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:       36/41 tests passed

Aggregate score: 97.56%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
3.2K Nov 12 20:46 BruteCollinearPoints.java
5.7K Nov 12 20:46 FastCollinearPoints.java
3.8K Nov 12 20:46 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                 167
=> passed    32   0.00       71920           0          71920                 616
=> passed    64   0.01     1270752           0        1270752                2327
=> passed   128   0.03    21336000           0       21336000                8862
=> passed   256   1.25   349585472           0      349585472               34383
==> 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        3840           0           3840                 168
=> passed    32   0.00       72638           0          72638                 616
=> passed    64   0.01     1274036           0        1274036                2322
=> passed   128   0.09    21348858           0       21348858                8867
=> passed   256   1.58   349638502           0      349638502               34370
==> 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       18731          45526                2016
=> passed   128   0.01       32512       87927         208366                8128
=> passed   256   0.02      130560      413736         958032               32640
=> passed   512   0.16      523264     1896617        4316498              130816
=> passed  1024   0.44     2095104     8515310       19125724              523776
=> passed  2048   0.88     8384512    37864565       84113642             2096152
==> 6/6 tests passed
```

```
lg ratio(slopeTo() + 2*compare()) = lg (84113642 / 19125724) = 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               21603
=> passed   128   0.01        32512       17796          68104              103060
=> FAILED   256   0.02       130560       68717         267994              475456   (1.2x)
=> FAILED   512   0.06       523264      269399        1062062             2167053   (1.5x)
=> FAILED  1024   0.19      2095104     1065026        4225156             9698206   (1.7x)
=> FAILED  2048   0.63      8384512     4231214       16846940            43013970   (2.0x)
=> FAILED  4096   2.73     33546240    16859163       67264566           188827714   (2.2x)
==> 2/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (67264566 / 16846940) = 2.00
=> passed

==> 3/8 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       17444          42952                6984
=> passed   128   0.00        32512       72805         178122               27051
=> passed   256   0.01       130560      283775         698110               96952
=> passed   512   0.03       523264     1108562        2740388              360371
=> passed  1024   0.11      2095104     4417992       10931088             1387945
=> passed  2048   0.34      8384512    17561579       43507670             5443686
=> passed  4096   1.23     33546240    69903061      173352362            21518717
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (173352362 / 43507670) = 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       18441          44946                6341
=> passed   128   0.00        32512       87471         207454               26555
=> passed   256   0.01       130560      384488         899536              103525
=> passed   512   0.04       523264     1627757        3778778              391270
=> passed  1024   0.15      2095104     6738435       15571974             1509753
=> passed  2048   0.57      8384512    27752778       63890068             5937225
=> passed  4096   2.15     33546240   112910302      259366844            23547702
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (259366844 / 63890068) = 2.02
=> passed

==> 8/8 tests passed

Total: 26/31 tests passed!


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