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%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
3.6K Aug 23 07:30 BruteCollinearPoints.java
3.4K Aug 23 07:30 FastCollinearPoints.java
4.3K Aug 23 07:30 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        1800           0          1800                 0
  => passed    32   0.00       15376           0         15376                 0
  => passed    64   0.00      127008           0        127008                 0
  => passed   128   0.01     1032256           0       1032256                 0
  => passed   256   0.05     8323200           0       8323200                 0
  ==> 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        1938           0           1938                   22
=> passed    32   0.00       16048           0          16048                   42
=> passed    64   0.00      129870           0         129870                   81
=> passed   128   0.01     1043869           0        1043869                  163
=> passed   256   0.03     8370888           0        8370888                  348
==> 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        8064       18528          45120                 2328
=> passed   128   0.01       32512       89282         211076                 8871
=> passed   256   0.02      130560      415814         962188                34371
=> passed   512   0.15      523264     1896467        4316198               134782
=> passed  1024   0.45     2095104     8540563       19176230               532743
=> passed  2048   0.92     8384512    38089507       84563526              2116093
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (84563526 / 19176230) = 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        8064        4764          17592                 2384
=> passed   128   0.00       32512       17796          68104                 8994
=> passed   256   0.00      130560       68717         267994                34633
=> passed   512   0.01      523264      269399        1062062               135302
=> passed  1024   0.03     2095104     1065026        4225156               533777
=> passed  2048   0.07     8384512     4231214       16846940              2118192
=> passed  4096   0.28    33546240    16859163       67264566              8434742
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (67264566 / 16846940) = 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        8064       14906          37876                 2737
=> passed   128   0.00       32512       43854         120220                10361
=> passed   256   0.00      130560      149618         429796                40101
=> passed   512   0.01      523264      548156        1619576               157146
=> passed  1024   0.04     2095104     2087496        6270096               621164
=> passed  2048   0.14     8384512     8122445       24629402              2467657
=> passed  4096   0.47    33546240    31990953       97528146              9832849
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (97528146 / 24629402) = 1.99
=> 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        8064       18045          44154                 2717
=> passed   128   0.00       32512       75863         184238                10271
=> passed   256   0.00      130560      232229         595018                39745
=> passed   512   0.02      523264      854545        2232354               155792
=> passed  1024   0.07     2095104     3260991        8617086               615706
=> passed  2048   0.18     8384512    12699218       33782948              2445829
=> passed  4096   0.68    33546240    50043244      133632728              9745454
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (133632728 / 33782948) = 1.98
=> passed

==> 8/8 tests passed

Total: 31/31 tests passed!


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