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:  35/35 tests passed
Memory:       16/16 tests passed
Timing:       42/42 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:
----------------------------------
9.0K Sep  5 15:26 KdTree.java
2.9K Sep  5 15:26 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 .
*-----------------------------------------------------------


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

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


% custom checkstyle checks for PointSET.java
*-----------------------------------------------------------


% custom checkstyle checks for KdTree.java
*-----------------------------------------------------------


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


********************************************************************************
*  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()
  * contains()
  * range()
  * nearest()
==> passed

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: 8/8 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
  * input25.txt
  * input50.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 k-d tree
    * input5.txt
    * input10.txt
  ==> passed
```

```
Test 4b: insert non-degenerate points; check range() with random query rectangles
         and check traversal of k-d 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 k-d tree
  * input5.txt
  * input10.txt
==> passed

Test 6b: insert non-degenerate points; check nearest() with random query points
         and check traversal of k-d tree
  * 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
  * 50 random non-degenerate points in a 128-by-128 grid
  * 1000 random non-degenerate points in a 2048-by-2048 grid
==> passed

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
  * argument to contains() is null
  * argument to range() is null
  * argument to nearest() is null
==> passed

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: 27/27 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          240                264
=> passed       2          336                360
=> passed       5          624                648
=> passed      10         1104               1128
=> passed      25         2544               2568
=> passed     100         9744               9768
=> passed     400        38544              38568
=> passed     800        76944              76968
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 96.00 n + 144.00  (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          200                288
=> passed       5          464                672
=> passed      10          904               1312
=> passed      25         2224               3232
=> passed     100         8824              12832
=> passed     400        35224              51232
=> passed     800        70424             102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 88.00 n + 24.00  (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    1730492
=> passed   320000    1659618
=> passed   640000    1296333
=> passed  1280000     980163
==> 4/4 tests passed

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

              n      ops per second
---------------------------------------
=> passed   160000     629917
```

```
=> passed   320000     565407
=> passed   640000     525794
=> passed  1280000     509003
==> 4/4 tests passed

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

               n      ops per second
---------------------------------------
=> passed    10000       5195
=> passed    20000       1747
=> passed    40000        781
==> 3/3 tests passed

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

               n      ops per second
---------------------------------------
=> passed    10000       8175
=> passed    20000       2042
=> passed    40000        876
==> 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    1257939              0.0              43.3             43.3               0.0
=> passed   320000    1268279              0.0              44.1             44.1               0.0
=> passed   640000     944869              0.0              47.1             47.1               0.0
=> passed  1280000     714035              0.0              51.3             51.3               0.0
==> 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     878503             36.0             36.0   0.0
=> passed    20000     860492             38.3             38.3   0.0
=> passed    40000     773886             42.6             42.6   0.0
=> passed    80000     714210             43.0             43.0   0.0
=> passed   160000     589965             45.5             45.5   0.0
=> passed   320000     500693             49.1             49.1   0.0
=> passed   640000     424333             50.4             50.4   0.0
=> passed  1280000     394375             53.4             53.4   0.0
==> 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     314268             62.3              31.1             81.3             43.2
=> passed    20000     332602             65.1              32.6             85.9             48.8
=> passed    40000     265001             78.6              39.3            102.4             53.4
=> passed    80000     234657             81.3              40.7            105.9             55.6
=> passed   160000     202704             85.0              42.5            113.4             62.9
=> passed   320000     184202             80.5              40.2            105.4             56.0
=> passed   640000     140160             86.6              43.3            113.9             62.4
=> passed  1280000     120854             94.1              47.0            121.9             61.2
```

```
==> 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   317628                91.0                   58.0                   226.8             195.3
=> passed    20000   301121               100.2                   63.7                   248.7             215.1
=> passed    40000   250056               117.9                   75.5                   293.9             253.5
=> passed    80000   238510               120.5                   76.6                   296.5             263.1
=> passed   160000   206781               130.8                   83.2                   322.2             280.7
=> passed   320000   171728               136.5                   86.3                   335.0             296.4
=> passed   640000   144494               141.5                   90.3                   349.6             308.8
=> passed  1280000   115953               158.5                  100.9                   393.0             336.6
==> 8/8 tests passed



Total: 28/28 tests passed!


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