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

ASSESSMENT SUMMARY

Compilation:  PASSED
API:         PASSED

Spotbugs:    FAILED (4 warnings)
PMD:         FAILED (6 warnings)
Checkstyle:   FAILED (0 errors, 22 warnings)

Correctness:  35/41 tests passed
Memory:      1/1 tests passed
Timing:      26/41 tests passed

Aggregate score: 83.90%
[Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%,
Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:
----------------------------------
2.5K Nov  5 08:26 BruteCollinearPoints.java
3.9K Nov  5 08:26 FastCollinearPoints.java
3.1K Nov  5 08:26 Point.java


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



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

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

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



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



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

BruteCollinearPoints:

FastCollinearPoints:

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



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



% spotbugs *.class
*-----------------------------------------------------------
M V EI_EXPOSE_REP EI: Returns a reference to the mutable object stored in
the instance variable 'segments', which exposes the internal representation
of the class 'BruteCollinearPoints'. Instead, create a defensive copy of the
object referenced by 'segments' and return the copy.  At
BruteCollinearPoints.java:[line 58]
L D FE_FLOATING_POINT_EQUALITY FE: Tests for exact floating-point equality.
Because floating-point calculations may involve rounding, the calculated
values may be imprecise.  At BruteCollinearPoints.java:[line 28]
M V EI_EXPOSE_REP EI: Returns a reference to the mutable object stored in
the instance variable 'segments', which exposes the internal representation
of the class 'FastCollinearPoints'. Instead, create a defensive copy of the
object referenced by 'segments' and return the copy.  At
FastCollinearPoints.java:[line 88]
```

```
L D FE_FLOATING_POINT_EQUALITY FE: Tests for exact floating-point equality.
Because floating-point calculations may involve rounding, the calculated
values may be imprecise.  At FastCollinearPoints.java:[line 33]
Warnings generated: 4


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


% pmd .
*-----------------------------------------------------------
BruteCollinearPoints.java:7: The private instance (or static) variable
'segments' can be made 'final'; it is initialized only in the declaration or
constructor. [ImmutableField]
BruteCollinearPoints.java:8: The private instance (or static) variable
'numberOfSegments' can be made 'final'; it is initialized only in the
declaration or constructor. [ImmutableField]
BruteCollinearPoints.java:58: Returning 'segments' may expose an internal
array. If so, return a defensive copy. [MethodReturnsInternalArray]
FastCollinearPoints.java:7: The private instance (or static) variable
'segments' can be made 'final'; it is initialized only in the declaration or
constructor. [ImmutableField]
FastCollinearPoints.java:8: The private instance (or static) variable
'numberOfSegments' can be made 'final'; it is initialized only in the
declaration or constructor. [ImmutableField]
FastCollinearPoints.java:88: Returning 'segments' may expose an internal
array. If so, return a defensive copy. [MethodReturnsInternalArray]
PMD ends with 6 warnings.


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


% checkstyle *.java
*-----------------------------------------------------------
[WARN] BruteCollinearPoints.java:15:11: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] BruteCollinearPoints.java:18:23: The local variable 'temp_segments'
must start with a lowercase letter and use camelCase. [LocalVariableName]
[WARN] BruteCollinearPoints.java:20:15: 'if' is not followed by whitespace.
[WhitespaceAfter]
```

```
[WARN] BruteCollinearPoints.java:23:19: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] BruteCollinearPoints.java:26:71: Do not put multiple statements on
the same line. [OneStatementPerLine]
[WARN] BruteCollinearPoints.java:28:23: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] BruteCollinearPoints.java:31:23: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] BruteCollinearPoints.java:33:88: ',' is not followed by whitespace.
[WhitespaceAfter]
[WARN] FastCollinearPoints.java:15:11: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] FastCollinearPoints.java:19:23: The local variable 'temp_segments'
must start with a lowercase letter and use camelCase. [LocalVariableName]
[WARN] FastCollinearPoints.java:23:15: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] FastCollinearPoints.java:29:25: The local variable 'temp_segment'
must start with a lowercase letter and use camelCase. [LocalVariableName]
[WARN] FastCollinearPoints.java:37:29: Control variable 'j' is modified
inside loop. [ModifiedControlVariable]
[WARN] Point.java:44:11: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Point.java:45:16: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Point.java:46:16: 'if' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Point.java:47:29: Typecast is not followed by whitespace.
[WhitespaceAfter]
[WARN] Point.java:60:28: ')' is preceded with whitespace. [ParenPad]
[WARN] Point.java:61:28: ')' is preceded with whitespace. [ParenPad]
[WARN] Point.java:77:19: The class 'slopeComparator' must start with an
uppercase letter and use CamelCase. [TypeName]
[WARN] Point.java:101:32: ',' is not followed by whitespace.
[WhitespaceAfter]
[WARN] Point.java:102:32: ',' is not followed by whitespace.
[WhitespaceAfter]
Checkstyle ends with 0 errors and 22 warnings.

% 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
    - failed after 15 operations involving BruteCollinearPoints
    - first and last call to segments() returned different arrays

    - sequence of operations was:
         BruteCollinearPoints collinear = new BruteCollinearPoints(points);
         collinear.segments()
         mutate points[] array that was passed to constructor
         collinear.numberOfSegments() -> 2
         mutate points[] array that was passed to constructor
         collinear.segments()
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         collinear.numberOfSegments() -> 2
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         collinear.numberOfSegments() -> 2
         collinear.segments()

    - failed on trial 1 of 100

  * equidistant.txt
    - failed after 8 operations involving BruteCollinearPoints
    - first and last call to segments() returned different arrays

    - sequence of operations was:
         BruteCollinearPoints collinear = new BruteCollinearPoints(points);
         collinear.numberOfSegments() -> 4
         collinear.numberOfSegments() -> 4
         collinear.segments()
         collinear.numberOfSegments() -> 4
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         collinear.segments()

    - failed on trial 1 of 100

==> FAILED

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
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
    10
     7187 23060
    14655 26617
    null
    13423 13690
    30413 23375
    22773 11171
     8185 26918
     3083  5040
    12865   349
     4992 13696

  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
    4
    20229 28803
    null
    14815 18863
```

```
     2744  7439

  * Point[] of length 3, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     3
      6293  2931
     18598  9134
     null

  * Point[] of length 2, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     2
     29772 21252
     null

  * Point[] of length 1, number of null entries = 1
==> FAILED

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: 15/17 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
    - number of entries in student   solution: 1
    - number of entries in reference solution: 4
    - 3 missing entries in student solution, including:
      '(30000, 0) -> (20000, 10000) -> (10000, 20000) -> (0, 30000)'


  * filename = input40.txt
    - number of entries in student   solution: 1
    - number of entries in reference solution: 4
    - 3 missing entries in student solution, including:
      '(2000, 29000) -> (4000, 29000) -> (22000, 29000) -> (28000, 29000)'


  * filename = input48.txt
    - number of entries in student   solution: 1
    - number of entries in reference solution: 6
    - 5 missing entries in student solution, including:
      '(1000, 26000) -> (9000, 26000) -> (11000, 26000) -> (18000, 26000)'



It is bad style to write code that depends on the particular format of
the output from the toString() method, especially if your reason for
doing so is to circumvent the public API (which intentionally does not
provide access to the x- and y-coordinates).

==> FAILED

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
    - failed after 16 operations involving FastCollinearPoints
    - first and last call to segments() returned different arrays
    - sequence of operations was:
         FastCollinearPoints collinear = new FastCollinearPoints(points);
         mutate points[] array that was passed to constructor
         mutate points[] array that was passed to constructor
         mutate points[] array that was passed to constructor
         collinear.numberOfSegments() -> 2
         collinear.segments()
         mutate array returned by last call to segments()
         collinear.numberOfSegments() -> 2
         mutate array returned by last call to segments()
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         collinear.numberOfSegments() -> 2
         mutate points[] array that was passed to constructor
         mutate array returned by last call to segments()
         collinear.numberOfSegments() -> 2
         collinear.segments()
    - failed on trial 1 of 100

  * equidistant.txt
    - failed after 12 operations involving FastCollinearPoints
    - first and last call to segments() returned different arrays
    - sequence of operations was:
         FastCollinearPoints collinear = new FastCollinearPoints(points);
         mutate points[] array that was passed to constructor
         mutate points[] array that was passed to constructor
         collinear.segments()
         collinear.segments()
         mutate points[] array that was passed to constructor
         mutate points[] array that was passed to constructor
         collinear.segments()
         mutate array returned by last call to segments()
         mutate array returned by last call to segments()
         mutate points[] array that was passed to constructor
         collinear.segments()
    - failed on trial 1 of 100

==> FAILED

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
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     10
      1514 18938
     14464  6591
     null
      5760 13908
     12450 30242
     12870 12594
     18427 18064
     14099  9679
     16208 18831
     26021 10414

  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     4
     30838 16338
     null
     11238 16511
     29158 17008

  * Point[] of length 3, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     3
     18446  8584
     17794 30793
     null

   * Point[] of length 2, number of null entries = 1
    - constructor throws wrong exception
    - constructor throws a java.lang.NullPointerException
    - constructor should throw a java.lang.IllegalArgumentException
     2
     26493 30417
     null

   * Point[] of length 1, number of null entries = 1
==> FAILED

Test 17: check that the constructor throws an exception if duplicate points
   * 50 points
    - failed on trial 1 of 5
    - constructor fails to throw a java.lang.IllegalArgumentException when
passed duplicate points


   * 25 points
    - failed on trial 1 of 10
    - constructor fails to throw a java.lang.IllegalArgumentException when
passed duplicate points


   * 5 points
    - failed on trial 1 of 100
    - constructor fails to throw a java.lang.IllegalArgumentException when
passed duplicate points

     5
     26048  8912
     23142 10708
     10207 21640
     10207 21640
      6118  9692

   * 4 points
```

```
    - failed on trial 1 of 100
    - constructor fails to throw a java.lang.IllegalArgumentException when
passed duplicate points

     4
     15768 14211
      4311 14308
      2639  6060
      4311 14308

  * 3 points
    - failed on trial 1 of 100
    - constructor fails to throw a java.lang.IllegalArgumentException when
passed duplicate points

     3
      9959 11619
     16094  2954
      9959 11619

  * 2 points
    - failed on trial 1 of 100
    - constructor throws wrong exception
    - constructor throws a java.lang.ArrayIndexOutOfBoundsException
    - constructor should throw a java.lang.IllegalArgumentException

     2
     20155  4652
     20155  4652

==> FAILED


Total: 17/21 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       1360          0          1360              120
=> passed   32   0.00      10912          0         10912              496
=> passed   64   0.01      87360          0         87360             2016
=> passed  128   0.02     699008          0        699008             8128
=> passed  256   0.05    5592320          0       5592320            32640
==> 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       1346          0          1346              137
=> passed    32   0.00      10780          0         10780              541
=> passed    64   0.00      86952          0         86952             2095
=> passed   128   0.01     697660          0        697660             8296
=> passed   256   0.04    5586290          0       5586290            32992
==> 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      18519         45102                0
=> passed   128   0.03      32512      87800        208112                0
=> passed   256   0.18     130560     415132        960824                0
=> passed   512   0.48     523276    1892233       4307742                0
=> passed  1024   1.64    2095116    8461585      19018286                0
=> passed  2048   8.29    8384602   37909882      84204366                0
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (84204366 / 19018286) = 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   0.01       8192       4764        17720              19443
=> passed  128   0.01      32768      17796        68360              93952
=> FAILED  256   0.02     131072      68717       268506             443677
(1.1x)
=> FAILED  512   0.19     524288     269399      1063086            2034329
(1.4x)
=> FAILED 1024   0.36    2097152    1065026      4227204            9155176
(1.6x)
=> FAILED 2048   0.70    8388608    4231214     16851036           40926359
(1.9x)
=> FAILED 4096   2.75   33554432   16859163     67272758          180396273
(2.1x)
==> 2/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (67272758 / 16851036) = 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   0.01       9384      17607        44598               4450
=> passed  128   0.15      37544      73089       183722              17498
=> passed  256   1.77     150184     282680       715544              58782
=> passed  512  16.76     600744    1106870      2814484             209278
Aborting: time limit of 10 seconds exceeded
```

```
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       9448      18420         46288               3844
=> passed   128   0.04      37864      87261        212386              16889
=> passed   256   0.63     151528     381485        914498              65343
=> passed   512  10.72     606184    1617427       3841038             239629
Aborting: time limit of 10 seconds exceeded

Total: 16/31 tests passed!


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