Collinear Points

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

ASSESSMENT SUMMARY

Compilation:  PASSED
API:          PASSED

Spotbugs:     PASSED
PMD:          PASSED
Checkstyle:   PASSED

Correctness:  41/41 tests passed
Memory:       1/1 tests passed
Timing:       41/41 tests passed

Aggregate score: 100.00%
[Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%,
Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:
----------------------------------
3.3K Oct 18 04:13 BruteCollinearPoints.java
4.4K Oct 18 04:13 FastCollinearPoints.java
4.0K Oct 18 04:13 Point.java


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


% javac Point.java
*-----------------------------------------------------------

% javac LineSegment.java
*-----------------------------------------------------------

% javac BruteCollinearPoints.java
*-----------------------------------------------------------

% javac FastCollinearPoints.java
*-----------------------------------------------------------


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


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

BruteCollinearPoints:

FastCollinearPoints:
```

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


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


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


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


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


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


% checkstyle *.java
*-----------------------------------------------------------

% custom checkstyle checks for Point.java
*-----------------------------------------------------------

% custom checkstyle checks for BruteCollinearPoints.java
*-----------------------------------------------------------

% custom checkstyle checks for FastCollinearPoints.java
*-----------------------------------------------------------


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


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

Testing correctness of Point
*-----------------------------------------------------------
Running 3 total tests.

Test 1: p.slopeTo(q)
  * positive infinite slope, where p and q have coordinates in [0, 500)
  * positive infinite slope, where p and q have coordinates in [0, 32768)
  * negative infinite slope, where p and q have coordinates in [0, 500)
  * negative infinite slope, where p and q have coordinates in [0, 32768)
  * positive zero     slope, where p and q have coordinates in [0, 500)
  * positive zero     slope, where p and q have coordinates in [0, 32768)
  * symmetric for random points p and q with coordinates in [0, 500)
  * symmetric for random points p and q with coordinates in [0, 32768)
  * transitive for random points p, q, and r with coordinates in [0, 500)
  * transitive for random points p, q, and r with coordinates in [0, 32768)
  * slopeTo(), where p and q have coordinates in [0, 500)
  * slopeTo(), where p and q have coordinates in [0, 32768)
  * slopeTo(), where p and q have coordinates in [0, 10)
  * throw a java.lang.NullPointerException if argument is null
==> passed
```

```
Test 2: p.compareTo(q)
  * reflexive, where p and q have coordinates in [0, 500)
  * reflexive, where p and q have coordinates in [0, 32768)
  * antisymmetric, where p and q have coordinates in [0, 500)
  * antisymmetric, where p and q have coordinates in [0, 32768)
  * transitive, where p, q, and r have coordinates in [0, 500)
  * transitive, where p, q, and r have coordinates in [0, 32768)
  * sign of compareTo(), where p and q have coordinates in [0, 500)
  * sign of compareTo(), where p and q have coordinates in [0, 32768)
  * sign of compareTo(), where p and q have coordinates in [0, 10)
  * throw java.lang.NullPointerException exception if argument is null
==> passed

Test 3: p.slopeOrder().compare(q, r)
  * reflexive, where p and q have coordinates in [0, 500)
  * reflexive, where p and q have coordinates in [0, 32768)
  * antisymmetric, where p, q, and r have coordinates in [0, 500)
  * antisymmetric, where p, q, and r have coordinates in [0, 32768)
  * transitive, where p, q, r, and s have coordinates in [0, 500)
  * transitive, where p, q, r, and s have coordinates in [0, 32768)
  * sign of compare(), where p, q, and r have coordinates in [0, 500)
  * sign of compare(), where p, q, and r have coordinates in [0, 32768)
  * sign of compare(), where p, q, and r have coordinates in [0, 10)
  * throw java.lang.NullPointerException if either argument is null
==> passed


Total: 3/3 tests passed!


================================================================
********************************************************************************
*  TESTING CORRECTNESS (substituting reference Point and LineSegment)
********************************************************************************

Testing correctness of BruteCollinearPoints
*-----------------------------------------------------------
Running 17 total tests.

The inputs satisfy the following conditions:
  - no duplicate points
  - no 5 (or more) points are collinear
  - all x- and y-coordinates between 0 and 32,767

Test 1: points from a file
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 2a: points from a file with horizontal line segments
  * filename = horizontal5.txt
  * filename = horizontal25.txt
==> passed

Test 2b: random horizontal line segments
  *  1 random horizontal line segment
  *  5 random horizontal line segments
  * 10 random horizontal line segments
  * 15 random horizontal line segments
==> passed
```

```
Test 3a: points from a file with vertical line segments
  * filename = vertical5.txt
  * filename = vertical25.txt
==> passed

Test 3b: random vertical line segments
  *  1 random vertical line segment
  *  5 random vertical line segments
  * 10 random vertical line segments
  * 15 random vertical line segments
==> passed

Test 4a: points from a file with no line segments
  * filename = random23.txt
  * filename = random38.txt
==> passed

Test 4b: random points with no line segments
  *  5 random points
  * 10 random points
  * 20 random points
  * 50 random points
==> passed

Test 5: points from a file with fewer than 4 points
  * filename = input1.txt
  * filename = input2.txt
  * filename = input3.txt
==> passed

Test 6: check for dependence on either compareTo() or compare()
        returning { -1, +1, 0 } instead of { negative integer,
        positive integer, zero }
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 7: check for fragile dependence on return value of toString()
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 8: random line segments, none vertical or horizontal
  *  1 random line segment
  *  5 random line segments
  * 10 random line segments
  * 15 random line segments
==> passed

Test 9: random line segments
  *  1 random line segment
  *  5 random line segments
  * 10 random line segments
  * 15 random line segments
==> passed

Test 10: check that data type is immutable by testing whether each method
         returns the same value, regardless of any intervening operations
  * input8.txt
  * equidistant.txt
==> passed
```

```
Test 11: check that data type does not mutate the constructor argument
  * input8.txt
  * equidistant.txt
==> passed

Test 12: numberOfSegments() is consistent with segments()
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = horizontal5.txt
  * filename = vertical5.txt
  * filename = random23.txt
==> passed

Test 13: throws an exception if either the constructor argument is null
         or any entry in array is null
  * argument is null
  * Point[] of length 10, number of null entries = 1
  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
  * Point[] of length 3, number of null entries = 1
  * Point[] of length 2, number of null entries = 1
  * Point[] of length 1, number of null entries = 1
==> passed

Test 14: check that the constructor throws an exception if duplicate points
  * 50 points
  * 25 points
  * 5 points
  * 4 points
  * 3 points
  * 2 points
==> passed


Total: 17/17 tests passed!


================================================================
Testing correctness of FastCollinearPoints
*-----------------------------------------------------------
Running 21 total tests.

The inputs satisfy the following conditions:
  - no duplicate points
  - all x- and y-coordinates between 0 and 32,767

Test 1: points from a file
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = input299.txt
==> passed

Test 2a: points from a file with horizontal line segments
  * filename = horizontal5.txt
  * filename = horizontal25.txt
  * filename = horizontal50.txt
  * filename = horizontal75.txt
  * filename = horizontal100.txt
```

```
==> passed

Test 2b: random horizontal line segments
  *  1 random horizontal line segment
  *  5 random horizontal line segments
  * 10 random horizontal line segments
  * 15 random horizontal line segments
==> passed

Test 3a: points from a file with vertical line segments
  * filename = vertical5.txt
  * filename = vertical25.txt
  * filename = vertical50.txt
  * filename = vertical75.txt
  * filename = vertical100.txt
==> passed

Test 3b: random vertical line segments
  *  1 random vertical line segment
  *  5 random vertical line segments
  * 10 random vertical line segments
  * 15 random vertical line segments
==> passed

Test 4a: points from a file with no line segments
  * filename = random23.txt
  * filename = random38.txt
  * filename = random91.txt
  * filename = random152.txt
==> passed

Test 4b: random points with no line segments
  *  5 random points
  * 10 random points
  * 20 random points
  * 50 random points
==> passed

Test 5a: points from a file with 5 or more on some line segments
  * filename = input9.txt
  * filename = input10.txt
  * filename = input20.txt
  * filename = input50.txt
  * filename = input80.txt
  * filename = input300.txt
  * filename = inarow.txt
==> passed

Test 5b: points from a file with 5 or more on some line segments
  * filename = kw1260.txt
  * filename = rs1423.txt
==> passed

Test 6: points from a file with fewer than 4 points
  * filename = input1.txt
  * filename = input2.txt
  * filename = input3.txt
==> passed

Test 7: check for dependence on either compareTo() or compare()
        returning { -1, +1, 0 } instead of { negative integer,
        positive integer, zero }
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = input299.txt
==> passed

Test 8: check for fragile dependence on return value of toString()
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 9: random line segments, none vertical or horizontal
  *  1 random line segment
  *  5 random line segments
  * 25 random line segments
  * 50 random line segments
  * 100 random line segments
==> passed

Test 10: random line segments
  *  1 random line segment
  *  5 random line segments
  * 25 random line segments
  * 50 random line segments
  * 100 random line segments
==> passed

Test 11: random distinct points in a given range
  * 5 random points in a 10-by-10 grid
  * 10 random points in a 10-by-10 grid
  * 50 random points in a 10-by-10 grid
  * 90 random points in a 10-by-10 grid
  * 200 random points in a 50-by-50 grid
==> passed

Test 12: m*n points on an m-by-n grid
  * 3-by-3 grid
  * 4-by-4 grid
  * 5-by-5 grid
  * 10-by-10 grid
  * 20-by-20 grid
  * 5-by-4 grid
  * 6-by-4 grid
  * 10-by-4 grid
  * 15-by-4 grid
  * 25-by-4 grid
==> passed

Test 13: check that data type is immutable by testing whether each method
         returns the same value, regardless of any intervening operations
  * input8.txt
  * equidistant.txt
==> passed

Test 14: check that data type does not mutate the constructor argument
  * input8.txt
  * equidistant.txt
==> passed

Test 15: numberOfSegments() is consistent with segments()
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = horizontal5.txt
  * filename = vertical5.txt
  * filename = random23.txt
==> passed

Test 16: throws an exception if either constructor argument is null
         or any entry in array is null
  * argument is null
  * Point[] of length 10, number of null entries = 1
  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
  * Point[] of length 3, number of null entries = 1
  * Point[] of length 2, number of null entries = 1
  * Point[] of length 1, number of null entries = 1
==> passed

Test 17: check that the constructor throws an exception if duplicate points
  * 50 points
  * 25 points
  * 5 points
  * 4 points
  * 3 points
  * 2 points
==> passed


Total: 21/21 tests passed!


================================================================
********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Point
*-----------------------------------------------------------
Running 1 total tests.

The maximum amount of memory per Point object is 32 bytes.

Student memory = 24 bytes (passed)

Total: 1/1 tests passed!


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



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

Timing BruteCollinearPoints
*-----------------------------------------------------------
Running 10 total tests.

Test 1a-1e: Find collinear points among n random distinct points


                                                      slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
```

```
--------------------------------------------------------------------------------
--------------
=> passed    16   0.00        3640           0           3640                   62
=> passed    32   0.00       71920           0          71920                  153
=> passed    64   0.02     1270752           0        1270752                  365
=> passed   128   0.08    21336000           0       21336000                  872
=> passed   256   1.22   349585280           0      349585280                 1985
==> 5/5 tests passed

Test 2a-2e: Find collinear points among n/4 arbitrary line segments


                                                      slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
--------------------------------------------------------------------------------
--------------
=> passed    16   0.00        3802           0           3802                   60
=> passed    32   0.00       72576           0          72576                  153
=> passed    64   0.01     1273550           0        1273550                  373
=> passed   128   0.08    21348616           0       21348616                  869
=> passed   256   1.28   349635170           0      349635170                 1988
==> 5/5 tests passed

Total: 10/10 tests passed!


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



Timing FastCollinearPoints
*-----------------------------------------------------------
Running 31 total tests.

Test 1a-1g: Find collinear points among n random distinct points


                                                      slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
--------------------------------------------------------------------------------
--------------
=> passed    64   0.01        7936       18788          45512                18789
=> passed   128   0.01       32256       89655         211566                88733
=> passed   256   0.03      130048      413913         957874               415528
=> passed   512   0.20      522240     1887601        4297442              1885992
=> passed  1024   0.44     2093059     8553881       19200821              8555819
=> passed  2048   1.28     8380479    38095593       84571665             38170147
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (84571665 / 19200821) = 2.14
=> passed

==> 7/7 tests passed

Test 2a-2g: Find collinear points among the n points on an n-by-1 grid

                                                      slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
--------------------------------------------------------------------------------
--------------
=> passed    64   0.00        4160        4764          13688                 7136
=> passed   128   0.00       16512       17796          52104                23193
=> passed   256   0.00       65792       68717         203226                80297
=> passed   512   0.01      262656      269399         801454              293565
=> passed  1024   0.03     1049600     1065026        3179652             1114901
=> passed  2048   0.07     4196352     4231214       12658780             4333590
=> passed  4096   0.31    16781312    16859163       50499638            17068504
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (50499638 / 12658780) = 2.00
=> passed

==> 8/8 tests passed

Test 3a-3g: Find collinear points among the n points on an n/4-by-4 grid

                                                      slopeTo()
           n    time     slopeTo()   compare()  + 2*compare()        compareTo()
--------------------------------------------------------------------------------
--------------
=> passed    64   0.00        6796       14906          36608               17254
=> passed   128   0.00       27084       43854         114792               64675
=> passed   256   0.01      108108      149618         407344              246259
=> passed   512   0.02      431948      548156        1528260              950082
=> passed  1024   0.07     1726796     2087496        5901788             3710996
=> passed  2048   0.25     6905164     8122445       23150054            14600594
=> passed  4096   0.97    27616588    31990953       91598494            57806408
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (91598494 / 23150054) = 1.98
=> passed

==> 8/8 tests passed

Test 4a-4g: Find collinear points among the n points on an n/8-by-8 grid

                                                      slopeTo()
           n    time     slopeTo()   compare()  + 2*compare()        compareTo()
--------------------------------------------------------------------------------
--------------
=> passed    64   0.00        6916       18045          43006               19197
=> passed   128   0.00       27688       75863         179414               86024
=> passed   256   0.01      110640      232229         575098              342237
=> passed   512   0.04      442208      854545        2151298             1346664
=> passed  1024   0.14     1767948     3260991        8289930             5336317
=> passed  2048   0.43     7069900    12699218       32468336            21239992
=> passed  4096   1.70    28275652    50043244      128362140            84631641
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (128362140 / 32468336) = 1.98
=> passed

==> 8/8 tests passed

Total: 31/31 tests passed!


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