```

ASSESSMENT SUMMARY

Compilation:  PASSED
API:          PASSED

Spotbugs:     PASSED
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 23 warnings)

Correctness:  31/33 tests passed
Memory:       8/8 tests passed
Timing:       20/20 tests passed

Aggregate score: 96.36%
[Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:
----------------------------------
3.2K Apr 14 15:40 Percolation.java
2.1K Apr 14 15:40 PercolationStats.java
  50 Apr 14 15:40 grupo-aed.txt


********************************************************************************
*  COMPILING
********************************************************************************


% javac Percolation.java
*-----------------------------------------------------------

% javac PercolationStats.java
*-----------------------------------------------------------


================================================================


Checking the APIs of your programs.
*-----------------------------------------------------------
Percolation:

PercolationStats:

================================================================


********************************************************************************
*  CHECKING STYLE AND COMMON BUG PATTERNS
********************************************************************************


% spotbugs *.class
*-----------------------------------------------------------


================================================================


% pmd .
*-----------------------------------------------------------


================================================================


% checkstyle *.java
*-----------------------------------------------------------
[WARN] Percolation.java:1:3: '//' or '/*' is not followed by whitespace.
```

```
[WhitespaceAfter]
[WARN] Percolation.java:2:3: '//' or '/*' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Percolation.java:3:3: '//' or '/*' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Percolation.java:8:1: File contains tab characters (this is the first
occurrence). Configure your editor to replace tabs with spaces. [FileTabCharacter]
[WARN] Percolation.java:8:11: The modifier 'private' is out of order. The preferred
order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] Percolation.java:9:11: The modifier 'private' is out of order. The preferred
order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] Percolation.java:12:11: The modifier 'private' is out of order. The preferred
order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] Percolation.java:40:47: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
[WARN] Percolation.java:46:42: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
[WARN] Percolation.java:52:57: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
[WARN] Percolation.java:58:65: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
[WARN] PercolationStats.java:1:3: '//' or '/*' is not followed by whitespace.
[WhitespaceAfter]
[WARN] PercolationStats.java:2:3: '//' or '/*' is not followed by whitespace.
[WhitespaceAfter]
[WARN] PercolationStats.java:3:3: '//' or '/*' is not followed by whitespace.
[WhitespaceAfter]
[WARN] PercolationStats.java:10:1: File contains tab characters (this is the first
occurrence). Configure your editor to replace tabs with spaces. [FileTabCharacter]
[WARN] PercolationStats.java:10:11: The modifier 'private' is out of order. The
preferred order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] PercolationStats.java:11:11: The modifier 'private' is out of order. The
preferred order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] PercolationStats.java:12:11: The modifier 'private' is out of order. The
preferred order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] PercolationStats.java:13:11: The modifier 'private' is out of order. The
preferred order is ['public', 'protected', 'private', 'abstract', 'static', 'final',
'transient', 'volatile', 'synchronized', 'native', and 'strictfp']. [ModifierOrder]
[WARN] PercolationStats.java:19:27: To specify an array type, put the square
brackets before the variable name, e.g., 'String[] args' instead of 'String args[]'.
[ArrayTypeStyle]
Checkstyle ends with 0 errors and 20 warnings.

% custom checkstyle checks for Percolation.java
*-----------------------------------------------------------
[WARN] Percolation.java:1: We recommend defining at least one private helper method,
e.g., to validate the row and column indices or to map from 2D to 1D indices. [Design]
Checkstyle ends with 0 errors and 1 warning.

% custom checkstyle checks for PercolationStats.java
*-----------------------------------------------------------
[WARN] PercolationStats.java:5: The number (0) of calls to 'Integer.parseInt()' must
equal the number (2) of integer command-line arguments. [CommandLineArgument]
[WARN] PercolationStats.java:5:1: The constant '1.96' appears more than once. Define
a constant variable (such as 'CONFIDENCE_95') to hold the constant '1.96'.
[NumericLiteralCount]
Checkstyle ends with 0 errors and 2 warnings.


================================================================


******************************************************************************
*  TESTING CORRECTNESS
******************************************************************************

Testing correctness of Percolation
```

```
*-----------------------------------------------------------
Running 18 total tests.

Tests 1 through 8 create a Percolation object using your code, then repeatedly
open sites by calling open(). After each call to open(), it checks the return
values of isOpen(), percolates(), numberOfOpenSites(), and isFull() in that order.
Tests 11 through 14 create a Percolation object using your code, then repeatedly
call the methods open(), isOpen(), isFull(), percolates(), and, numberOfOpenSites()
in random order with probabilities p = (p1, p2, p3, p4, p5). The tests stop
immediately after the system percolates.

Tests 16 through 18 test backwash.

Except as noted, a site is opened at most once.

Test 1: open predetermined list of sites using file inputs
  * filename = input6.txt
  * filename = input8.txt
  * filename = input8-no.txt
  * filename = input10-no.txt
  * filename = greeting57.txt
  * filename = heart25.txt
==> passed

Test 2: open random sites until just before system percolates
  * n = 3
  * n = 5
  * n = 10
  * n = 10
  * n = 20
  * n = 20
  * n = 50
  * n = 50
==> passed

Test 3: open predetermined sites for n = 1 and n = 2 (corner case test)
  * filename = input1.txt
  * filename = input1-no.txt
  * filename = input2.txt
  * filename = input2-no.txt
==> passed

Test 4: check predetermined sites with long percolating path
  * filename = snake13.txt
  * filename = snake101.txt
==> passed

Test 5: open every site
  * filename = input5.txt
==> passed

Test 6: open random sites until just before system percolates,
        allowing open() to be called on a site more than once
  * n = 3
  * n = 5
  * n = 10
  * n = 10
  * n = 20
  * n = 20
  * n = 50
  * n = 50
==> passed

Test 7: call methods with invalid arguments
  * n = 10, (row, col) = (-1, 5)
  * n = 10, (row, col) = (11, 5)
  * n = 10, (row, col) = (0, 5)
  * n = 10, (row, col) = (5, -1)
  * n = 10, (row, col) = (5, 11)
  * n = 10, (row, col) = (5, 0)
  * n = 10, (row, col) = (-2147483648, -2147483648)
  * n = 10, (row, col) = (2147483647, 2147483647)
==> passed
```

```

Test 8: call constructor with invalid argument
  * n = -10
  * n = -1
  * n = 0
==> passed

Test 9: create multiple Percolation objects at the same time
        (to make sure you didn't store data in static variables)
==> passed

Test 10: open predetermined list of sites using file inputs,
         but permute the order in which methods are called
  * filename = input8.txt;  order =     isFull(),     isOpen(), percolates()
  * filename = input8.txt;  order =     isFull(), percolates(),     isOpen()
  * filename = input8.txt;  order =     isOpen(),     isFull(), percolates()
  * filename = input8.txt;  order =     isOpen(), percolates(),     isFull()
  * filename = input8.txt;  order = percolates(),     isOpen(),     isFull()
  * filename = input8.txt;  order = percolates(),     isFull(),     isOpen()
==> passed

Test 11: call open(), isOpen(), and numberOfOpenSites()
         in random order until system percolates
  * n = 3, trials = 40, p = (0.4, 0.4, 0.0, 0.0, 0.3)
  * n = 5, trials = 20, p = (0.4, 0.4, 0.0, 0.0, 0.3)
  * n = 7, trials = 10, p = (0.4, 0.4, 0.0, 0.0, 0.3)
  * n = 10, trials = 5, p = (0.4, 0.4, 0.0, 0.0, 0.3)
  * n = 20, trials = 2, p = (0.4, 0.4, 0.0, 0.0, 0.3)
  * n = 50, trials = 1, p = (0.4, 0.4, 0.0, 0.0, 0.3)
==> passed

Test 12: call open() and percolates() in random order until system percolates
  * n = 3, trials = 40, p = (0.5, 0.0, 0.0, 0.5, 0.0)
  * n = 5, trials = 20, p = (0.5, 0.0, 0.0, 0.5, 0.0)
  * n = 7, trials = 10, p = (0.5, 0.0, 0.0, 0.5, 0.0)
  * n = 10, trials = 5, p = (0.5, 0.0, 0.0, 0.5, 0.0)
  * n = 20, trials = 2, p = (0.5, 0.0, 0.0, 0.5, 0.0)
  * n = 50, trials = 1, p = (0.5, 0.0, 0.0, 0.5, 0.0)
==> passed

Test 13: call open() and isFull() in random order until system percolates
  * n = 3, trials = 40, p = (0.5, 0.0, 0.5, 0.0, 0.0)
  * n = 5, trials = 20, p = (0.5, 0.0, 0.5, 0.0, 0.0)
  * n = 7, trials = 10, p = (0.5, 0.0, 0.5, 0.0, 0.0)
  * n = 10, trials = 5, p = (0.5, 0.0, 0.5, 0.0, 0.0)
  * n = 20, trials = 2, p = (0.5, 0.0, 0.5, 0.0, 0.0)
  * n = 50, trials = 1, p = (0.5, 0.0, 0.5, 0.0, 0.0)
==> passed

Test 14: call all methods in random order until system percolates
  * n = 3, trials = 40, p = (0.2, 0.2, 0.2, 0.2, 0.2)
  * n = 5, trials = 20, p = (0.2, 0.2, 0.2, 0.2, 0.2)
  * n = 7, trials = 10, p = (0.2, 0.2, 0.2, 0.2, 0.2)
  * n = 10, trials = 5, p = (0.2, 0.2, 0.2, 0.2, 0.2)
  * n = 20, trials = 2, p = (0.2, 0.2, 0.2, 0.2, 0.2)
  * n = 50, trials = 1, p = (0.2, 0.2, 0.2, 0.2, 0.2)
==> passed

Test 15: call all methods in random order until almost all sites are open,
         but with inputs not prone to backwash
  * n = 3
  * n = 5
  * n = 7
  * n = 10
  * n = 20
  * n = 50
==> passed

Test 16: check for backwash with predetermined sites
  * filename = input20.txt
  * filename = input10.txt
  * filename = input50.txt
  * filename = jerry47.txt
  * filename = sedgewick60.txt
  * filename = wayne98.txt


        ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        OperationCountLimitExceededException
        Number of calls to methods in WeightedQuickUnionUF exceeds limit: 250000000
        ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

==> FAILED

Test 17: check for backwash with predetermined sites that have
         multiple percolating paths
  * filename = input3.txt
  * filename = input4.txt
  * filename = input7.txt
==> passed

Test 18: call all methods in random order until all sites are open,
         allowing isOpen() to be called on a site more than once
         (these inputs are prone to backwash)
  * n = 3
  * n = 5
  * n = 7
  * n = 10
  * n = 20
  * n = 50
==> passed


Total: 17/18 tests passed!


================================================================
********************************************************************************
*  TESTING CORRECTNESS (substituting reference Percolation)
********************************************************************************

Testing correctness of PercolationStats
*-----------------------------------------------------------
Running 15 total tests.

Test 1: check that methods in PercolationStats do not print to standard output
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 2: check that mean() returns value in expected range
  * n =   2, trials = 10000
  * n =   5, trials = 10000
  * n =  10, trials = 10000
  * n =  25, trials = 10000
==> passed

Test 3: check that stddev() returns value in expected range
  * n =   2, trials = 10000
  * n =   5, trials = 10000
  * n =  10, trials = 10000
  * n =  25, trials = 10000
==> passed

Test 4: check that PercolationStats creates trials Percolation objects, each of size
n-by-n
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 5: check that PercolationStats calls open() until system percolates
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 6: check that PercolationStats does not call open() after system percolates
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 7: check that mean() is consistent with the number of intercepted calls to open()
        on blocked sites
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 8: check that stddev() is consistent with the number of intercepted calls to open()
        on blocked sites
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 9: check that confidenceLo() and confidenceHigh() are consistent with mean() and stddev()
  * n =  20, trials =  10
    - PercolationStats confidence low  = -0.007806421393002361
    - PercolationStats confidence high = 1.2318064213930024
    - PercolationStats mean            = 0.612
    - PercolationStats stddev          = 0.045625285387235315
    - T                                = 10
    - student T                        = 10
    - mean - 1.96 * stddev / sqrt(T)   = 0.5837211551391033
    - mean + 1.96 * stddev / sqrt(T)   = 0.6402788448608967

  * n =  50, trials =  20
    - PercolationStats confidence low  = 0.15551067641004124
    - PercolationStats confidence high = 1.0320493235899586
    - PercolationStats mean            = 0.59378
    - PercolationStats stddev          = 0.019869246270767818
    - T                                = 20
    - student T                        = 20
    - mean - 1.96 * stddev / sqrt(T)   = 0.5850719188766682
    - mean + 1.96 * stddev / sqrt(T)   = 0.6024880811233317

  * n = 100, trials =  50
    - PercolationStats confidence low  = 0.3185021417748734
    - PercolationStats confidence high = 0.8728738582251265
    - PercolationStats mean            = 0.595688
    - PercolationStats stddev          = 0.01693840751014073
    - T                                = 50
    - student T                        = 50
    - mean - 1.96 * stddev / sqrt(T)   = 0.5909929129773347
    - mean + 1.96 * stddev / sqrt(T)   = 0.6003830870226653

  * n =  64, trials = 150
    - PercolationStats confidence low  = 0.4314184930548324
    - PercolationStats confidence high = 0.7514851527785009
    - PercolationStats mean            = 0.5914518229166666
    - PercolationStats stddev          = 0.02079900623081705
    - T                                = 150
    - student T                        = 150
    - mean - 1.96 * stddev / sqrt(T)   = 0.5881232886917319
    - mean + 1.96 * stddev / sqrt(T)   = 0.5947803571416014

==> FAILED
```

```
Test 10: check that exception is thrown if either n or trials is out of bounds
  * n = -23, trials =  42
  * n =  23, trials =   0
  * n = -42, trials =   0
  * n =  42, trials =  -1
  * n = -2147483648, trials = -2147483648
==> passed

Test 11: create two PercolationStats objects at the same time and check mean()
         (to make sure you didn't store data in static variables)
  * n1 =  50, trials1 =  10, n2 =  50, trials2 =   5
  * n1 =  50, trials1 =   5, n2 =  50, trials2 =  10
  * n1 =  50, trials1 =  10, n2 =  25, trials2 =  10
  * n1 =  25, trials1 =  10, n2 =  50, trials2 =  10
  * n1 =  50, trials1 =  10, n2 =  15, trials2 = 100
  * n1 =  15, trials1 = 100, n2 =  50, trials2 =  10
==> passed

Test 12: check that the methods return the same value, regardless of
         the order in which they are called
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 13: check that no calls to StdRandom.setSeed()
  * n = 20, trials = 10
  * n = 20, trials = 10
  * n = 40, trials = 10
  * n = 80, trials = 10
==> passed

Test 14: check distribution of number of sites opened until percolation
  * n = 2, trials = 100000
  * n = 3, trials = 100000
  * n = 4, trials = 100000
==> passed

Test 15: check that each site is opened the expected number of times
  * n = 2, trials = 100000
  * n = 3, trials = 100000
  * n = 4, trials = 100000
==> passed


Total: 14/15 tests passed!


================================================================
********************************************************************************
*  MEMORY (substituting reference Percolation)
********************************************************************************

Analyzing memory of PercolationStats
*-----------------------------------------------------------
Running 4 total tests.

Test 1a-1d: check memory usage as a function of T trials for n = 100
            (max allowed: 8*T + 128 bytes)

                 T        bytes
--------------------------------------------
=> passed       16           48
=> passed       32           48
=> passed       64           48
=> passed      128           48
==> 4/4 tests passed


Estimated student memory = 48.00   (R^2 = 1.000)

Total: 4/4 tests passed!
```

```


================================================================



********************************************************************************
*  TIMING (substituting reference Percolation)
********************************************************************************

Timing PercolationStats
*-----------------------------------------------------------
Running 4 total tests.

Test 1: count calls to StdStats.mean() and StdStats.stddev()
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 2: count calls to methods in StdRandom
  * n = 20, trials = 10
  * n = 20, trials = 10
  * n = 40, trials = 10
  * n = 80, trials = 10
==> passed

Test 3: count calls to methods in Percolation
  * n =  20, trials =  10
  * n =  50, trials =  20
  * n = 100, trials =  50
  * n =  64, trials = 150
==> passed

Test 4: Call PercolationStats methods with trials = 3 and values of n that go up
        by a factor of sqrt(2). The test passes when n reaches 2,896.

     The approximate order-of-growth is n ^ (log ratio)

         n  seconds log ratio
     ------------------------
       512     0.10       2.2
       724     0.26       2.6
      1024     0.63       2.5
      1448     1.46       2.4
      2048     3.32       2.4
      2896     7.27       2.3
==> passed


Total: 4/4 tests passed!


================================================================



********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Percolation
*-----------------------------------------------------------
Running 4 total tests.

Test 1a-1d: check that total memory <= 17 n^2 + 128 n + 1024 bytes

                 n        bytes
--------------------------------------------
=> passed       64        69920
=> passed      256      1114400
=> passed      512      4456736
=> passed     1024     17826080
==> 4/4 tests passed


Estimated student memory = 17.00 n^2 + 0.00 n + 288.00   (R^2 = 1.000)


Test 2 (bonus): check that total memory <= 11 n^2 + 128 n + 1024 bytes
   -  failed memory test for n = 64
==> FAILED


Total: 4/4 tests passed!


================================================================



**********************************************************************************
*  TIMING
**********************************************************************************

Timing Percolation
*-----------------------------------------------------------
Running 16 total tests.

Test 1a-1e: Creates an n-by-n percolation system; open sites at random until
            the system percolates, interleaving calls to percolates() and open().
            Count calls to connected(), union() and find().

                                         2 * connected()
                 n       union()              + find()        constructor
-----------------------------------------------------------------------------------
=> passed       16         406                   318                  2
=> passed       32        1350                  1152                  2
=> passed       64        6044                  4960                  2
=> passed      128       23328                 19582                  2
=> passed      256       95773                 79118                  2
=> passed      512      361245                307734                  2
=> passed     1024     1487377               1248460                  2
==> 7/7 tests passed


If one of the values in the table violates the performance limits
the factor by which you failed the test appears in parentheses.
For example, (9.6x) in the union() column indicates that it uses
9.6x too many calls.


Tests 2a-2f: Check whether the number of calls to union(), connected(), and find()
             is a constant per call to open(), isOpen(), isFull(), and percolates().
             The table shows the maximum number of union() and find() calls made
             during a single call to open(), isOpen(), isFull(), and percolates().
             One call to connected() counts as two calls to find().

                 n     per open()      per isOpen()    per isFull()    per percolates()
----------------------------------------------------------------------------------------------------
=> passed       16       8               0               2               2
=> passed       32       8               0               2               2
=> passed       64       8               0               2               2
=> passed      128       8               0               2               2
=> passed      256       8               0               2               2
=> passed      512       8               0               2               2
=> passed     1024       8               0               2               2
==> 7/7 tests passed



Running time (in seconds) depends on the machine on which the script runs.
```

```

Test 3: Create an n-by-n percolation system; interleave calls to percolates()
        and open() until the system percolates. The values of n go up by a
        factor of sqrt(2). The test is passed if n >= 4096 in under 10 seconds.

     The approximate order-of-growth is n ^ (log ratio)

                         log   union-find     log
         n  seconds   ratio   operations   ratio
     -------------------------------------------
       724     0.13     1.7      2068718     2.0
      1024     0.31     2.6      4194368     2.0
      1448     0.70     2.4      8333532     2.0
      2048     1.25     1.7     16590548     2.0
      2896     2.81     2.3     33349242     2.0
      4096     7.14     2.7     67220450     2.0
==> passed



Test 4: Create an n-by-n percolation system; interleave calls to open(),
        percolates(), isOpen(), isFull(), and numberOfOpenSites() until.
        the system percolates. The values of n go up by a factor of sqrt(2).
        The test is passed if n >= 4096 in under 10 seconds.

                         log   union-find     log
         n  seconds   ratio   operations   ratio
     -------------------------------------------
       724     0.15     2.5      2709302     1.9
      1024     0.33     2.3      5399334     2.0
      1448     0.70     2.1     10847968     2.0
      2048     1.38     2.0     21812142     2.0
      2896     2.93     2.2     43458604     2.0
      4096     6.04     2.1     86769526     2.0
==> passed


Total: 16/16 tests passed!


================================================================
```