

```
1
2 ASSESSMENT SUMMARY
3
4 Compilation: PASSED
5 API: PASSED
6
7 Spotbugs: PASSED
8 PMD: PASSED
9 Checkstyle: FAILED (0 errors, 23 warnings)
10
11 Correctness: 31/33 tests passed
12 Memory: 8/8 tests passed
13 Timing: 20/20 tests passed
14
15 Aggregate score: 96.36%
16 [Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%,
Memory: 10%, Timing: 20%]
17
18 ASSESSMENT DETAILS
19
20 The following files were submitted:
21 -----
22 3.2K Apr 14 15:40 Percolation.java
23 2.1K Apr 14 15:40 PercolationStats.java
24 50 Apr 14 15:40 grupo-aed.txt
25
26
27 *****
28 * COMPILING
29 *****
30
31
32 % javac Percolation.java
33 *-----
34
35 % javac PercolationStats.java
36 *-----
37
38
39 =====
40
41
42 Checking the APIs of your programs.
43 *-----
44 Percolation:
45
46 PercolationStats:
47
48 =====
49
50
51 *****
52 * CHECKING STYLE AND COMMON BUG PATTERNS
53 *****
54
55
56 % spotbugs *.class
57 *-----
58
59
60 =====
61
62
63 % pmd .
64 *-----
65
66
67 =====
68
69
70 % checkstyle *.java
71 *-----
72 [WARN] Percolation.java:1:3: '//' or '/*' is not followed by whitespace.
```

```

[WhitespaceAfter]
73 [WARN] Percolation.java:2:3: '/' or '/' is not followed by whitespace.
[WhitespaceAfter]
74 [WARN] Percolation.java:3:3: '/' or '/' is not followed by whitespace.
[WhitespaceAfter]
75 [WARN] Percolation.java:8:1: File contains tab characters (this is the first
occurrence). Configure your editor to replace tabs with spaces. [FileTabCharacter]
76 [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]
77 [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]
78 [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]
79 [WARN] Percolation.java:40:47: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
80 [WARN] Percolation.java:46:42: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
81 [WARN] Percolation.java:52:57: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
82 [WARN] Percolation.java:58:65: Boolean expression can be simplified, e.g., use 'if
(!isEmpty)' instead of 'if (isEmpty == false)'. [SimplifyBooleanExpression]
83 [WARN] PercolationStats.java:1:3: '/' or '/' is not followed by whitespace.
[WhitespaceAfter]
84 [WARN] PercolationStats.java:2:3: '/' or '/' is not followed by whitespace.
[WhitespaceAfter]
85 [WARN] PercolationStats.java:3:3: '/' or '/' is not followed by whitespace.
[WhitespaceAfter]
86 [WARN] PercolationStats.java:10:1: File contains tab characters (this is the first
occurrence). Configure your editor to replace tabs with spaces. [FileTabCharacter]
87 [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]
88 [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]
89 [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]
90 [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]
91 [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]
92 Checkstyle ends with 0 errors and 20 warnings.
93
94 % custom checkstyle checks for Percolation.java
95 *-----
96 [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]
97 Checkstyle ends with 0 errors and 1 warning.
98
99 % custom checkstyle checks for PercolationStats.java
100 *-----
101 [WARN] PercolationStats.java:5: The number (0) of calls to 'Integer.parseInt()' must
equal the number (2) of integer command-line arguments. [CommandLineArgument]
102 [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]
103 Checkstyle ends with 0 errors and 2 warnings.
104
105
106 =====
107
108
109 *****
110 * TESTING CORRECTNESS
111 *****
112
113 Testing correctness of Percolation

```

```

114  *-----
115  Running 18 total tests.
116
117  Tests 1 through 8 create a Percolation object using your code, then repeatedly
118  open sites by calling open(). After each call to open(), it checks the return
119  values of isOpen(), percolates(), numberOfOpenSites(), and isFull() in that order.
120  Tests 11 through 14 create a Percolation object using your code, then repeatedly
121  call the methods open(), isOpen(), isFull(), percolates(), and, numberOfOpenSites()
122  in random order with probabilities p = (p1, p2, p3, p4, p5). The tests stop
123  immediately after the system percolates.
124
125  Tests 16 through 18 test backwash.
126
127  Except as noted, a site is opened at most once.
128
129  Test 1: open predetermined list of sites using file inputs
130      * filename = input6.txt
131      * filename = input8.txt
132      * filename = input8-no.txt
133      * filename = input10-no.txt
134      * filename = greeting57.txt
135      * filename = heart25.txt
136  ==> passed
137
138  Test 2: open random sites until just before system percolates
139      * n = 3
140      * n = 5
141      * n = 10
142      * n = 10
143      * n = 20
144      * n = 20
145      * n = 50
146      * n = 50
147  ==> passed
148
149  Test 3: open predetermined sites for n = 1 and n = 2 (corner case test)
150      * filename = input1.txt
151      * filename = input1-no.txt
152      * filename = input2.txt
153      * filename = input2-no.txt
154  ==> passed
155
156  Test 4: check predetermined sites with long percolating path
157      * filename = snake13.txt
158      * filename = snake101.txt
159  ==> passed
160
161  Test 5: open every site
162      * filename = input5.txt
163  ==> passed
164
165  Test 6: open random sites until just before system percolates,
166          allowing open() to be called on a site more than once
167      * n = 3
168      * n = 5
169      * n = 10
170      * n = 10
171      * n = 20
172      * n = 20
173      * n = 50
174      * n = 50
175  ==> passed
176
177  Test 7: call methods with invalid arguments
178      * n = 10, (row, col) = (-1, 5)
179      * n = 10, (row, col) = (11, 5)
180      * n = 10, (row, col) = (0, 5)
181      * n = 10, (row, col) = (5, -1)
182      * n = 10, (row, col) = (5, 11)
183      * n = 10, (row, col) = (5, 0)
184      * n = 10, (row, col) = (-2147483648, -2147483648)
185      * n = 10, (row, col) = (2147483647, 2147483647)
186  ==> passed

```

```

187
188 Test 8: call constructor with invalid argument
189     * n = -10
190     * n = -1
191     * n = 0
192 ==> passed
193
194 Test 9: create multiple Percolation objects at the same time
195         (to make sure you didn't store data in static variables)
196 ==> passed
197
198 Test 10: open predetermined list of sites using file inputs,
199          but permute the order in which methods are called
200     * filename = input8.txt; order = isFull(), isOpen(), percolates()
201     * filename = input8.txt; order = isFull(), percolates(), isOpen()
202     * filename = input8.txt; order = isOpen(), isFull(), percolates()
203     * filename = input8.txt; order = isOpen(), percolates(), isFull()
204     * filename = input8.txt; order = percolates(), isOpen(), isFull()
205     * filename = input8.txt; order = percolates(), isFull(), isOpen()
206 ==> passed
207
208 Test 11: call open(), isOpen(), and numberOfOpenSites()
209          in random order until system percolates
210     * n = 3, trials = 40, p = (0.4, 0.4, 0.0, 0.0, 0.3)
211     * n = 5, trials = 20, p = (0.4, 0.4, 0.0, 0.0, 0.3)
212     * n = 7, trials = 10, p = (0.4, 0.4, 0.0, 0.0, 0.3)
213     * n = 10, trials = 5, p = (0.4, 0.4, 0.0, 0.0, 0.3)
214     * n = 20, trials = 2, p = (0.4, 0.4, 0.0, 0.0, 0.3)
215     * n = 50, trials = 1, p = (0.4, 0.4, 0.0, 0.0, 0.3)
216 ==> passed
217
218 Test 12: call open() and percolates() in random order until system percolates
219     * n = 3, trials = 40, p = (0.5, 0.0, 0.0, 0.5, 0.0)
220     * n = 5, trials = 20, p = (0.5, 0.0, 0.0, 0.5, 0.0)
221     * n = 7, trials = 10, p = (0.5, 0.0, 0.0, 0.5, 0.0)
222     * n = 10, trials = 5, p = (0.5, 0.0, 0.0, 0.5, 0.0)
223     * n = 20, trials = 2, p = (0.5, 0.0, 0.0, 0.5, 0.0)
224     * n = 50, trials = 1, p = (0.5, 0.0, 0.0, 0.5, 0.0)
225 ==> passed
226
227 Test 13: call open() and isFull() in random order until system percolates
228     * n = 3, trials = 40, p = (0.5, 0.0, 0.5, 0.0, 0.0)
229     * n = 5, trials = 20, p = (0.5, 0.0, 0.5, 0.0, 0.0)
230     * n = 7, trials = 10, p = (0.5, 0.0, 0.5, 0.0, 0.0)
231     * n = 10, trials = 5, p = (0.5, 0.0, 0.5, 0.0, 0.0)
232     * n = 20, trials = 2, p = (0.5, 0.0, 0.5, 0.0, 0.0)
233     * n = 50, trials = 1, p = (0.5, 0.0, 0.5, 0.0, 0.0)
234 ==> passed
235
236 Test 14: call all methods in random order until system percolates
237     * n = 3, trials = 40, p = (0.2, 0.2, 0.2, 0.2, 0.2)
238     * n = 5, trials = 20, p = (0.2, 0.2, 0.2, 0.2, 0.2)
239     * n = 7, trials = 10, p = (0.2, 0.2, 0.2, 0.2, 0.2)
240     * n = 10, trials = 5, p = (0.2, 0.2, 0.2, 0.2, 0.2)
241     * n = 20, trials = 2, p = (0.2, 0.2, 0.2, 0.2, 0.2)
242     * n = 50, trials = 1, p = (0.2, 0.2, 0.2, 0.2, 0.2)
243 ==> passed
244
245 Test 15: call all methods in random order until almost all sites are open,
246          but with inputs not prone to backwash
247     * n = 3
248     * n = 5
249     * n = 7
250     * n = 10
251     * n = 20
252     * n = 50
253 ==> passed
254
255 Test 16: check for backwash with predetermined sites
256     * filename = input20.txt
257     * filename = input10.txt
258     * filename = input50.txt
259     * filename = jerry47.txt

```

```

260     * filename = sedgewick60.txt
261     * filename = wayne98.txt
262
263
264     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
265     OperationCountLimitExceededException
266     Number of calls to methods in WeightedQuickUnionUF exceeds limit: 250000000
267     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
268
269 ==> FAILED
270
271 Test 17: check for backwash with predetermined sites that have
272         multiple percolating paths
273     * filename = input3.txt
274     * filename = input4.txt
275     * filename = input7.txt
276 ==> passed
277
278 Test 18: call all methods in random order until all sites are open,
279         allowing isOpen() to be called on a site more than once
280         (these inputs are prone to backwash)
281     * n = 3
282     * n = 5
283     * n = 7
284     * n = 10
285     * n = 20
286     * n = 50
287 ==> passed
288
289
290 Total: 17/18 tests passed!
291
292
293 =====
294 *****
295 * TESTING CORRECTNESS (substituting reference Percolation)
296 *****
297
298 Testing correctness of PercolationStats
299 *-----
300 Running 15 total tests.
301
302 Test 1: check that methods in PercolationStats do not print to standard output
303     * n = 20, trials = 10
304     * n = 50, trials = 20
305     * n = 100, trials = 50
306     * n = 64, trials = 150
307 ==> passed
308
309 Test 2: check that mean() returns value in expected range
310     * n = 2, trials = 10000
311     * n = 5, trials = 10000
312     * n = 10, trials = 10000
313     * n = 25, trials = 10000
314 ==> passed
315
316 Test 3: check that stddev() returns value in expected range
317     * n = 2, trials = 10000
318     * n = 5, trials = 10000
319     * n = 10, trials = 10000
320     * n = 25, trials = 10000
321 ==> passed
322
323 Test 4: check that PercolationStats creates trials Percolation objects, each of size
n-by-n
324     * n = 20, trials = 10
325     * n = 50, trials = 20
326     * n = 100, trials = 50
327     * n = 64, trials = 150
328 ==> passed
329
330 Test 5: check that PercolationStats calls open() until system percolates
331     * n = 20, trials = 10

```

```

332     * n = 50, trials = 20
333     * n = 100, trials = 50
334     * n = 64, trials = 150
335 ==> passed
336
337 Test 6: check that PercolationStats does not call open() after system percolates
338     * n = 20, trials = 10
339     * n = 50, trials = 20
340     * n = 100, trials = 50
341     * n = 64, trials = 150
342 ==> passed
343
344 Test 7: check that mean() is consistent with the number of intercepted calls to open()
345         on blocked sites
346     * n = 20, trials = 10
347     * n = 50, trials = 20
348     * n = 100, trials = 50
349     * n = 64, trials = 150
350 ==> passed
351
352 Test 8: check that stddev() is consistent with the number of intercepted calls to
353         open()
354         on blocked sites
355     * n = 20, trials = 10
356     * n = 50, trials = 20
357     * n = 100, trials = 50
358     * n = 64, trials = 150
359 ==> passed
360
361 Test 9: check that confidenceLo() and confidenceHigh() are consistent with mean()
362         and stddev()
363     * n = 20, trials = 10
364         - PercolationStats confidence low = -0.007806421393002361
365         - PercolationStats confidence high = 1.2318064213930024
366         - PercolationStats mean = 0.612
367         - PercolationStats stddev = 0.045625285387235315
368         - T = 10
369         - student T = 10
370         - mean - 1.96 * stddev / sqrt(T) = 0.5837211551391033
371         - mean + 1.96 * stddev / sqrt(T) = 0.6402788448608967
372
373     * n = 50, trials = 20
374         - PercolationStats confidence low = 0.15551067641004124
375         - PercolationStats confidence high = 1.0320493235899586
376         - PercolationStats mean = 0.59378
377         - PercolationStats stddev = 0.019869246270767818
378         - T = 20
379         - student T = 20
380         - mean - 1.96 * stddev / sqrt(T) = 0.5850719188766682
381         - mean + 1.96 * stddev / sqrt(T) = 0.6024880811233317
382
383     * n = 100, trials = 50
384         - PercolationStats confidence low = 0.3185021417748734
385         - PercolationStats confidence high = 0.8728738582251265
386         - PercolationStats mean = 0.595688
387         - PercolationStats stddev = 0.01693840751014073
388         - T = 50
389         - student T = 50
390         - mean - 1.96 * stddev / sqrt(T) = 0.5909929129773347
391         - mean + 1.96 * stddev / sqrt(T) = 0.6003830870226653
392
393     * n = 64, trials = 150
394         - PercolationStats confidence low = 0.4314184930548324
395         - PercolationStats confidence high = 0.7514851527785009
396         - PercolationStats mean = 0.5914518229166666
397         - PercolationStats stddev = 0.02079900623081705
398         - T = 150
399         - student T = 150
400         - mean - 1.96 * stddev / sqrt(T) = 0.5881232886917319
401         - mean + 1.96 * stddev / sqrt(T) = 0.5947803571416014
402 ==> FAILED

```

```

403 Test 10: check that exception is thrown if either n or trials is out of bounds
404 * n = -23, trials = 42
405 * n = 23, trials = 0
406 * n = -42, trials = 0
407 * n = 42, trials = -1
408 * n = -2147483648, trials = -2147483648
409 ==> passed
410
411 Test 11: create two PercolationStats objects at the same time and check mean()
412 (to make sure you didn't store data in static variables)
413 * n1 = 50, trials1 = 10, n2 = 50, trials2 = 5
414 * n1 = 50, trials1 = 5, n2 = 50, trials2 = 10
415 * n1 = 50, trials1 = 10, n2 = 25, trials2 = 10
416 * n1 = 25, trials1 = 10, n2 = 50, trials2 = 10
417 * n1 = 50, trials1 = 10, n2 = 15, trials2 = 100
418 * n1 = 15, trials1 = 100, n2 = 50, trials2 = 10
419 ==> passed
420
421 Test 12: check that the methods return the same value, regardless of
422 the order in which they are called
423 * n = 20, trials = 10
424 * n = 50, trials = 20
425 * n = 100, trials = 50
426 * n = 64, trials = 150
427 ==> passed
428
429 Test 13: check that no calls to StdRandom.setSeed()
430 * n = 20, trials = 10
431 * n = 20, trials = 10
432 * n = 40, trials = 10
433 * n = 80, trials = 10
434 ==> passed
435
436 Test 14: check distribution of number of sites opened until percolation
437 * n = 2, trials = 100000
438 * n = 3, trials = 100000
439 * n = 4, trials = 100000
440 ==> passed
441
442 Test 15: check that each site is opened the expected number of times
443 * n = 2, trials = 100000
444 * n = 3, trials = 100000
445 * n = 4, trials = 100000
446 ==> passed
447
448
449 Total: 14/15 tests passed!
450
451
452 =====
453 *****
454 * MEMORY (substituting reference Percolation)
455 *****
456
457 Analyzing memory of PercolationStats
458 *-----
459 Running 4 total tests.
460
461 Test 1a-1d: check memory usage as a function of T trials for n = 100
462 (max allowed: 8*T + 128 bytes)
463
464 T bytes
465 -----
466 => passed 16 48
467 => passed 32 48
468 => passed 64 48
469 => passed 128 48
470 ==> 4/4 tests passed
471
472
473 Estimated student memory = 48.00 (R^2 = 1.000)
474
475 Total: 4/4 tests passed!

```

```

476
477
478 =====
479
480
481
482 *****
483 * TIMING (substituting reference Percolation)
484 *****
485
486 Timing PercolationStats
487 *-----
488 Running 4 total tests.
489
490 Test 1: count calls to StdStats.mean() and StdStats.stddev()
491 * n = 20, trials = 10
492 * n = 50, trials = 20
493 * n = 100, trials = 50
494 * n = 64, trials = 150
495 ==> passed
496
497 Test 2: count calls to methods in StdRandom
498 * n = 20, trials = 10
499 * n = 20, trials = 10
500 * n = 40, trials = 10
501 * n = 80, trials = 10
502 ==> passed
503
504 Test 3: count calls to methods in Percolation
505 * n = 20, trials = 10
506 * n = 50, trials = 20
507 * n = 100, trials = 50
508 * n = 64, trials = 150
509 ==> passed
510
511 Test 4: Call PercolationStats methods with trials = 3 and values of n that go up
512 by a factor of sqrt(2). The test passes when n reaches 2,896.
513
514 The approximate order-of-growth is  $n^{\log \text{ratio}}$ 
515
516 n seconds log ratio
517 -----
518 512 0.10 2.2
519 724 0.26 2.6
520 1024 0.63 2.5
521 1448 1.46 2.4
522 2048 3.32 2.4
523 2896 7.27 2.3
524 ==> passed
525
526
527 Total: 4/4 tests passed!
528
529
530 =====
531
532
533 *****
534 * MEMORY
535 *****
536
537 Analyzing memory of Percolation
538 *-----
539 Running 4 total tests.
540
541 Test 1a-1d: check that total memory  $\leq 17 n^2 + 128 n + 1024$  bytes
542
543 n bytes
544 -----
545
546 => passed 64 69920
547 => passed 256 1114400
548 => passed 512 4456736

```



```

549 => passed      1024      17826080
550 ==> 4/4 tests passed
551
552
553 Estimated student memory = 17.00 n^2 + 0.00 n + 288.00    (R^2 = 1.000)
554
555
556 Test 2 (bonus): check that total memory <= 11 n^2 + 128 n + 1024 bytes
557     - failed memory test for n = 64
558 ==> FAILED
559
560
561 Total: 4/4 tests passed!
562
563
564 =====
565
566
567
568 *****
569 *   TIMING
570 *****
571
572 Timing Percolation
573 *-----
574 Running 16 total tests.
575
576 Test 1a-1e: Creates an n-by-n percolation system; open sites at random until
577             the system percolates, interleaving calls to percolates() and open().
578             Count calls to connected(), union() and find().
579
580                                     2 * connected()
581                                n      union()      + find()      constructor
582 -----
583 => passed      16          406          318          2
584 => passed      32         1350         1152          2
585 => passed      64         6044         4960          2
586 => passed     128        23328        19582          2
587 => passed     256        95773        79118          2
588 => passed     512       361245       307734          2
589 => passed    1024      1487377      1248460          2
590 ==> 7/7 tests passed
591
592
593 If one of the values in the table violates the performance limits
594 the factor by which you failed the test appears in parentheses.
595 For example, (9.6x) in the union() column indicates that it uses
596 9.6x too many calls.
597
598
599 Tests 2a-2f: Check whether the number of calls to union(), connected(), and find()
600             is a constant per call to open(), isOpen(), isFull(), and percolates().
601             The table shows the maximum number of union() and find() calls made
602             during a single call to open(), isOpen(), isFull(), and percolates().
603             One call to connected() counts as two calls to find().
604
605                                n      per open()      per isOpen()      per isFull()      per
606                                percolates()
607 -----
608 -----
609 => passed      16          8          0          2          2
610 => passed      32          8          0          2          2
611 => passed      64          8          0          2          2
612 => passed     128          8          0          2          2
613 => passed     256          8          0          2          2
614 => passed     512          8          0          2          2
615 => passed    1024          8          0          2          2
616 ==> 7/7 tests passed
617
618 Running time (in seconds) depends on the machine on which the script runs.
619

```

620  
621 Test 3: Create an n-by-n percolation system; interleave calls to percolates()  
622 and open() until the system percolates. The values of n go up by a  
623 factor of sqrt(2). The test is passed if n >= 4096 in under 10 seconds.  
624

625 The approximate order-of-growth is  $n^{(\log \text{ ratio})}$   
626

	n	seconds	log ratio	union-find operations	log ratio
627					
628					
629					
630	724	0.13	1.7	2068718	2.0
631	1024	0.31	2.6	4194368	2.0
632	1448	0.70	2.4	8333532	2.0
633	2048	1.25	1.7	16590548	2.0
634	2896	2.81	2.3	33349242	2.0
635	4096	7.14	2.7	67220450	2.0

636 ==> passed  
637  
638  
639

640 Test 4: Create an n-by-n percolation system; interleave calls to open(),  
641 percolates(), isOpen(), isFull(), and numberOfOpenSites() until.  
642 the system percolates. The values of n go up by a factor of sqrt(2).  
643 The test is passed if n >= 4096 in under 10 seconds.  
644

	n	seconds	log ratio	union-find operations	log ratio
645					
646					
647					
648	724	0.15	2.5	2709302	1.9
649	1024	0.33	2.3	5399334	2.0
650	1448	0.70	2.1	10847968	2.0
651	2048	1.38	2.0	21812142	2.0
652	2896	2.93	2.2	43458604	2.0
653	4096	6.04	2.1	86769526	2.0

654 ==> passed  
655  
656

657 Total: 16/16 tests passed!  
658  
659

660 =====