

*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:

-----  
3.1K Apr 5 05:49 BruteCollinearPoints.java  
8.4K Apr 5 05:49 FastCollinearPoints.java  
4.1K Apr 5 05:49 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 = horizontal15.txt
- \* filename = vertical15.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 = horizontal15.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 = vertical15.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:  $m \times n$  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 = horizontal15.txt
* filename = vertical15.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	3640	0	3640	120
=> passed	32	0.00	71920	0	71920	496
=> passed	64	0.02	1270752	0	1270752	2016
=> passed	128	0.02	21336000	0	21336000	8128
=> passed	256	1.23	349585298	0	349585298	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	3730	0	3730	142
=> passed	32	0.00	72334	0	72334	538
=> passed	64	0.01	1272940	0	1272940	2102
=> passed	128	0.11	21343806	0	21343806	8294
=> passed	256	1.59	349616400	0	349616400	32984

==> 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() + 2*compare()	compareTo()
=> passed	64	0.01	7503	8218	23939	10325
=> passed	128	0.01	31375	40124	111623	48716
=> passed	256	0.02	128271	189129	506529	222941
=> passed	512	0.08	518671	874563	2267797	1009007
=> passed	1024	0.35	2085903	3976027	10037957	4518587
=> passed	2048	1.25	8366095	17859490	44085075	20047983

==> 6/6 tests passed

$\lg \text{ratio}(\text{slopeTo}() + 2*\text{compare}()) = \lg (44085075 / 10037957) = 2.13$

=> 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() + 2*compare()	compareTo()
=> passed	64	0.00	9454	2013	13480	4331
=> passed	128	0.00	39374	8125	55624	16988
=> passed	256	0.00	160654	32637	225928	66995
=> passed	512	0.01	648974	130813	910600	265595
=> passed	1024	0.03	2608654	523773	3656200	1056544
=> passed	2048	0.09	10460174	2096125	14652424	4212231
=> passed	4096	0.28	41891854	8386557	58664968	16817214

==> 7/7 tests passed

$\lg \text{ratio}(\text{slopeTo}() + 2*\text{compare}()) = \lg (58664968 / 14652424) = 2.00$

=> 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() + 2*compare()	compareTo()
=> passed	64	0.00	8072	7204	22480	9766
=> passed	128	0.00	33880	27976	89832	40705

```

=> passed 256 0.01 138744 70690 280124 150463
=> passed 512 0.02 561464 247521 1056506 568755
=> passed 1024 0.06 2258872 931210 4121292 2203666
=> passed 2048 0.20 9061560 3614569 16290698 8680151
=> passed 4096 0.67 36298424 14235437 64769298 34406667
==> 7/7 tests passed

```

```

lg ratio(slopeTo() + 2*compare()) = lg (64769298 / 16290698) = 1.99
=> passed

```

```

==> 8/8 tests passed

```

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

	n	time	slopeTo()	compare()	slopeTo() + 2*compare()	compareTo()
=> passed	64	0.00	8032	7964	23960	10350
=> passed	128	0.00	33714	36286	106286	47357
=> passed	256	0.01	138102	134078	406258	197941
=> passed	512	0.02	558958	387647	1334252	776563
=> passed	1024	0.07	2249016	1443068	5135152	3022320
=> passed	2048	0.25	9022488	5577023	20176534	11903375
=> passed	4096	1.01	36142876	21917827	79978530	47183201

```

==> 7/7 tests passed

```

```

lg ratio(slopeTo() + 2*compare()) = lg (79978530 / 20176534) = 1.99
=> passed

```

```

==> 8/8 tests passed

```

Total: 31/31 tests passed!

```

=====

```