```
================================
= Created By: Frendy Lio
= Date: August 31th
================================
```

Analysis of running time and memory usage:

```
==========================
1 - Use the stopwatch data type from stdlib.jar to measure the total running time of
PercolationStats. How does doubling N affect the total running time? How does
doubling T affect the total running time? Give a formula (using tilde notation) of
the total running time on your computer (in seconds) as a single function of both N
and T.
==========================
```

N	T
100	1-> 0.101
200	1-> 0.114
400	1-> 0.168
800	1-> 0.304
1600	1-> 0.781
3200	1-> 3.741
6400	1-> 16.925

T	N
100	1-> 0.083
200	1-> 0.085
400	1-> 0.086
800	1-> 0.088
1600	1-> 0.089
3200	1-> 0.092
6400	1-> 0.083

We can observe that Doubling the N increases by nLogn, where n = N^2.

- This is because when we create a Percolation, the constructor takes O(n). Since the dimension is N^2. It will take N^2 to create a new Percolation since the WeightedQuickUnionUF constructor takes O(n).
- When we call the open() function, it takes O(logn), where n = N^2. This is because the Union and Connected function takes O(logn)

We can observe that by doubling T, it will increase by N.

- This can be explained as in PercolationStats, our T is Linear (Only 1 for loop).

Thus, a formula for the running time can be:

~ T*N^2 * log(N^2) * 10^(-8) s

PS: The 10^(-8) came after testing the formula above, we observed that it was 10^8 times bigger.
Example: IF N = 400 and T = 100 => Running Time = 2.76 * 10^8, Real running time =

2.46 s.

```
==========================
2 - Using the 64-bit memory-cost model from lecture and Section 1.4, give the total memory usage in bytes (using tilde notation) that a Percolation object uses to model an N-by-N percolation system. Count all memory that is used, including memory for the union-find data structure.
==========================
```

In my Percolation,

- We have 4 private int => 4 * 4 = 16
- We have 1 boolean[][] (Grid) => 1 * N^2 = N^2
- We have 2 WeighthedQuickUnionUF (WQU)
    - A WQU is an int with size n
    - Thus, 4 * (N*N + 2) + 4*(N*N) + 1 ~= 8N^3
- Since is an Object, we need to add 16 bytes and have the memory be padded to be a multiple of 8 bytes.

Answer:

~ 8N^3 + N^2 + 16 + 16 = ~8N^3 + N^2 + 32 + X, where X is the number of bytes needed for the padding.

```
/*******************************************************************************
*  Describe how you implemented Percolation.java. How did you check whether the system percolates?
******************************************************************************/
```

How Percolation.java was implemented:

- The idea is to have each site in the percolation to have a number from 1 to N and use the Union-Find Data Structure to group them by sets.
- Everytime there is an open site, we check if the adjacent sites are open or not. If they are open, we union them together.
- In order to have a constant time to check if the Percolation percolates, we created a virtual source with site number 0 and a virtual sink with site number N + 1.
    - The source will be connected to all the sites from the first row that are open.
    - The sink will be connected to all the sites from the last row that are open.
    - It percolates if the Virtual source and Virtual sink are connected.

```
/*******************************************************************************
*  Perform computational experiments to estimate the running time of PercolationStats.java for values of n and T when implementing Percolation.java with WeightedQuickFindUF.java.
*
*  To do so, fill in the two tables below. Each table must have at least 4 data points, ranging in time from around 0.1 seconds to around 60 seconds. Do not include data points that takes less than 0.1 seconds.
```

```
******************************************************************************
/ (keep T constant)  T = 100
n          time (seconds)
------------------------------
100         0.297
150         0.447
200         0.66
300         1.302
400         2.46

(keep n constant)   n = 100
T          time (seconds)
------------------------------
100         0.297
150         0.353
200         0.401
300         0.497
400         0.615

/*******************************************************************************
*  Using the empirical data from the above two tables, give a formula (using tilde
notation) for the running time (in seconds) of PercolationStats.java as function of
both n and T, such as
*       ~ 5.3*10^-8 * n^5.0 T^1.5
*  With the tilde notation, you include both the coefficient and exponents of the
leading term (but not lower-order terms). Round each coefficient to two significant
digits.
******************************************************************************/
~ T*N^2 * log(N^2) * 10^(-8) s

The explanation for this answer is above in question 1.

------------------------------------------------------------------------------------
---------
Correctness Tests:

==========================
1
==========================
All program passed

    a - pass
    b - pass
    c - pass
    d - pass
    e - pass
    f - pass

==========================
2
```

```
==========================
All program passed

    a - pass
    b - pass
    c - pass
    d - pass

==========================
3
==========================
All program passed

    a - pass
    b - pass

==========================
4
==========================
    a - pass

==========================
5
==========================
All program passed

    PercolationStats(200, 100) - pass
        mean = 0.5940162499999998
        stddev = 0.010243517520827262
        95% confidence interval = [0.5920085205659177, 0.5960239794340819]

    PercolationStats(200, 100) - pass
        mean = 0.5904457499999998
        stddev = 0.010157205641908617
        95% confidence interval = [0.5884549376941857, 0.5924365623058139]

    PercolationStats(2, 100000) - pass
        mean = 0.666435
        stddev = 0.11793337041650533
        95% confidence interval = [0.6657040413971933, 0.6671659586028067]

    Threshold = 0.5926
        (obtained by running N = 1000 T = 1000)
        (Also matches the value from the percolation probability on the assignment)

==========================
6
==========================
All program passed
        a
```

```
                n = 3 : Percolated after opening 7 random sites
        b
                n = 5 : Percolated after opening 17 random sites
        c
                n = 10 : Percolated after opening 61 random sites
        d
                n = 20 : Percolated after opening 265 random sites
        e
                n = 50 : Percolated after opening 1490 random sites
        f
                n = 250 : Percolated after opening 36970 random sites
        g
                n = 500 : Percolated after opening 145013 random sites
        h
                n = 1000 : Percolated after opening 594151 random sites
        g
                n = 2000 : Percolated after opening 2378404 random sites

==========================
7
==========================
All program passed
    a
        -1,5
        open() causes an exception: IndexOutOfBoundsException.
    b
        11,5
        open() causes an exception: IndexOutOfBoundsException.
    c
        0,5
        open() causes an exception: IndexOutOfBoundsException.
    d
        5,-1
        open() causes an exception: IndexOutOfBoundsException.
    e
        -2147483648,-2147483648
        open() causes an exception: IndexOutOfBoundsException.
    f
        2147483647,2147483647
        open() causes an exception: IndexOutOfBoundsException.

==========================
 8
==========================
All program passed
        a
                Constructor invalid argument:-10
        b
                Constructor invalid argument:-1
        c
```

```
Constructor invalid argument:0
```