[陈](https://www.coursera.org/" \o "主页)



返回到第 3 周**课程**

[上一个](https://www.coursera.org/learn/algorithms-part1/quiz/LojjQ/interview-questions-optional)

[下一个](https://www.coursera.org/learn/algorithms-part1/supplement/efbDN/lecture-slides)

### Mergesort

##### [Overview](https://www.coursera.org/learn/algorithms-part1/supplement/mCXVI/overview)

##### [Lecture Slides](https://www.coursera.org/learn/algorithms-part1/supplement/4E9fa/lecture-slides)

##### [Mergesort](https://www.coursera.org/learn/algorithms-part1/lecture/ARWDq/mergesort)

[23 分](https://www.coursera.org/learn/algorithms-part1/lecture/ARWDq/mergesort)

##### [Bottom-up Mergesort](https://www.coursera.org/learn/algorithms-part1/lecture/PWNEl/bottom-up-mergesort)

[3 分](https://www.coursera.org/learn/algorithms-part1/lecture/PWNEl/bottom-up-mergesort)

##### [Sorting Complexity](https://www.coursera.org/learn/algorithms-part1/lecture/xAltF/sorting-complexity)

[9 分](https://www.coursera.org/learn/algorithms-part1/lecture/xAltF/sorting-complexity)

##### [Comparators](https://www.coursera.org/learn/algorithms-part1/lecture/9FYhS/comparators)

[6 分](https://www.coursera.org/learn/algorithms-part1/lecture/9FYhS/comparators)

##### [Stability](https://www.coursera.org/learn/algorithms-part1/lecture/pvvLZ/stability)

[5 分](https://www.coursera.org/learn/algorithms-part1/lecture/pvvLZ/stability)

##### ****[练习测验:](https://www.coursera.org/learn/algorithms-part1/quiz/LojjQ/interview-questions-optional)****

##### [Interview Questions (optional)](https://www.coursera.org/learn/algorithms-part1/quiz/LojjQ/interview-questions-optional)

[3 questions](https://www.coursera.org/learn/algorithms-part1/quiz/LojjQ/interview-questions-optional)

##### ****[编程作业:](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points)****

##### [Collinear Points](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points)

[5小时](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points)

### Quicksort

## 编程作业: Collinear Points

已通过 · 91/100 分

|  |  |
| --- | --- |
| **截止时间** | Pass this assignment by 二月 12, 11:59 晚上 PST |

1. [说明](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points)
2. [**我提交的作业**](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points/submission)
3. [讨论](https://www.coursera.org/learn/algorithms-part1/programming/prXiW/collinear-points/discussions)

  Create submission

### Your Submissions

**日期**

**分数**

**通过了吗？**

27 一月 2017 在 2:58 下午

91/100

是

Collinear Points

91/100

隐藏 评分反馈

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

ASSESSMENT SUMMARY

Compilation: PASSED

API: PASSED

Findbugs: FAILED (7 warnings)

Checkstyle: FAILED (10 warnings)

Correctness: 36/41 tests passed

Memory: 1/1 tests passed

Timing: 36/41 tests passed

Aggregate score: 90.24%

[Compilation: 5%, API: 5%, Findbugs: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:

----------------------------------

4.0K Jan 27 06:58 BruteCollinearPoints.java

4.9K Jan 27 06:58 FastCollinearPoints.java

4.8K Jan 27 06:58 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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% findbugs \*.class

\*-----------------------------------------------------------

M V EI\_EXPOSE\_REP EI: Returns a reference to the mutable object stored in the instance variable 'ls', which exposes the internal representation of the class 'BruteCollinearPoints'. Instead, create a defensive copy of the object referenced by 'ls' and return the copy. At BruteCollinearPoints.java:[line 80]

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 34]

L B NM\_CLASS\_NAMING\_CONVENTION Nm: The class name 'Point$slopeComparator' doesn't start with an upper-case letter. At Point.java:[lines 108-116]

M V EI\_EXPOSE\_REP EI: Returns a reference to the mutable object stored in the instance variable 'ls', which exposes the internal representation of the class 'FastCollinearPoints'. Instead, create a defensive copy of the object referenced by 'ls' and return the copy. At FastCollinearPoints.java:[line 132]

L D PZLA\_PREFER\_ZERO\_LENGTH\_ARRAYS PZLA: Should 'segments()' return a zero-length array instead of null? At FastCollinearPoints.java:[line 130]

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 51]

M P UUF\_UNUSED\_FIELD UuF: The instance (or static) variable 'slope' is never used. Consider removing it from the class. In Point.java

Warnings generated: 7

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

% checkstyle \*.java

\*-----------------------------------------------------------

Point.java:108:19: The class 'slopeComparator' must start with an uppercase letter and use CamelCase. [TypeName]

BruteCollinearPoints.java:5: Do not use .\* in import statements. [AvoidStarImport]

BruteCollinearPoints.java:31:16: 'for' is not followed by whitespace. [WhitespaceAfter]

FastCollinearPoints.java:1:1: Do not import 'java.lang.reflect.Array' in this course. [IllegalImport]

FastCollinearPoints.java:1:8: Unused import statement for 'java.lang.reflect.Array'. [UnusedImports]

FastCollinearPoints.java:7:8: Unused import statement for 'edu.princeton.cs.algs4.In'. [UnusedImports]

FastCollinearPoints.java:8:8: Unused import statement for 'edu.princeton.cs.algs4.StdOut'. [UnusedImports]

FastCollinearPoints.java:9:8: Unused import statement for 'edu.princeton.cs.algs4.StdDraw'. [UnusedImports]

FastCollinearPoints.java:86:12: 'for' is not followed by whitespace. [WhitespaceAfter]

FastCollinearPoints.java:99:11: '//' or '/\*' is not followed by whitespace. [IllegalTokenText]

Checkstyle ends with 10 errors.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 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 9 operations involving BruteCollinearPoints

- first and last call to segments() returned different arrays

- sequence of operations was:

BruteCollinearPoints collinear = new BruteCollinearPoints(points);

mutate points[] array that was passed to constructor

mutate points[] array that was passed to constructor

collinear.numberOfSegments() -> 2

mutate points[] array that was passed to constructor

collinear.segments()

collinear.segments()

mutate array returned by last call to segments()

collinear.segments()

- failed on trial 1 of 100

\* equidistant.txt

- failed after 9 operations involving BruteCollinearPoints

- first and last call to segments() returned different arrays

- sequence of operations was:

BruteCollinearPoints collinear = new BruteCollinearPoints(points);

collinear.segments()

collinear.numberOfSegments() -> 4

collinear.segments()

mutate points[] array that was passed to constructor

collinear.numberOfSegments() -> 4

mutate points[] array that was passed to constructor

mutate array returned by last call to segments()

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 exception either if argument to constructor is null

or if 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

- student solution has 26 non-null entries

- reference solution has 31 non-null entries

- 5 missing entries in student solution, including: (26000, 13000) -> (26000, 16000) -> (26000, 22000) -> (26000, 23000)

\* filename = input300.txt

\* filename = inarow.txt

==> FAILED

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

- student solution has 17 non-null entries

- reference solution has 38 non-null entries

- 21 missing entries in student solution, including: (2, 6) -> (6, 6) -> (8, 6) -> (9, 6)

- failed on trial 1 of 100

\* 90 random points in a 10-by-10 grid

- student solution has 30 non-null entries

- reference solution has 124 non-null entries

- 94 missing entries in student solution, including: (8, 6) -> (6, 7) -> (4, 8) -> (2, 9)

- failed on trial 1 of 50

\* 200 random points in a 50-by-50 grid

- student solution has 88 non-null entries

- reference solution has 197 non-null entries

- 109 missing entries in student solution, including: (2, 48) -> (7, 48) -> (8, 48) -> (17, 48) -> (25, 48)

- failed on trial 1 of 10

==> FAILED

Test 12: M\*N points on an M-by-N grid

\* 3-by-3 grid

\* 4-by-4 grid

- student solution has 5 non-null entries

- reference solution has 10 non-null entries

- 5 missing entries in student solution, including: (3, 0) -> (3, 1) -> (3, 2) -> (3, 3)

\* 5-by-5 grid

- student solution has 8 non-null entries

- reference solution has 16 non-null entries

- 8 missing entries in student solution, including: (0, 1) -> (1, 2) -> (2, 3) -> (3, 4)

\* 10-by-10 grid

- student solution has 35 non-null entries

- reference solution has 154 non-null entries

- 119 missing entries in student solution, including: (9, 6) -> (7, 7) -> (5, 8) -> (3, 9)

\* 20-by-20 grid

- student solution has 149 non-null entries

- reference solution has 2446 non-null entries

- 2297 missing entries in student solution, including: (19, 16) -> (17, 17) -> (15, 18) -> (13, 19)

\* 5-by-4 grid

- student solution has 6 non-null entries

- reference solution has 13 non-null entries

- 7 missing entries in student solution, including: (4, 0) -> (4, 1) -> (4, 2) -> (4, 3)

\* 6-by-4 grid

- student solution has 7 non-null entries

- reference solution has 16 non-null entries

- 9 missing entries in student solution, including: (5, 0) -> (5, 1) -> (5, 2) -> (5, 3)

\* 10-by-4 grid

- student solution has 11 non-null entries

- reference solution has 38 non-null entries

- 27 missing entries in student solution, including: (9, 0) -> (9, 1) -> (9, 2) -> (9, 3)

\* 15-by-4 grid

- student solution has 16 non-null entries

- reference solution has 79 non-null entries

- 63 missing entries in student solution, including: (14, 0) -> (14, 1) -> (14, 2) -> (14, 3)

\* 25-by-4 grid

- student solution has 26 non-null entries

- reference solution has 213 non-null entries

- 187 missing entries in student solution, including: (24, 0) -> (24, 1) -> (24, 2) -> (24, 3)

==> FAILED

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 15 operations involving FastCollinearPoints

- first and last call to segments() returned different arrays

- sequence of operations was:

FastCollinearPoints collinear = new FastCollinearPoints(points);

collinear.numberOfSegments() -> 2

collinear.numberOfSegments() -> 2

collinear.numberOfSegments() -> 2

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.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

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 6 operations involving FastCollinearPoints

- first and last call to segments() returned different arrays

- sequence of operations was:

FastCollinearPoints collinear = new FastCollinearPoints(points);

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 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 exception either if argument to constructor is null

or if 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: 17/21 tests passed!

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MEMORY

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Computing memory of Point

\*-----------------------------------------------------------

Running 1 total tests.

The maximum amount of memory per Point object is 32 bytes.

Student memory = 32 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 0.01 3640 0 3640 45

=> passed 32 0.00 71920 0 71920 123

=> passed 64 0.01 1270752 0 1270752 305

=> passed 128 0.02 21336000 0 21336000 743

=> passed 256 1.62 349585280 0 349585280 1733

==> 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 3746 0 3746 48

=> passed 32 0.00 72784 0 72784 121

=> passed 64 0.03 1273888 0 1273888 307

=> passed 128 0.10 21349412 0 21349412 747

=> passed 256 1.64 349638368 0 349638368 1714

==> 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 0.02 7936 18704 45344 19776

=> passed 128 0.02 32256 89059 210374 93440

=> passed 256 0.07 130048 415878 961804 440576

=> passed 512 0.24 522240 1890413 4303066 2040832

=> passed 1024 0.57 2093062 8530665 19154392 9165824

=> passed 2048 1.55 8380512 38090557 84561626 40894464

==> 6/6 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (84561626 / 19154392) = 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 8448 4764 17976 26199

=> passed 128 0.01 33280 17796 68872 118359

=> FAILED 256 0.02 132096 68717 269530 520439 (1.3x)

=> FAILED 512 0.10 526336 269399 1065134 2338647 (1.6x)

=> FAILED 1024 0.22 2101248 1065026 4231300 10296375 (1.8x)

=> FAILED 2048 0.97 8396800 4231214 16859228 45191703 (2.1x)

=> FAILED 4096 3.03 33570816 16859163 67289142 197527031 (2.3x)

==> 2/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (67289142 / 16859228) = 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 8484 14906 38296 20248

=> passed 128 0.01 33348 43854 121056 96826

=> passed 256 0.01 132228 149618 431464 445306

=> passed 512 0.03 526596 548156 1622908 2030074

=> passed 1024 0.18 2101764 2087496 6276756 9221882

=> passed 2048 0.73 8397828 8122445 24642718 40975610

=> passed 4096 3.66 33572868 31990953 97554774 180504826

==> 7/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (97554774 / 24642718) = 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 8564 18045 44654 19723

=> passed 128 0.00 33508 75863 185234 94524

=> passed 256 0.01 132548 232229 597006 445694

=> passed 512 0.05 527236 854545 2236326 2037502

=> passed 1024 0.18 2103044 3260991 8625026 9198846

=> passed 2048 0.79 8400388 12699218 33798824 40990974

=> passed 4096 3.49 33577988 50043244 133664476 180588798

==> 7/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (133664476 / 33798824) = 1.98

=> passed

==> 8/8 tests passed

Total: 26/31 tests passed!

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

26 一月 2017 在 4:57 下午

52/100

否

Collinear Points

52/100

显示 评分反馈

26 一月 2017 在 2:31 下午

59/100

否

26 一月 2017 在 2:24 下午

50/100

否

23 一月 2017 在 3:33 下午

59/100

否

23 一月 2017 在 3:26 下午

38/100

否