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

ASSESSMENT SUMMARY

Compilation: PASSED
API: PASSED

Spotbugs: PASSED
PMD: FAILED (2 warnings)
Checkstyle: FAILED (0 errors, 6 warnings)

Correctness: 31/35 tests passed
Memory: 16/16 tests passed
Timing: 42/42 tests passed

Aggregate score: 93.14%
[Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:

7.9K Oct 18 04:10 KdTree.java
2.3K Oct 18 04:10 PointSET.java

==
**
* COMPILING
==
**

% javac PointSET.java
*--

% javac KdTree.java
*--

==

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

KdTree:

==

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

% spotbugs *.class
*--

==

% pmd .

```
*-----------------------------------------------------------
KdTree.java:26: The private instance (or static) variable 'point' can be made
'final'; it is initialized only in the declaration or constructor.
[ImmutableField]
PointSET.java:12: The private instance (or static) variable 'points' can be
made 'final'; it is initialized only in the declaration or constructor.
[ImmutableField]
PMD ends with 2 warnings.


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


% checkstyle *.java
*-----------------------------------------------------------

% custom checkstyle checks for PointSET.java
*-----------------------------------------------------------
[WARN] PointSET.java:94:39: Do not call 'distanceTo()' in this program;
instead use 'distanceSquaredTo()'. [Performance]
[WARN] PointSET.java:94:61: Do not call 'distanceTo()' in this program;
instead use 'distanceSquaredTo()'. [Performance]
Checkstyle ends with 0 errors and 2 warnings.

% custom checkstyle checks for KdTree.java
*-----------------------------------------------------------
[WARN] KdTree.java:242:24: Do not call 'distanceTo()' in this program; instead
use 'distanceSquaredTo()'. [Performance]
[WARN] KdTree.java:242:52: Do not call 'distanceTo()' in this program; instead
use 'distanceSquaredTo()'. [Performance]
[WARN] KdTree.java:274:44: Do not call 'distanceTo()' in this program; instead
use 'distanceSquaredTo()'. [Performance]
[WARN] KdTree.java:274:72: Do not call 'distanceTo()' in this program; instead
use 'distanceSquaredTo()'. [Performance]
Checkstyle ends with 0 errors and 4 warnings.


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


********************************************************************************
**
*  TESTING CORRECTNESS
********************************************************************************
**

Testing correctness of PointSET
*-----------------------------------------------------------
Running 8 total tests.

A point in an m-by-m grid means that it is of the form (i/m, j/m),
where i and j are integers between 0 and m

Test 1: insert n random points; check size() and isEmpty() after each
insertion
        (size may be less than n because of duplicates)
  * 5 random points in a 1-by-1 grid
  * 50 random points in a 8-by-8 grid
  * 100 random points in a 16-by-16 grid
  * 1000 random points in a 128-by-128 grid
  * 5000 random points in a 1024-by-1024 grid
  * 50000 random points in a 65536-by-65536 grid
==> passed

Test 2: insert n random points; check contains() with random query points
  * 1 random points in a 1-by-1 grid
  * 10 random points in a 4-by-4 grid
  * 20 random points in a 8-by-8 grid
```

```
  * 10000 random points in a 128-by-128 grid
  * 100000 random points in a 1024-by-1024 grid
  * 100000 random points in a 65536-by-65536 grid
==> passed

Test 3: insert random points; check nearest() with random query points
  * 10 random points in a 4-by-4 grid
  * 15 random points in a 8-by-8 grid
  * 20 random points in a 16-by-16 grid
  * 100 random points in a 32-by-32 grid
  * 10000 random points in a 65536-by-65536 grid
==> passed

Test 4: insert random points; check range() with random query rectangles
  * 2 random points and random rectangles in a 2-by-2 grid
  * 10 random points and random rectangles in a 4-by-4 grid
  * 20 random points and random rectangles in a 8-by-8 grid
  * 100 random points and random rectangles in a 16-by-16 grid
  * 1000 random points and random rectangles in a 64-by-64 grid
  * 10000 random points and random rectangles in a 128-by-128 grid
==> passed

Test 5: call methods before inserting any points
 * size() and isEmpty()
 * contains()
 * nearest()
 * range()
==> passed

Test 6: call methods with null argument
  * insert()
    - throws wrong exception when calling insert() with a null first argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * contains()
    - throws wrong exception when calling contains() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * range()
    - throws wrong exception when calling range() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * nearest()
    - throws wrong exception when calling nearest() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

==> FAILED

Test 7: check intermixed sequence of calls to insert(), isEmpty(),
        size(), contains(), range(), and nearest() with
        probabilities (p1, p2, p3, p4, p5, p6, p7), respectively
  * 10000 calls with random points in a 1-by-1 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
  * 10000 calls with random points in a 16-by-16 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
  * 10000 calls with random points in a 128-by-128 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
  * 10000 calls with random points in a 1024-by-1024 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
  * 10000 calls with random points in a 8192-by-8192 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
  * 10000 calls with random points in a 65536-by-65536 grid
    and probabilities (0.3, 0.1, 0.1, 0.1, 0.2, 0.2)
==> passed
```

```
Test 8: check that two PointSET objects can be created at the same time
==> passed


Total: 7/8 tests passed!


================================================================
Testing correctness of KdTree
*----------------------------------------------------------
Running 27 total tests.

In the tests below, we consider three classes of points and rectangles.

  * Non-degenerate points: no two points (or rectangles) share either an
                           x-coordinate or a y-coordinate

  * Distinct points:       no two points (or rectangles) share both an
                           x-coordinate and a y-coordinate

  * General points:        no restrictions on the x-coordinates or y-
coordinates
                           of the points (or rectangles)

A point in an m-by-m grid means that it is of the form (i/m, j/m),
where i and j are integers between 0 and m (inclusive).

Test 1a: insert points from file; check size() and isEmpty() after each
insertion
  * input0.txt
  * input1.txt
  * input5.txt
  * input10.txt
==> passed

Test 1b: insert non-degenerate points; check size() and isEmpty() after each
insertion
  * 1 random non-degenerate points in a 1-by-1 grid
  * 5 random non-degenerate points in a 8-by-8 grid
  * 10 random non-degenerate points in a 16-by-16 grid
  * 50 random non-degenerate points in a 128-by-128 grid
  * 500 random non-degenerate points in a 1024-by-1024 grid
  * 50000 random non-degenerate points in a 65536-by-65536 grid
==> passed

Test 1c: insert distinct points; check size() and isEmpty() after each
insertion
  * 1 random distinct points in a 1-by-1 grid
  * 10 random distinct points in a 8-by-8 grid
  * 20 random distinct points in a 16-by-16 grid
  * 10000 random distinct points in a 128-by-128 grid
  * 100000 random distinct points in a 1024-by-1024 grid
  * 100000 random distinct points in a 65536-by-65536 grid
==> passed

Test 1d: insert general points; check size() and isEmpty() after each
insertion
  * 5 random general points in a 1-by-1 grid
  * 10 random general points in a 4-by-4 grid
  * 50 random general points in a 8-by-8 grid
  * 100000 random general points in a 16-by-16 grid
  * 100000 random general points in a 128-by-128 grid
  * 100000 random general points in a 1024-by-1024 grid
==> passed

Test 2a: insert points from file; check contains() with random query points
  * input0.txt
```

```
  * input1.txt
  * input5.txt
  * input10.txt
==> passed

Test 2b: insert non-degenerate points; check contains() with random query
points
  * 1 random non-degenerate points in a 1-by-1 grid
  * 5 random non-degenerate points in a 8-by-8 grid
  * 10 random non-degenerate points in a 16-by-16 grid
  * 20 random non-degenerate points in a 32-by-32 grid
  * 500 random non-degenerate points in a 1024-by-1024 grid
  * 10000 random non-degenerate points in a 65536-by-65536 grid
==> passed

Test 2c: insert distinct points; check contains() with random query points
  * 1 random distinct points in a 1-by-1 grid
  * 10 random distinct points in a 4-by-4 grid
  * 20 random distinct points in a 8-by-8 grid
  * 10000 random distinct points in a 128-by-128 grid
  * 100000 random distinct points in a 1024-by-1024 grid
  * 100000 random distinct points in a 65536-by-65536 grid
==> passed

Test 2d: insert general points; check contains() with random query points
  * 10000 random general points in a 1-by-1 grid
  * 10000 random general points in a 16-by-16 grid
  * 10000 random general points in a 128-by-128 grid
  * 10000 random general points in a 1024-by-1024 grid
==> passed

Test 3a: insert points from file; check range() with random query rectangles
  * input0.txt
  * input1.txt
  * input5.txt
  * input10.txt
==> passed

Test 3b: insert non-degenerate points; check range() with random query
rectangles
  * 1 random non-degenerate points and random rectangles in a 2-by-2 grid
  * 5 random non-degenerate points and random rectangles in a 8-by-8 grid
  * 10 random non-degenerate points and random rectangles in a 16-by-16 grid
  * 20 random non-degenerate points and random rectangles in a 32-by-32 grid
  * 500 random non-degenerate points and random rectangles in a 1024-by-1024
grid
  * 10000 random non-degenerate points and random rectangles in a 65536-by-
65536 grid
==> passed

Test 3c: insert distinct points; check range() with random query rectangles
  * 2 random distinct points and random rectangles in a 2-by-2 grid
  * 10 random distinct points and random rectangles in a 4-by-4 grid
  * 20 random distinct points and random rectangles in a 8-by-8 grid
  * 100 random distinct points and random rectangles in a 16-by-16 grid
  * 1000 random distinct points and random rectangles in a 64-by-64 grid
  * 10000 random distinct points and random rectangles in a 128-by-128 grid
==> passed

Test 3d: insert general points; check range() with random query rectangles
  * 5000 random general points and random rectangles in a 2-by-2 grid
  * 5000 random general points and random rectangles in a 16-by-16 grid
  * 5000 random general points and random rectangles in a 128-by-128 grid
  * 5000 random general points and random rectangles in a 1024-by-1024 grid
==> passed

Test 3e: insert random points; check range() with tiny rectangles
         enclosing each point
```

```
  * 5 tiny rectangles and 5 general points in a 2-by-2 grid
  * 10 tiny rectangles and 10 general points in a 4-by-4 grid
  * 20 tiny rectangles and 20 general points in a 8-by-8 grid
  * 5000 tiny rectangles and 5000 general points in a 128-by-128 grid
  * 5000 tiny rectangles and 5000 general points in a 1024-by-1024 grid
  * 5000 tiny rectangles and 5000 general points in a 65536-by-65536 grid
==> passed

Test 4a: insert points from file; check range() with random query rectangles
         and check traversal of kd-tree
  * input5.txt
  * input10.txt
==> passed

Test 4b: insert non-degenerate points; check range() with random query
rectangles
         and check traversal of kd-tree
  * 3 random non-degenerate points and 1000 random rectangles in a 4-by-4 grid
  * 6 random non-degenerate points and 1000 random rectangles in a 8-by-8 grid
  * 10 random non-degenerate points and 1000 random rectangles in a 16-by-16
grid
  * 20 random non-degenerate points and 1000 random rectangles in a 32-by-32
grid
  * 30 random non-degenerate points and 1000 random rectangles in a 64-by-64
grid
==> passed

Test 5a: insert points from file; check nearest() with random query points
  * input0.txt
  * input1.txt
  * input5.txt
  * input10.txt
==> passed

Test 5b: insert non-degenerate points; check nearest() with random query
points
  * 5 random non-degenerate points in a 8-by-8 grid
  * 10 random non-degenerate points in a 16-by-16 grid
  * 20 random non-degenerate points in a 32-by-32 grid
  * 30 random non-degenerate points in a 64-by-64 grid
  * 10000 random non-degenerate points in a 65536-by-65536 grid
==> passed

Test 5c: insert distinct points; check nearest() with random query points
  * 10 random distinct points in a 4-by-4 grid
  * 15 random distinct points in a 8-by-8 grid
  * 20 random distinct points in a 16-by-16 grid
  * 100 random distinct points in a 32-by-32 grid
  * 10000 random distinct points in a 65536-by-65536 grid
==> passed

Test 5d: insert general points; check nearest() with random query points
  * 10000 random general points in a 16-by-16 grid
  * 10000 random general points in a 128-by-128 grid
  * 10000 random general points in a 1024-by-1024 grid
==> passed

Test 6a: insert points from file; check nearest() with random query points
         and check traversal of kd-tree
  * input5.txt
    - student   nearest() = (0.7, 0.2)
    - reference nearest() = (0.7, 0.2)
    - performs incorrect traversal of kd-tree during call to nearest()
    - query point = (0.59, 0.16)
    - sequence of points inserted:
      A  0.7 0.2
      B  0.5 0.4
      C  0.2 0.3
```

```
      D  0.4 0.7
      E  0.9 0.6
    - student sequence of kd-tree nodes involved in calls to Point2D methods:
      A B C (0.2, 0.16) (0.59, 0.4) (0.7, 0.16) E (0.59, 0.6)
    - reference sequence of kd-tree nodes involved in calls to Point2D
methods:
      A B C E
    - failed on trial 1 of 1000

  * input10.txt
    - student   nearest() = (0.32, 0.708)
    - reference nearest() = (0.32, 0.708)
    - performs incorrect traversal of kd-tree during call to nearest()
    - query point = (0.34, 0.86)
    - sequence of points inserted:
      A  0.372 0.497
      B  0.564 0.413
      C  0.226 0.577
      D  0.144 0.179
      E  0.083 0.51
      F  0.32 0.708
      G  0.417 0.362
      H  0.862 0.825
      I  0.785 0.725
      J  0.499 0.208
    - student sequence of kd-tree nodes involved in calls to Point2D methods:
      A C F (0.32, 0.86) (0.34, 0.577) (0.372, 0.86) B H I (0.34, 0.725)
(0.862, 0.86) (0.34, 0.413)
    - reference sequence of kd-tree nodes involved in calls to Point2D
methods:
      A C F B H I
    - failed on trial 1 of 1000

==> FAILED

Test 6b: insert non-degenerate points; check nearest() with random query
points
         and check traversal of kd-tree
  * 5 random non-degenerate points in a 8-by-8 grid
    - student   nearest() = (0.75, 0.875)
    - reference nearest() = (0.75, 0.875)
    - performs incorrect traversal of kd-tree during call to nearest()
    - query point = (0.375, 1.0)
    - sequence of points inserted:
      A  0.75 0.875
      B  1.0 0.125
      C  0.875 0.625
      D  0.625 0.0
      E  0.0 0.75
    - student sequence of kd-tree nodes involved in calls to Point2D methods:
      A D E (0.0, 1.0) (0.375, 0.0) (0.75, 1.0) B C (0.875, 1.0) (0.375,
0.125)
    - reference sequence of kd-tree nodes involved in calls to Point2D
methods:
      A D E B C
    - failed on trial 1 of 1000

  * 10 random non-degenerate points in a 16-by-16 grid
    - student   nearest() = (0.6875, 0.9375)
    - reference nearest() = (0.6875, 0.9375)
    - performs incorrect traversal of kd-tree during call to nearest()
    - query point = (0.625, 0.8125)
    - sequence of points inserted:
      A  0.375 0.0
      B  0.3125 0.5625
      C  0.4375 0.125
      D  0.1875 0.375
      E  1.0 1.0
```

```
      F  0.5 0.1875
      G  0.5625 0.625
      H  0.75 0.0625
      I  0.0 0.3125
      J  0.6875 0.9375
    - student sequence of kd-tree nodes involved in calls to Point2D methods:
      A C E F G J (0.625, 0.9375) (0.5625, 0.8125) (0.625, 0.1875) (1.0,
0.8125) (0.625, 0.125) (0.375, 0.8125)
    - reference sequence of kd-tree nodes involved in calls to Point2D
methods:
      A C E F G J
    - failed on trial 1 of 1000

  * 20 random non-degenerate points in a 32-by-32 grid
    - student   nearest() = (0.625, 0.8125)
    - reference nearest() = (0.625, 0.8125)
    - performs incorrect traversal of kd-tree during call to nearest()
    - query point = (0.53125, 0.9375)
    - sequence of points inserted:
      A  0.3125 0.6875
      B  0.875 0.96875
      C  0.1875 0.03125
      D  0.21875 0.34375
      E  0.71875 0.875
      F  0.625 0.8125
      G  0.59375 0.71875
      H  0.9375 0.40625
      I  0.90625 0.21875
      J  0.5625 0.28125
      K  0.5 0.3125
      L  0.0 0.65625
      M  0.34375 0.90625
      N  0.03125 0.53125
      O  0.75 0.75
      P  0.0625 0.0
      Q  0.4375 0.59375
      R  1.0 0.5625
      S  0.09375 0.125
      T  0.28125 0.25
    - student sequence of kd-tree nodes involved in calls to Point2D methods:
      A B E F M (0.34375, 0.9375) (0.53125, 0.8125) G J K (0.5, 0.9375) Q
(0.53125, 0.59375) (0.53125, 0.28125) (0.59375, 0.9375) (0.71875, 0.9375)
(0.53125, 0.96875) (0.3125, 0.9375)
    - reference sequence of kd-tree nodes involved in calls to Point2D
methods:
      A B E F M G J K Q
    - failed on trial 1 of 1000

  * 30 random non-degenerate points in a 64-by-64 grid
    - student   nearest() = (0.171875, 0.625)
    - reference nearest() = (0.171875, 0.625)
    - performs incorrect traversal of kd-tree during call to nearest()
    - number of student   entries = 8
    - number of reference entries = 4
    - failed on trial 1 of 1000

  * 50 random non-degenerate points in a 128-by-128 grid
    - student   nearest() = (0.8515625, 0.0234375)
    - reference nearest() = (0.8515625, 0.0234375)
    - performs incorrect traversal of kd-tree during call to nearest()
    - number of student   entries = 12
    - number of reference entries = 6
    - failed on trial 1 of 1000

  * 1000 random non-degenerate points in a 2048-by-2048 grid
    - student   nearest() = (0.619140625, 0.91357421875)
    - reference nearest() = (0.619140625, 0.91357421875)
    - performs incorrect traversal of kd-tree during call to nearest()
```

```
    - number of student   entries = 32
    - number of reference entries = 16
    - entry 14 of the two sequences are not equal
    - student   entry 14 = (0.6416015625, 0.89111328125)
    - reference entry 14 = (0.66845703125, 0.953125)

    - failed on trial 1 of 1000

==> FAILED

Test 7: check with no points
  * size() and isEmpty()
  * contains()
  * nearest()
  * range()
==> passed

Test 8: check that the specified exception is thrown with null arguments
  * argument to insert() is null
    - throws wrong exception when calling add() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * argument to contains() is null
    - throws wrong exception when calling contains() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * argument to range() is null
    - throws wrong exception when calling range() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

  * argument to nearest() is null
    - throws wrong exception when calling nearest() with a null argument
    - throws a java.lang.NullPointerException
    - should throw a java.lang.IllegalArgumentException

==> FAILED

Test 9a: check intermixed sequence of calls to insert(), isEmpty(),
         size(), contains(), range(), and nearest() with probabilities
         (p1, p2, p3, p4, p5, p6), respectively
  * 20000 calls with non-degenerate points in a 1-by-1 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with non-degenerate points in a 16-by-16 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with non-degenerate points in a 128-by-128 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with non-degenerate points in a 1024-by-1024 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with non-degenerate points in a 8192-by-8192 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with non-degenerate points in a 65536-by-65536 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
==> passed

Test 9b: check intermixed sequence of calls to insert(), isEmpty(),
         size(), contains(), range(), and nearest() with probabilities
         (p1, p2, p3, p4, p5, p6), respectively
  * 20000 calls with distinct points in a 1-by-1 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with distinct points in a 16-by-16 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with distinct points in a 128-by-128 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with distinct points in a 1024-by-1024 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
```

```
  * 20000 calls with distinct points in a 8192-by-8192 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with distinct points in a 65536-by-65536 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
==> passed

Test 9c: check intermixed sequence of calls to insert(), isEmpty(),
         size(), contains(), range(), and nearest() with probabilities
         (p1, p2, p3, p4, p5, p6), respectively
  * 20000 calls with general points in a 1-by-1 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with general points in a 16-by-16 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with general points in a 128-by-128 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with general points in a 1024-by-1024 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with general points in a 8192-by-8192 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
  * 20000 calls with general points in a 65536-by-65536 grid
    and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)
==> passed

Test 10: insert n random points into two different KdTree objects;
        check that repeated calls to size(), contains(), range(),
        and nearest() with the same arguments yield same results
  * 10 random general points in a 4-by-4 grid
  * 20 random general points in a 8-by-8 grid
  * 100 random general points in a 128-by-128 grid
  * 1000 random general points in a 65536-by-65536 grid
==> passed


Total: 24/27 tests passed!


================================================================
********************************************************************************
**
*  MEMORY
********************************************************************************
**

Analyzing memory of Point2D
*-----------------------------------------------------------
Memory of Point2D object = 32 bytes

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



Analyzing memory of RectHV
*-----------------------------------------------------------
Memory of RectHV object = 48 bytes

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



Analyzing memory of PointSET
*-----------------------------------------------------------
Running 8 total tests.

Memory usage of a PointSET with n points (including Point2D and RectHV
objects).
Maximum allowed memory is 96n + 200 bytes.

                n       student (bytes)    reference (bytes)
```

```
-------------------------------------------------------------
=> passed        1          264                264
=> passed        2          360                360
=> passed        5          648                648
=> passed       10         1128               1128
=> passed       25         2568               2568
=> passed      100         9768               9768
=> passed      400        38568              38568
=> passed      800        76968              76968
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 96.00 n + 168.00  (R^2 = 1.000)
Estimated reference memory (bytes) = 96.00 n + 168.00  (R^2 = 1.000)

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



Analyzing memory of KdTree
*-----------------------------------------------------------
Running 8 total tests.

Memory usage of a KdTree with n points (including Point2D and RectHV objects).
Maximum allowed memory is 312n + 192 bytes.

                 n       student (bytes)    reference (bytes)
-------------------------------------------------------------
=> passed        1          112                160
=> passed        2          192                288
=> passed        5          432                672
=> passed       10          832               1312
=> passed       25         2032               3232
=> passed      100         8032              12832
=> passed      400        32032              51232
=> passed      800        64032             102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 80.00 n + 32.00  (R^2 = 1.000)
Estimated reference memory (bytes) = 128.00 n + 32.00  (R^2 = 1.000)

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



*****************************************************************************
**
*  TIMING
*****************************************************************************
**

Timing PointSET
*-----------------------------------------------------------
Running 14 total tests.


Inserting n points into a PointSET

               n      ops per second
----------------------------------------
=> passed   160000    1488246
=> passed   320000    1636654
=> passed   640000     960804
=> passed  1280000    1005915
==> 4/4 tests passed
```

```
Performing contains() queries after inserting n points into a PointSET

               n      ops per second
---------------------------------------
=> passed   160000     843657
=> passed   320000     731231
=> passed   640000     627793
=> passed  1280000     506746
==> 4/4 tests passed

Performing range() queries after inserting n points into a PointSET

               n      ops per second
---------------------------------------
=> passed    10000       4246
=> passed    20000       1511
=> passed    40000        634
==> 3/3 tests passed

Performing nearest() queries after inserting n points into a PointSET

               n      ops per second
---------------------------------------
=> passed    10000       4518
=> passed    20000       1503
=> passed    40000        647
==> 3/3 tests passed

Total: 14/14 tests passed!


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



Timing KdTree
*------------------------------------------------------------
Running 28 total tests.


Test 1a-d: Insert n points into a 2d tree. The table gives the average number
of calls
           to methods in RectHV and Point per call to insert().


Point2D
               n      ops per second       RectHV()           x()
y()             equals()
-------------------------------------------------------------------------------
----------------------------------
=> passed   160000     734593               0.0              44.3
42.3              21.6
=> passed   320000     824923               0.0              45.1
43.1              22.0
=> passed   640000     802346               0.0              48.1
46.1              23.5
=> passed  1280000     646620               0.0              52.3
50.3              25.6
==> 4/4 tests passed


Test 2a-h: Perform contains() queries after inserting n points into a 2d tree.
The table gives
           the average number of calls to methods in RectHV and Point per call
to contains().
```

```
Point2D
              n      ops per second       x()               y()
equals()
-------------------------------------------------------------------------------
-----------------
=> passed    10000    1239559              18.5              17.5
18.0
=> passed    20000    1260830              19.7              18.7
19.2
=> passed    40000    1106309              21.8              20.8
21.3
=> passed    80000     837153              22.0              21.0
21.5
=> passed   160000     808316              23.2              22.2
22.7
=> passed   320000     631779              25.0              24.0
24.5
=> passed   640000     553235              25.7              24.7
25.2
=> passed  1280000     445133              27.2              26.2
26.7
==> 8/8 tests passed


Test 3a-h: Perform range() queries after inserting n points into a 2d tree.
The table gives
           the average number of calls to methods in RectHV and Point per call
to range().

               n      ops per second       intersects()      contains()
x()               y()
-------------------------------------------------------------------------------
---------------------------------
=> passed    10000     533522               0.0              31.1
81.9              42.5
=> passed    20000     437039               0.0              32.6
85.9              48.8
=> passed    40000     481072               0.0              39.3
103.2              52.7
=> passed    80000     416745               0.0              40.7
106.5              55.0
=> passed   160000     314986               0.0              42.5
113.1              63.2
=> passed   320000     217990               0.0              40.2
105.7              55.7
=> passed   640000     143793               0.0              43.3
113.8              62.6
=> passed  1280000     178297               0.0              47.0
123.0              60.1
==> 8/8 tests passed


Test 4a-h: Perform nearest() queries after inserting n points into a 2d tree.
The table gives
           the average number of calls to methods in RectHV and Point per call
to nearest().

                                      Point2D                RectHV
               n      ops per second     distanceSquaredTo()
distanceSquaredTo()        x()               y()
-------------------------------------------------------------------------------
----------------------------------------
=> passed    10000   436093                  0.0                   0.0
50.4              50.0
=> passed    20000   552827                  0.0                   0.0
55.3              54.8
```

```
=> passed    40000   356621                 0.0                    0.0
64.7              64.3
=> passed    80000   448393                 0.0                    0.0
66.4              65.1
=> passed   160000   255282                 0.0                    0.0
71.4              71.0
=> passed   320000   410032                 0.0                    0.0
74.7              73.4
=> passed   640000   177708                 0.0                    0.0
77.5              76.3
=> passed  1280000   181350                 0.0                    0.0
85.6              86.0
==> 8/8 tests passed


Total: 28/28 tests passed!
```