Andrew Walters

CSC 2053

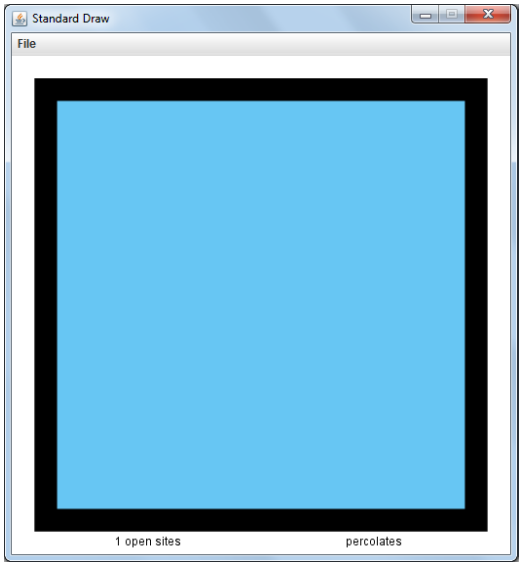
Percolation Project

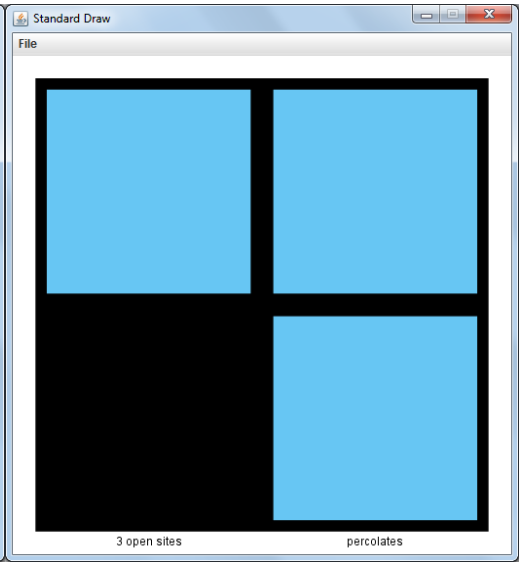
The java classes Percolation and PercolationStats were implemented successfully and testing shows that they behave as expected. The Percolation visualizer demonstrates that the backwash problem was successfully handled in the Percolation class, and the results from PercolationStats shows are sufficiently close to the expected results.

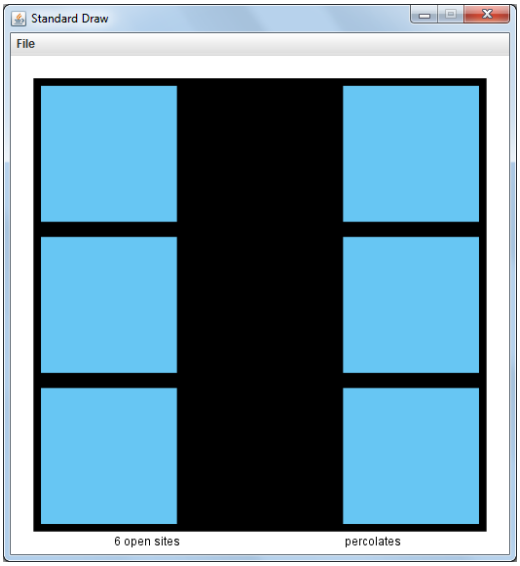
## `public class Percolation`

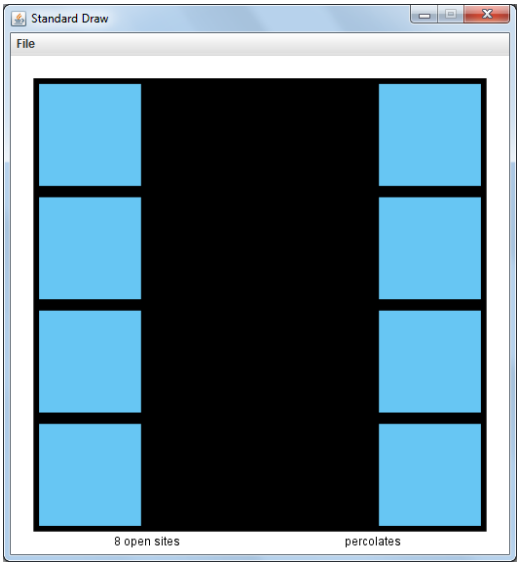
The complete listing of code for this class can be found in the appendix. The following page of figures is the result of testing the Percolation class by running the PercolationVisualizer. The following commands were run and the figures on the next two pages were produced in order.

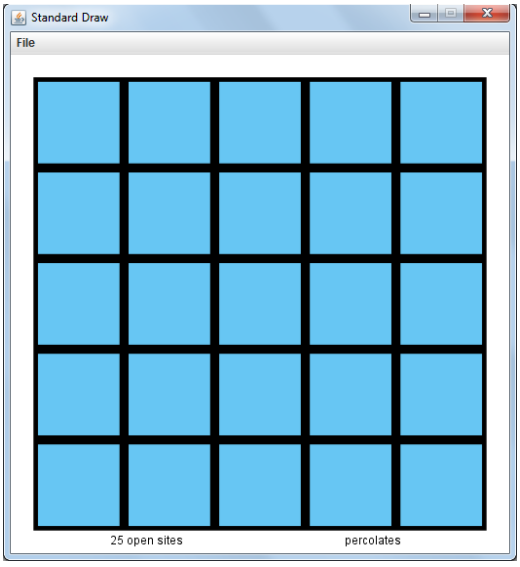
```
>java PercolationVisualizer input1.txt
>java PercolationVisualizer input2.txt
>java PercolationVisualizer input3.txt
>java PercolationVisualizer input4.txt
>java PercolationVisualizer input5.txt
>java PercolationVisualizer input6.txt
>java PercolationVisualizer input7.txt
>java PercolationVisualizer input8.txt
>java PercolationVisualizer input10.txt
>java PercolationVisualizer input20.txt
>java PercolationVisualizer input50.txt
>java PercolationVisualizer greeting57.txt
```

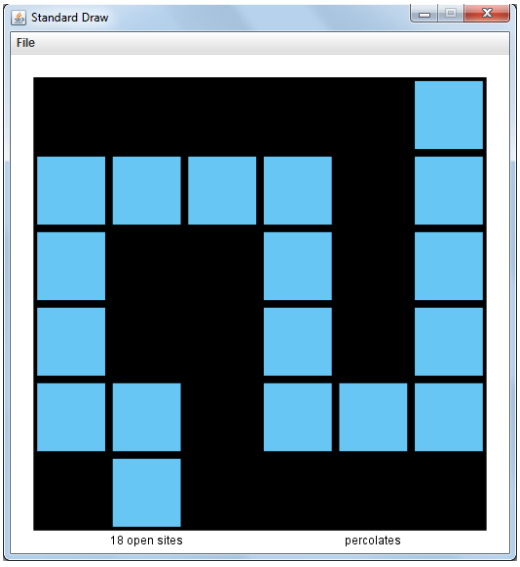

Standard Draw
File
1 open sites
percolates

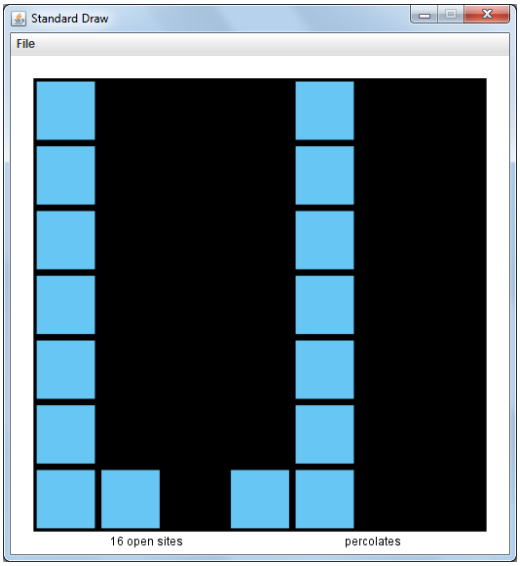

Standard Draw
File
3 open sites
percolates

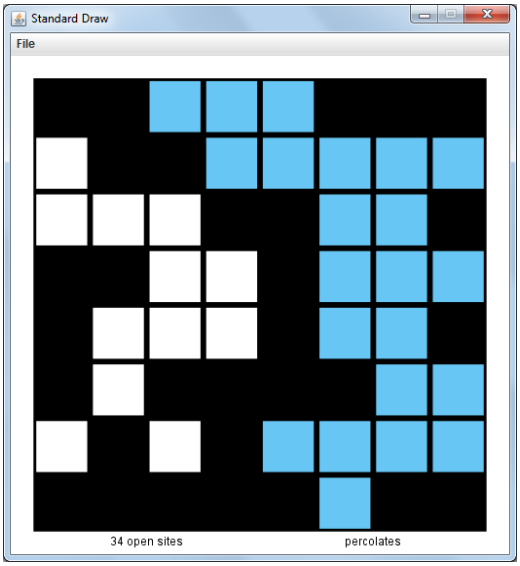

Standard Draw
File
6 open sites
percolates

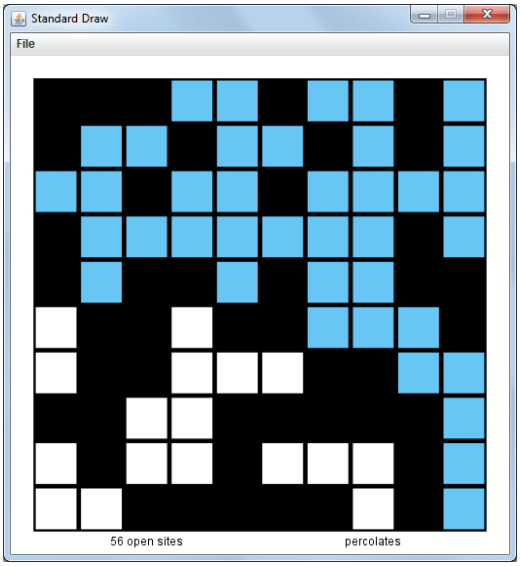

Standard Draw
File
8 open sites
percolates

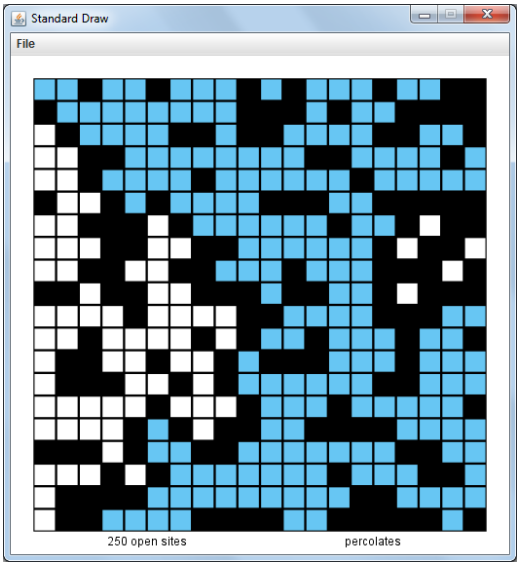

Standard Draw
File
25 open sites
percolates

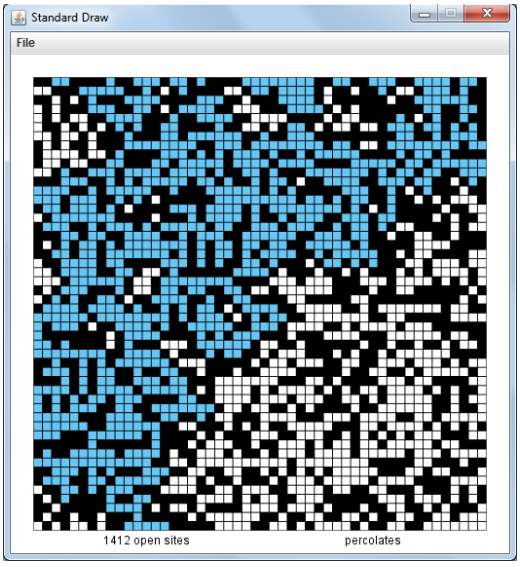

Standard Draw
File
18 open sites
percolates

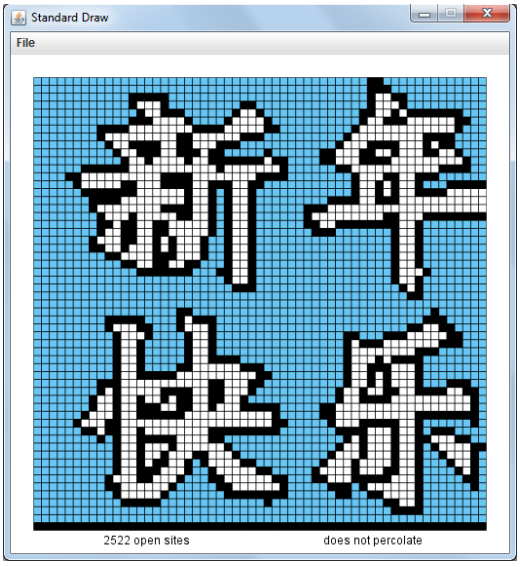

Standard Draw
File
16 open sites
percolates


Standard Draw
File
34 open sites
percolates


Standard Draw
File
56 open sites
percolates


Standard Draw
File
250 open sites
percolates


Standard Draw
File
1412 open sites
percolates


Standard Draw
File
2522 open sites
does not percolate

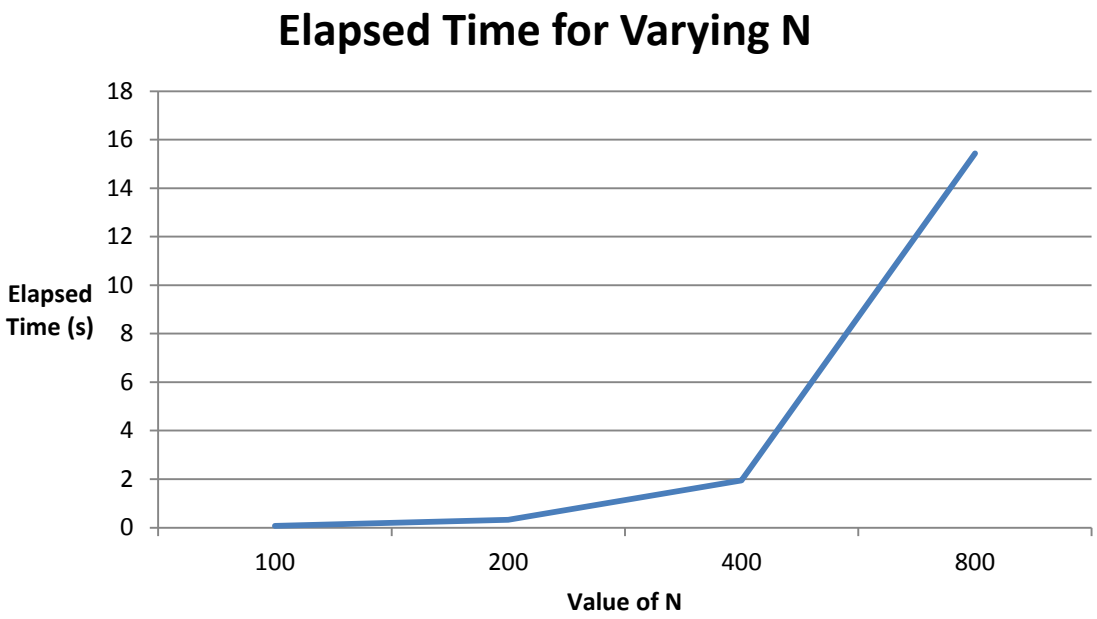
**public class PercolationStats**

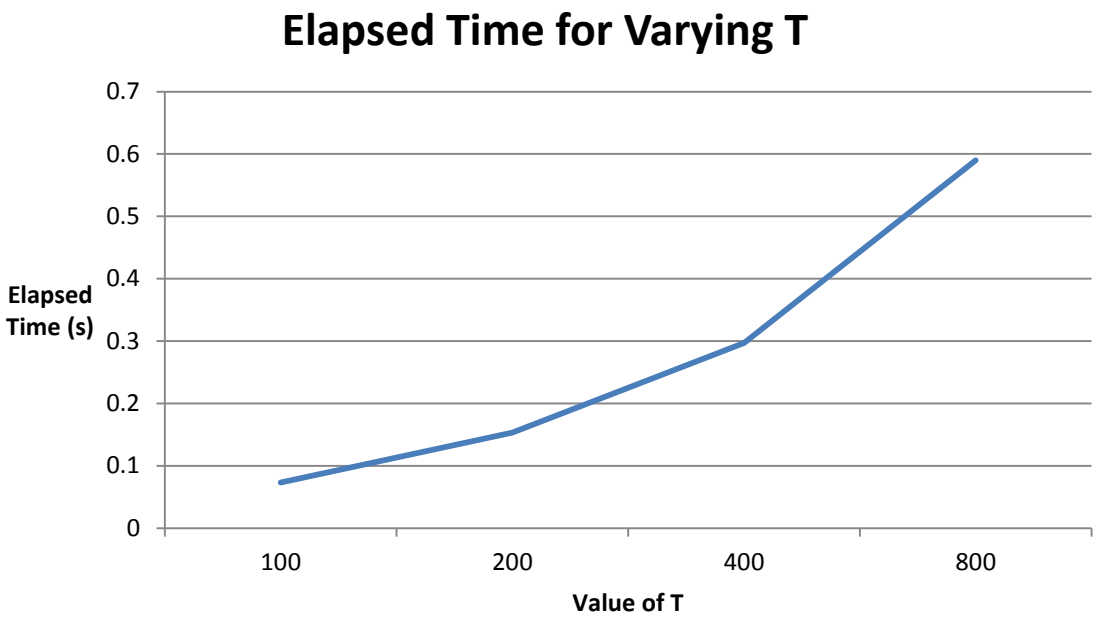
The complete listing of code for this class can be found in the appendix. PercolationStats was executed with a wide range of input parameters. The expected value for the mean was 0.593 which was provided by the instructor, and all runs of more than ten experiments on grids larger than 10 produced a 95% confidence interval containing the value 0.593.

```
> java PercolationStats 100 100
mean:                    0.5917209999999999
stddev:                  0.017504344597784292
95% confidence interval: 0.5882901484588342, 0.5951518515411657
elapsed time:            0.078s

> java PercolationStats 100 1000
mean:                    0.5920285999999999
stddev:                  0.016250363576128277
95% confidence interval: 0.5910213920305544, 0.5930358079694454
elapsed time:            0.732s

> java PercolationStats 100 10000
mean:                    0.5928147299999984
stddev:                  0.01632101153632497
95% confidence interval: 0.5924948381738865, 0.5931346218261103
elapsed time:            73.192s
```

After the percolation threshold was successfully found, a timing analysis for varying values of N and T was performed. It was found that increasing N results in a increase in computing time proportional to $N^2$, whereas increasing T increases computing time proportionally to T. This is expected because T corresponds to an array of length T while N corresponds to an array of length N-by-N. Below you can see the table and graphs for the varying value of N and T. In all cases, the non-varying parameter was kept at 100.

| N | Elapsed Time (s) | T | Elapsed Time (s) |
|---|---|---|---|
| **100** | 0.073 | **100** | 0.073 |
| **200** | 0.323 | **200** | 0.153 |
| **400** | 1.945 | **400** | 0.297 |
| **800** | 15.438 | **800** | 0.59 |


Elapsed Time for Varying N
18
16
14
12
10
8
6
4
2
0
Elapsed Time (s)
100
200
400
800
Value of N


Elapsed Time for Varying T
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Elapsed Time (s)
100
200
400
800
Value of T

```
1   /*
2    *  Compilation:  javac Percolation.java
3    *  Execution:    This class does not contain a main method
4    *  Dependancies: WeightedQuickUnionUF.java
5    *
6    *  This class creates an N by N grid of closed sites
7    *  that can be opened by the function open(row,col).
8    *  The function isFull(row,col) will determine whether
9    *  water has percolated to that site and percolates() will
10   *  determine if water has percolated to through the grid.
11   *
12   *  grid indicies begin at 1 instead of zero for all function in this class
13   */
14
15
16  public class Percolation {
17      //constants
18      private final int CLOSED = 0;
19      private final int OPEN = 1;
20      //variables
21      private int size;
22      private int grid[][];
23      private WeightedQuickUnionUF perc;
24      private WeightedQuickUnionUF full;
25
26      // create N-by-N grid, with all sites blocked
27      public Percolation(int N) {
28
29          perc = new WeightedQuickUnionUF(N*N+2);
30          full = new WeightedQuickUnionUF(N*N+1);
31
32          grid = new int[N][N];
33          for(int i = 0; i < N; i++){
34              for(int j = 0; j < N; j++){
35                  grid[i][j] = CLOSED;
36              }//end for
37          }//end for
38
39          size = N;
40
41      }//end constructor
42
43      // open site (row i, column j) if it is not already
44      public void open(int i, int j) {
45          if(isOpen(i,j)) {
46              return;
47          }
48          else {
49              grid[(i-1)][(j-1)] = OPEN;
50
51              //join left
52              if(j!=1)
53                  con(i,j,i,(j-1));
54
55              //join right
56              if(j!=size)
57                  con(i,j,i,(j+1));
58
59              //join above
60              if(i!=1) {
61                  con(i,j,(i-1),j);
62              }
63              else {
64                  perc.union(g2p(i,j),0);
65                  full.union(g2p(i,j),0);
66              }
67
68              //join below
69              if(i!=size)
70                  con(i,j,(i+1),j);
```

```
71              else
72                  perc.union(g2p(i,j),(size*size+1));
73
74          }//end else
75
76
77      }//end open
78
79      // is site (row i, column j) open?
80      public boolean isOpen(int i, int j) {
81          if(grid[(i-1)][(j-1)] == OPEN)
82              return true;
83          else
84              return false;
85      }//end isOpen
86
87      // is site (row i, column j) full?
88      public boolean isFull(int i, int j) {
89          return full.connected(0,g2p(i,j));
90      }//end isFull
91
92      // does the system percolate?
93      public boolean percolates() {
94          return perc.connected(0,size*size+1);
95      }//end percolates
96
97      //converts grid coordinates to index of UF struct
98      private int g2p(int i, int j){
99          return (i-1)*size + (j-1) + 1;
100     }//end mod
101
102     //union two sites
103     private void con(int pi, int pj, int qi, int qj) {
104         //ensure both sites are open
105         if( isOpen(pi,pj) && isOpen(qi,qj) ) {
106             perc.union(g2p(pi,pj),g2p(qi,qj));
107             full.union(g2p(pi,pj),g2p(qi,qj));
108         }//end if
109     }//end con
110 }//end class
```

```
1   /*
2    *  Compilation:  javac PercolationStats.java
3    *  Execution:    java PercolationStats N T
4    *  Dependancies: Percolation.java, Stopwatch.java,
5    *                StdRandom.java, Math.java
6    *
7    *  When executed, this class runs T expiriments on N-by-N grids
8    *  of class percolation. Sites in the grid are opened at random
9    *  until the system percolates. The fraction of open sites for each
10   *  expiriment is recored and the mean and standard deviation
11   *  for all experiments is found and printed.
12   */
13
14  public class PercolationStats {
15
16      // variables
17      private double thresh[];
18      private double av, sd;
19
20      // perform T independent computational experiments on an N-by-N grid
21      public PercolationStats(int N, int T) {
22          thresh = new double[T];
23          for(int i = 0; i < T; i++) {
24              thresh[i] = perc(N);
25          }//end for
26      }//end PercolationStats
27
28      // perform a percolation of N by N grid and return fraction of open sites
29      private double perc(int N) {
30          Percolation p = new Percolation(N);
31          int i, j;
32          int count = 0;
33          while(!p.percolates()) {
34              i = StdRandom.uniform(N) + 1;
35              j = StdRandom.uniform(N) + 1;
36              if(!p.isOpen(i,j)){
37                  p.open(i,j);
38                  count++;
39              }//end if
40          }//end while
41          return (double)count/(double)(N*N);
42      }//end perc
43
44      // sample mean of percolation threshold
45      public double mean() {
46          double count = 0;
47          for(int i = 0; i < thresh.length; i++) {
48              count+=thresh[i];
49          }//end for
50          av = count/thresh.length;
51          return av;
52      }//end mean
53
54      // sample standard deviation of percolation threshold
55      public double stddev() {
56          double diff = 0;
57          for(int i = 0; i < thresh.length; i++) {
58              diff += Math.pow((thresh[i] - av),2);
59          }//end for
60          sd = Math.sqrt(diff / (thresh.length-1));
61          return sd;
62      }//end stdev
63
64      // returns lower bound of the 95% confidence interval
65      public double confidenceLo() {
66          return av - ( (1.96 * sd) / Math.sqrt(thresh.length));
67      }//end confidenceLo
68
69      // returns upper bound of the 95% confidence interval
70      public double confidenceHi() {
```

```
71          return av + ( (1.96 * sd) / Math.sqrt(thresh.length));
72      }//end confidenceHi
73
74      // test client, described below
75      public static void main(String[] args) {
76          int N = Integer.parseInt(args[0]);
77          int T = Integer.parseInt(args[1]);
78          Stopwatch clock = new Stopwatch();
79          PercolationStats ps = new PercolationStats(N,T);
80          double time = clock.elapsedTime();
81
82          System.out.println("mean:\t\t\t\t" + ps.mean());
83          System.out.println("stddev:\t\t\t\t" + ps.stddev());
84          System.out.println("95% confidence interval:\t\t" + ps.confidenceLo() + ",
    " + ps.confidenceHi());
85          System.out.println("elapsed time:\t\t\t" + time + "s");
86      }//end main
87
88  }//end class
```