): 50
```

doPathCompression method :

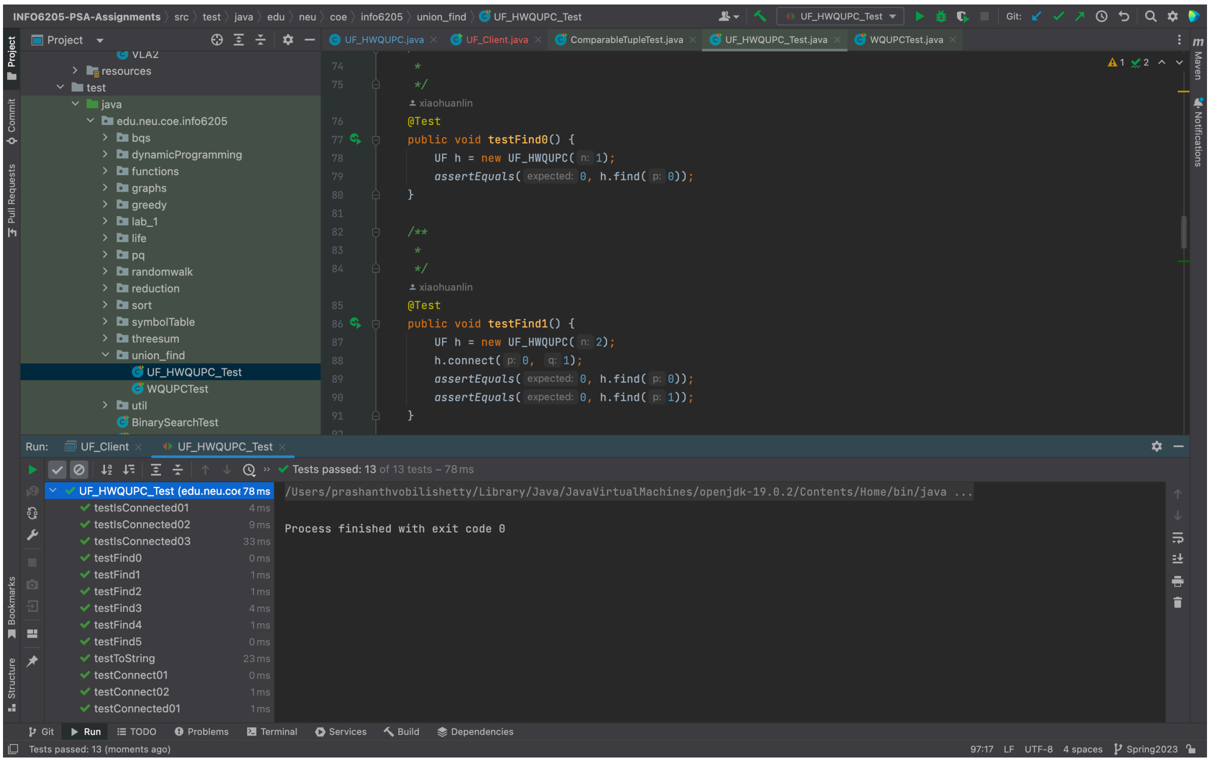


```
    /**
     * This implements the single-pass path-halving mechanism of path compression
     */
    1 usage  xiaohuanlin *
    private void doPathCompression(int i) {

        // FIXME update parent to value of grandparent
        while(i!=parent[i]) {
            parent[i]=parent[parent[i]];
            i=parent[i];
        }
        // END
    }
}

```

```
/Users/prashanthvobilishetty/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java ...
Input the Initial Number of Sites(n): 50
```

Unit Test Screenshots:

UF_HWQUPC_Test.java



INFO6205-PSA-Assignments 〉 src 〉 test 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 union_find 〉 UF_HWQUPC_Test

```java
     *
     */
    @Test
    public void testFind0() {
        UF h = new UF_HWQUPC(1);
        assertEquals(0, h.find(0));
    }

    /**
     *
     */
    @Test
    public void testFind1() {
        UF h = new UF_HWQUPC(2);
        h.connect(0, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
    }
```

Tests passed: 13 of 13 tests – 78 ms

```
UF_HWQUPC_Test (edu.neu.coe 78 ms
  testIsConnected01   4 ms
  testIsConnected02   9 ms
  testIsConnected03  33 ms
  testFind0           0 ms
  testFind1           1 ms
  testFind2           1 ms
  testFind3           4 ms
  testFind4           1 ms
  testFind5           0 ms
  testToString       23 ms
  testConnect01       0 ms
  testConnect02       1 ms
  testConnected01     1 ms
```

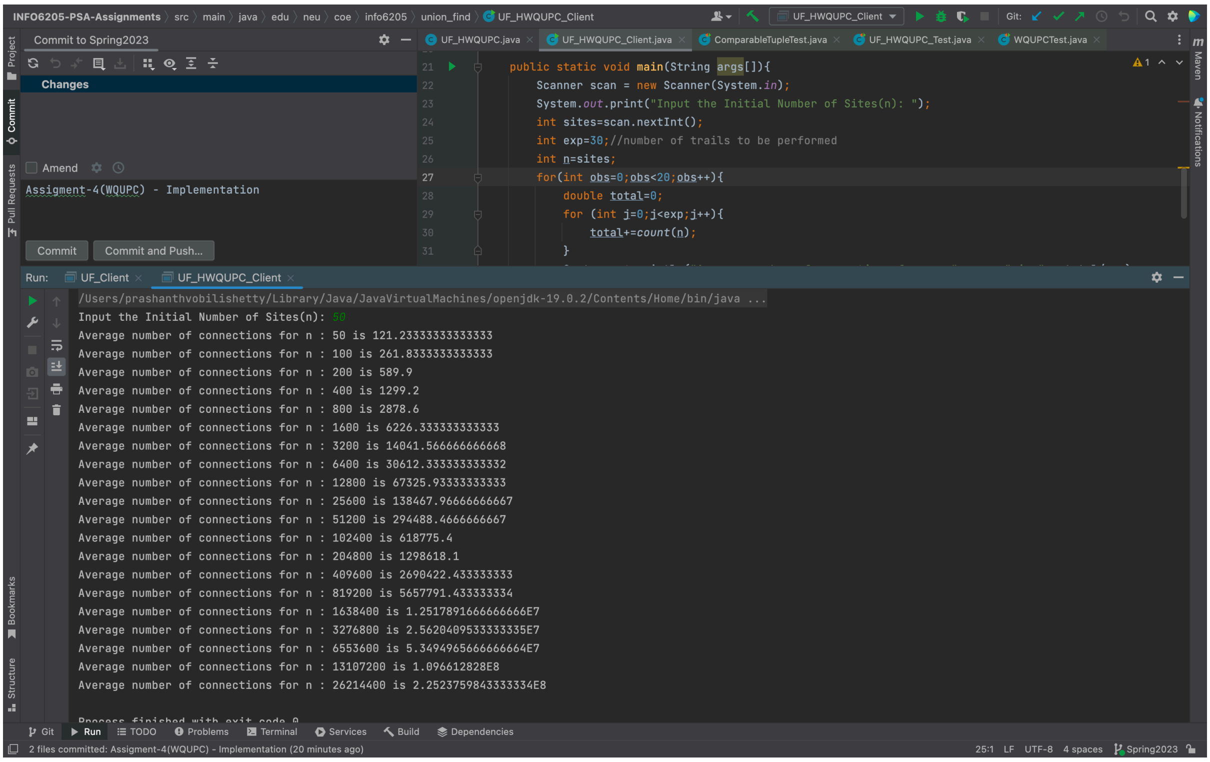
```
/Users/prashanthvobilishetty/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java ...

Process finished with exit code 0
```

Tests passed: 13 (moments ago)

Step 2 :

Using the implementation of UF_HWQUPC, develop a UF("union-find) client

Output:



INFO6205-PSA-Assignments 〉 src 〉 main 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 union_find 〉 UF_HWQUPC_Client

Commit to Spring2023

Assigment-4(WQUPC) - Implementation

```java
    public static void main(String args[]){
        Scanner scan = new Scanner(System.in);
        System.out.print("Input the Initial Number of Sites(n): ");
        int sites=scan.nextInt();
        int exp=30;//number of trails to be performed
        int n=sites;
        for(int obs=0;obs<20;obs++){
            double total=0;
            for (int j=0;j<exp;j++){
                total+=count(n);
            }
```

```
/Users/prashanthvobilishetty/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java ...
Input the Initial Number of Sites(n): 50
Average number of connections for n : 50 is 121.23333333333333
Average number of connections for n : 100 is 261.8333333333333
Average number of connections for n : 200 is 589.9
Average number of connections for n : 400 is 1299.2
Average number of connections for n : 800 is 2878.6
Average number of connections for n : 1600 is 6226.333333333333
Average number of connections for n : 3200 is 14041.566666666668
Average number of connections for n : 6400 is 30612.333333333332
Average number of connections for n : 12800 is 67325.93333333333
Average number of connections for n : 25600 is 138467.96666666667
Average number of connections for n : 51200 is 294488.4666666667
Average number of connections for n : 102400 is 618775.4
Average number of connections for n : 204800 is 1298618.1
Average number of connections for n : 409600 is 2690422.433333333
Average number of connections for n : 819200 is 5657791.433333334
Average number of connections for n : 1638400 is 1.2517891666666666E7
Average number of connections for n : 3276800 is 2.5620409533333335E7
Average number of connections for n : 6553600 is 5.3494965666666664E7
Average number of connections for n : 13107200 is 1.096612828E8
Average number of connections for n : 26214400 is 2.2523759843333334E8

Process finished with exit code 0
```

2 files committed: Assigment-4(WQUPC) - Implementation (20 minutes ago)

Objects(n)	Pairs(m)
50	
100	121.23
200	261.83
400	589.9
800	1299.2
1600	2878.6
3200	6226.33
6400	14041.56
12800	30612.33
25600	67325.933
51200	138467.97
102400	294488.47
204800	618775.4
409600	1298618.1
819200	5657791.43

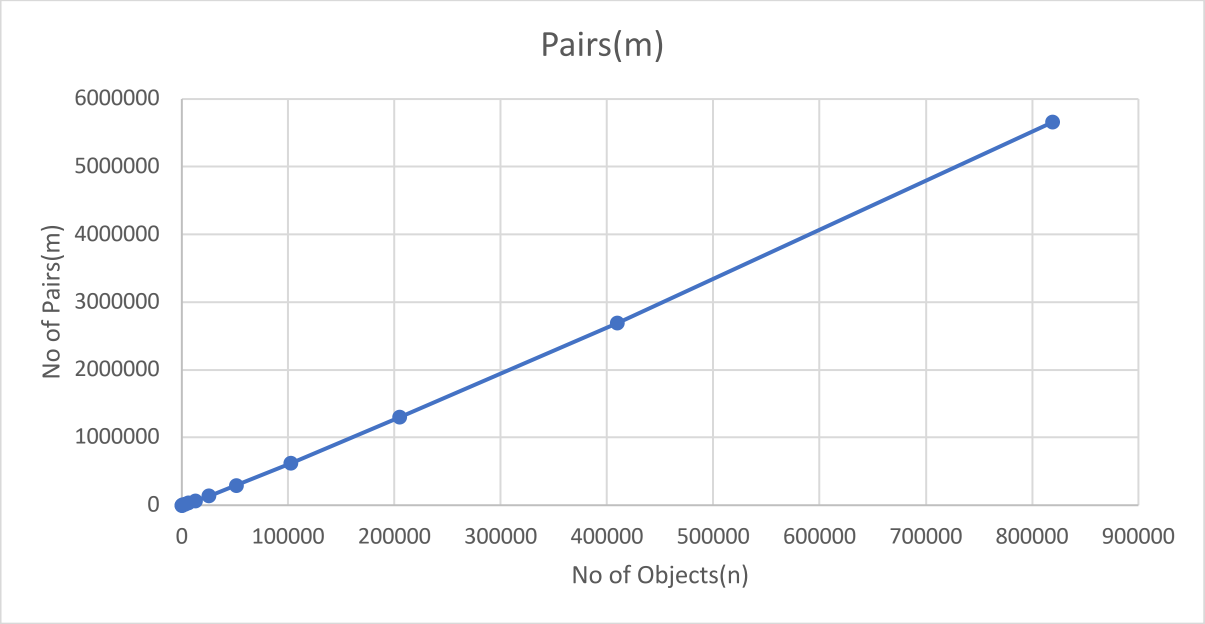
Relationship Conclusion:

- The number of pairs generated to connect all the sites is linearly proportional to the log scale of no. of sites. I.e., m is proportional to nlogn.

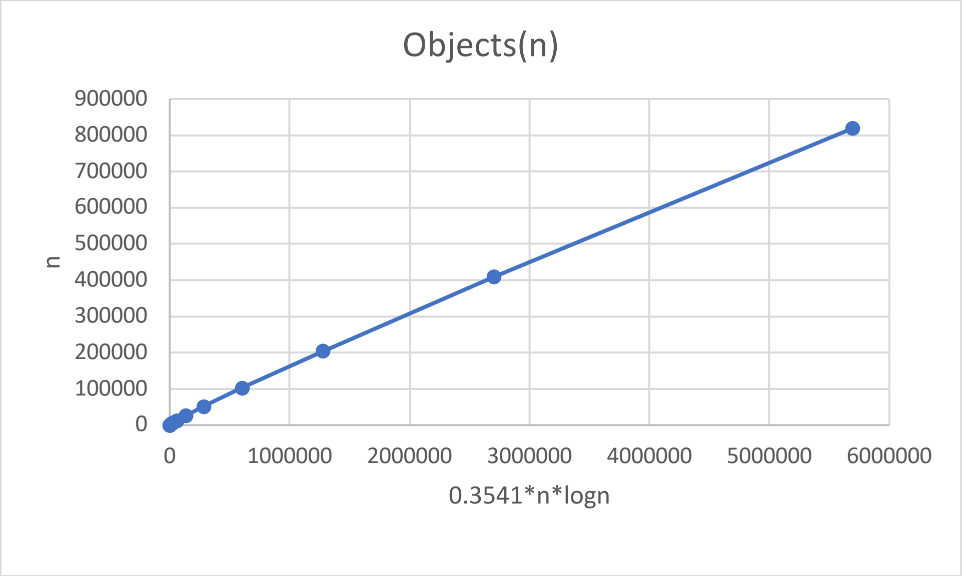
- By averaging out the constant factor, it is found that the average constant factor is 0.354

Objects(n)	**Pairs(m)**	**n*log(n,2)**	**Average**		**0.3541*n*logn**
50	121.23	282.192809	0.42959989		99.8962546
100	261.83	664.385619	0.39409342		235.192509
200	589.9	1528.77124	0.38586545		541.185018
400	1299.2	3457.54248	0.37575822		1223.97004
800	2878.6	7715.08495	0.3731132		2731.14007
1600	6226.33	17030.1699	0.36560587		6028.68015
3200	14041.56	37260.3398	0.37685003		13190.1603
6400	30612.33	80920.6796	0.37830046		28645.9206
12800	67325.933	174641.359	0.38550967		61823.0412
25600	138467.97	374882.718	0.36936344		132708.482
51200	294488.47	800965.437	0.36766689		283541.765
102400	618775.4	1704330.87	0.36306061		603333.129
204800	1298618.1	3613461.75	0.35938338		1279165.46
409600	2690422.43	7636523.5	0.35230985	Average	2703329.32
819200	567791.43	16092247	0.03528354	0.35411759	5696655.43

Graph of No of Objects(n) V/S No of Pairs(m)



n vs 0.354 n*logn


Objects(n)
n
0.3541*n*logn