See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  26/26 tests passed
Memory:       8/8 tests passed
Timing:       9/9 tests passed

Aggregate score: 100.00% [Correctness: 65%, Memory: 10%,
Timing: 25%, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 12K
-rw-r--r-- 1 2.4K Dec 29 15:47 Percolation.java
-rw-r--r-- 1 2.4K Dec 29 15:47 PercolationStats.java
-rw-r--r-- 1 1.9K Dec 29 15:47 studentSubmission.zip


*********************************************************
**********************
*  compiling
*********************************************************
**********************
```

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

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

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

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



% checkstyle Percolation.java PercolationStats.java
*-----------------------------------------------------------
---
Percolation.java:2:8: Unused import statement for
'edu.princeton.cs.algs4.StdRandom'.
Percolation.java:3:8: Unused import statement for
'edu.princeton.cs.algs4.StdStats'.
Percolation.java:9:1: File contains tab characters (this
is the first instance). Configure your editor to replace
tabs with spaces.
Percolation.java:14:34: Instance variable 'WQUUF' should
start with a lowercase letter and use camelCase.
PercolationStats.java:4:8: Unused import statement for
'edu.princeton.cs.algs4.WeightedQuickUnionUF'.
PercolationStats.java:14:1: File contains tab characters
```

```
(this is the first instance). Configure your editor to
replace tabs with spaces.
PercolationStats.java:26:22: 'while' is not followed by
whitespace.
Checkstyle ends with 7 errors.
================================================================
=======


% findbugs *.class
*----------------------------------------------------------
---
================================================================
=======


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

PercolationStats:

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


*****************************************************************
*********************
*  correctness
*****************************************************************
*********************
```

```
Testing methods in Percolation
*-----------------------------------------------------------
---
Running 15 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(),
we check the return
values of isOpen(i, j) for every (i, j), the return value
of percolates(),
and the return value of isFull(i, j) for every (i, j), in
that order.

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: Opens 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 for backwash with predetermined sites
  *  filename = input20.txt
  *  filename = input10.txt
  *  filename = input50.txt
  *  filename = jerry47.txt
==> passed

Test 5: Check for backwash with predetermined sites that
have
        multiple percolating paths
  *  filename = input3.txt
  *  filename = input4.txt
  *  filename = input7.txt
==> passed

Test 6: Predetermined sites with long percolating path
  *  filename = snake13.txt
```

```
  *  filename = snake101.txt
==> passed

Test 7: Opens every site
  *  filename = input5.txt
==> passed

Test 8: 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 9: Check that IndexOutOfBoundsException is thrown if
(i, j) is out of bounds
  *  N = 10, (i, j) = (0, 6)
  *  N = 10, (i, j) = (12, 6)
  *  N = 10, (i, j) = (11, 6)
  *  N = 10, (i, j) = (6, 0)
  *  N = 10, (i, j) = (6, 12)
  *  N = 10, (i, j) = (6, 11)
==> passed

Test 10: Check that IllegalArgumentException is thrown if
```

```
N <= 0 in constructor
  *  N = -10
  *  N = -1
  *  N = 0
==> passed

Test 11: Create multiple Percolation objects at the same
time
         (to make sure you didn't store data in static
variables)
==> passed

Test 12: Open predetermined list of sites using file
inputs,
         but change 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 13: Call all methods in random order until just
before system percolates
  *  N = 3
```

```
  *  N = 5
  *  N = 7
  *  N = 10
  *  N = 20
  *  N = 50
==> passed

Test 14: 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 15: 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: 15/15 tests passed!

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

*****************************************************************
*********************
*  executing (substituting reference Percolation.java)
*****************************************************************
*********************

Testing methods in PercolationStats
*-----------------------------------------------------------
---
Running 11 total tests.

Test 1: Test that PercolationStats creates T Percolation
objects, each of size N-by-N
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed

Test 2: Test that PercolationStats calls open() until
system percolates
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed
```

```
Test 3: Test that PercolationStats does not call open()
after system percolates
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed

Test 4: Test that mean() is consistent with the number of
intercepted calls to open()
        on blocked sites
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed

Test 5: Test that stddev() is consistent with the number
of intercepted calls to open()
        on blocked sites
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed

Test 6: Test that confidenceLo() and confidenceHigh() are
consistent with mean() and stddev()
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
```

```
==> passed

Test 7: Check whether exception is thrown if either N or
T is out of bounds
  * N = -23, T =  42
  * N =  23, T =   0
  * N = -42, T =   0
  * N =  42, T =  -1
==> passed

Test 8: Create two PercolationStats objects at the same
time and check mean()
        (to make sure you didn't store data in static
variables)
  * N1 =  50, T1 =  10, N2 =  50, T2 =   5
  * N1 =  50, T1 =   5, N2 =  50, T2 =  10
  * N1 =  50, T1 =  10, N2 =  25, T2 =  10
  * N1 =  25, T1 =  10, N2 =  50, T2 =  10
  * N1 =  50, T1 =  10, N2 =  15, T2 = 100
  * N1 =  15, T1 = 100, N2 =  50, T2 =  10
==> passed

Test 9: Check that the methods return the same value,
regardless of
        the order in which they are called
  * N =  20, T =  10
  * N =  50, T =  20
  * N = 100, T =  50
  * N =  64, T = 150
==> passed

Test 10: Check for any calls to StdRandom.setSeed()
```

```
  *  N = 20, T = 10
  *  N = 20, T = 10
  *  N = 40, T = 10
  *  N = 80, T = 10
==> passed

Test 11: Check distribution of number of sites opened
until percolation
  * N = 2, T = 100000
  * N = 3, T = 100000
  * N = 4, T = 100000
==> passed


Total: 11/11 tests passed!

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

*****************************************************************
*********************
*  memory (substituting reference Percolation.java)
*****************************************************************
*********************

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

Test 1a-1d: Memory usage as a function of T for N = 100
            (max allowed: 8 T + 128 bytes)
```

```
                 T        bytes
-------------------------------------------
=> passed       16          208
=> passed       32          336
=> passed       64          592
=> passed      128         1104
==> 4/4 tests passed


Estimated student memory = 8.00 T + 80.00   (R^2 = 1.000)

Total: 4/4 tests passed!

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



*****************************************************************
**********************
*  memory
*****************************************************************
**********************

Computing 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        71984
=> passed      256      1122608
=> passed      512      4473136
=> passed     1024     17858864
==> 4/4 tests passed


Estimated student memory = 17.00 N^2 + 32.00 N + 304.00
(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 9 total tests.

Test 1a-1e: Create an N-by-N percolation system; open
sites at random until
            the system percolates. Count calls to
connected(), union() and
            find() in WeightedQuickUnionUF.
                                                    2 *
connected()
                 N   seconds       union()              +
find()        constructor
-----------------------------------------------------------
-----------------------------------
=> passed        8     0.00          124
250                  2
=> passed       32     0.00         1501
3092                 2
=> passed      128     0.01        22515
48006                 2
=> passed      512     0.10       370380
785726                  2
=> passed     1024     0.31      1457240
3100964                 2
==> 5/5 tests passed

Running time in seconds depends on the machine on which
the script runs,
and  varies each time that you submit. 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-2d: Check whether number of calls to union(),
connected(), and find()
             is a constant per call to open(), isFull(),
and percolates().
             The table shows the maximum number of
union(), connected(), and
             find() calls made during a single call to
open(), isFull(), and
             percolates().

                 N     per open()      per isOpen()
per isFull()    per percolates()
------------------------------------------------------
-----------------------------------
=> passed       32        8               0
1               1
=> passed      128        8               0
1               1
=> passed      512        8               0
1               1
=> passed     1024        8               0
1               1
==> 4/4 tests passed
```

```
Total: 9/9 tests passed!
================================================================
=======
```