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:  40/41 tests passed
Memory:       1/1 tests passed
Timing:       41/41 tests passed

Aggregate score: 98.54%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
4.6K Jun  7 10:47 BruteCollinearPoints.java
 11K Jun  7 10:47 FastCollinearPoints.java
4.3K Jun  7 10:47 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
    - data type mutated the points[] array
    - data type should have no side effects unless documented in API

  * equidistant.txt
    - data type mutated the points[] array
    - data type should have no side effects unless documented in API

==> FAILED

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: 16/17 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                   60
=> passed    32   0.00       71920           0          71920                  154
=> passed    64   0.01     1270752           0        1270752                  371
=> passed   128   0.08    21336000           0       21336000                  861
=> passed   256   0.32   349585280           0      349585280                 1995
==> 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        3828           0           3828                   61
=> passed    32   0.00       72712           0          72712                  153
=> passed    64   0.01     1273932           0        1273932                  362
=> passed   128   0.09    21348870           0       21348870                  865
=> passed   256   1.65   349637930           0      349637930                 1986
==> 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   0.00        4032       18278          40588                 372
=> passed   128   0.01       16256       87435         191126                 875
=> passed   256   0.02       65280      409827         884934                1983
=> passed   512   0.13      261632     1867731        3997094                4488
=> passed  1024   0.32     1047552     8476297       18000146                9989
=> passed  2048   0.88     4192256    37854029       79900314               21993
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (79900314 / 18000146) = 2.15
=> 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        4032        3968          11968                7180
=> passed   128   0.00       16256       16128          48512               23219
=> passed   256   0.00       65280       65024         195328               80314
=> passed   512   0.02      261632      261120         783872              293575
=> passed  1024   0.06     1047552     1046528        3140608             1114965
=> passed  2048   0.14     4192256     4190208       12572672             4333615
=> passed  4096   0.34    16773120    16769024       50311168            17068565
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (50311168 / 12572672) = 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        4032       16222          36476                5091
=> passed   128   0.00       16256       58163         132582               17876
=> passed   256   0.01       65280      152338         369956               61995
=> passed   512   0.02      261632      546156        1353944              225003
=> passed  1024   0.05     1047552     2080221        5207994              847609
=> passed  2048   0.14     4192256     8125112       20442480             3275630
=> passed  4096   0.48    16773120    32104200       80981520            12856625
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (80981520 / 20442480) = 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        4032       17691          39414                4404
=> passed   128   0.00       16256       80299         176854               18110
=> passed   256   0.01       65280      311802         688884                 67032
=> passed   512   0.02      261632      859701        1981034                248439
=> passed  1024   0.07     1047552     3252409        7552370                946562
=> passed  2048   0.18     4192256    12677306       29546868               3679012
=> passed  4096   0.68    16773120    50035440      116844000              14482226
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (116844000 / 29546868) = 1.98
=> passed

==> 8/8 tests passed

Total: 31/31 tests passed!


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