

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

ASSESSMENT SUMMARY

Compilation:	PASSED
API:	PASSED
SpotBugs:	PASSED
PMD:	PASSED
Checkstyle:	FAILED (0 errors, 1 warning)
Correctness:	41/41 tests passed
Memory:	1/1 tests passed
Timing:	39/41 tests passed
Aggregate score: 99.02%	
[Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20%]	

ASSESSMENT DETAILS

```
The following files were submitted:
-----
3.6K Mar 12 18:26 BruteCollinearPoints.java
6.6K Mar 12 18:26 FastCollinearPoints.java
4.4K Mar 12 18:26 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
*-----
[WARN] BruteCollinearPoints.java:46:30: Do not use the letter 'l' as a local variable name. It is hard to distinguish from the number '1'. [LocalVariable
Checkstyle ends with 0 errors and 1 warning.

% 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 = rsl423.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

	n	time	slopeTo()	compare()	slopeTo() + 2*compare()	compareTo()
=> passed	16	0.00	5460	0	5460	164
=> passed	32	0.00	107880	0	107880	615
=> passed	64	0.02	1906128	0	1906128	2325
=> passed	128	0.10	32004000	0	32004000	8868
=> passed	256	1.47	524377920	0	524377920	34375
=> 5/5 tests passed						

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

```

      n    time    slopeTo()    compare()    slopeTo()
-----+-----+-----+-----+-----+-----
      n    time    slopeTo()    compare()    + 2*compare()    compareTo()
-----+-----+-----+-----+-----+-----
=> passed    16    0.00         5460           0           5460           166
=> passed    32    0.00        107880          0          107880          618
=> passed    64    0.01        1906128         0          1906128         2323
=> passed   128    0.10       32004000         0          32004000         8870
=> passed   256    1.49      524377920         0          524377920        34378
==> 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

      n    time    slopeTo()    compare()    slopeTo()
-----+-----+-----+-----+-----+-----
      n    time    slopeTo()    compare()    + 2*compare()    compareTo()
-----+-----+-----+-----+-----+-----
=> passed    64    0.00         3965           7865          19695          2322
=> passed   128    0.00         16125          39353          94831          8862
=> passed   256    0.01         65021          187354         439729          34363
=> passed   512    0.04        261117          872018         2005153         134779
=> passed  1024    0.21       1046523         3955924         8958371         532770
=> passed  2048    0.65       4190180         17845989        39882158        2116105
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (39882158 / 8958371) = 2.15
=> passed

==> 7/7 tests passed

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

      n    time    slopeTo()    compare()    slopeTo()
-----+-----+-----+-----+-----+-----
      n    time    slopeTo()    compare()    + 2*compare()    compareTo()
-----+-----+-----+-----+-----+-----
=> passed    64    0.00         2073           1952           5977           2312
=> passed   128    0.00         8249           8000           24249          8871
=> passed   256    0.00        32889          32384           97657          34380
=> passed   512    0.00        131321         130304          391929         134794
=> passed  1024    0.01        524793         522752          1570297         532724
=> passed  2048    0.05       2098169         2094080         6286329         2116110
=> passed  4096    0.13       8390649         8382464        25155577         8430642
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (25155577 / 6286329) = 2.00
=> passed

==> 8/8 tests passed

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

      n    time    slopeTo()    compare()    slopeTo()
-----+-----+-----+-----+-----+-----
      n    time    slopeTo()    compare()    + 2*compare()    compareTo()
-----+-----+-----+-----+-----+-----
=> passed    64    0.00         3993           6975          17943           2317
=> passed   128    0.00        18505          26476          71457           8872
=> passed   256    0.01        95401          69827          235055          34367
=> passed   512    0.03       555369          245903          1047175          134776
=> passed  1024    0.16       3617513         927949          5473411          532768
=> passed  2048    0.12       25650665        3607526          32865717         2116131
=> passed  4096    0.65       192072681       14220881        220514443         8430611
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (220514443 / 32865717) = 2.75
=> FAILED (lg ratio is much greater than 2, your algorithm is probably cubic (or worse))

==> 7/8 tests passed

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

      n    time    slopeTo()    compare()    slopeTo()
-----+-----+-----+-----+-----+-----
      n    time    slopeTo()    compare()    + 2*compare()    compareTo()
-----+-----+-----+-----+-----+-----
=> passed    64    0.00         4791           7700          20191           2322
=> passed   128    0.00        23809          35865          95539           8882
=> passed   256    0.00       135093          127119          389331          34363
=> passed   512    0.01       865709          385257          1636223          134771
=> passed  1024    0.05      6079427          1438625          8956677          532741
=> passed  2048    0.27      45279395          5568216          56415827         2116035
=> passed  4096    1.77     348869919         21899885          392669689         8430563
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (392669689 / 56415827) = 2.80
=> FAILED (lg ratio is much greater than 2, your algorithm is probably cubic (or worse))

==> 7/8 tests passed

Total: 29/31 tests passed!

=====
```

