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

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          PASSED

SpotBugs:     PASSED
PMD:          FAILED (1 warning)
Checkstyle:   FAILED (0 errors, 2 warnings)

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:
----------------------------------
8.4K Aug  7 01:03 KdTree.java
2.2K Aug  7 01:03 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:228: Avoid unused private instance (or static) variables, such as 'a'. [UnusedPrivateField]
PMD ends with 1 warning.
```

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


% checkstyle *.java
*-----------------------------------------------------------
[WARN] KdTree.java:228:34: The constant 'a' must be ALL_UPPERCASE, with words separated by underscores. [ConstantName]
Checkstyle ends with 0 errors and 1 warning.

% custom checkstyle checks for PointSET.java
*-----------------------------------------------------------
[WARN] PointSET.java:40:30: The numeric literal '0.01' appears to be unnecessary. [NumericLiteral]
Checkstyle ends with 0 errors and 1 warning.

% 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
==> 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
  * input10.txt
==> passed

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
  * 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          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          136                160
=> passed        2          272                288
=> passed        5          680                672
=> passed       10         1360               1312
=> passed       25         3400               3232
=> passed      100        13600              12832
=> passed      400        54400              51232
=> passed      800       108800             102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 136.00 n + -0.00  (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    1993380
=> passed   320000    1994215
=> passed   640000    1704140
=> passed  1280000    1225963
==> 4/4 tests passed

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

```
               n      ops per second
----------------------------------------
=> passed   160000     701115
=> passed   320000     638736
=> passed   640000     640796
=> passed  1280000     528163
==> 4/4 tests passed

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

               n      ops per second
----------------------------------------
=> passed    10000       4881
=> passed    20000       1727
=> passed    40000        746
==> 3/3 tests passed

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

               n      ops per second
----------------------------------------
=> passed    10000       6449
=> passed    20000       2122
=> passed    40000        871
==> 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    1142895               1.0              22.6              21.6              21.6
=> passed   320000    1410044               1.0              23.0              22.0              22.0
=> passed   640000    1137495               1.0              24.5              23.5              23.5
=> passed  1280000     859062               1.0              26.6              25.6              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     870644              46.1              38.2              18.0
=> passed    20000     884223              50.9              41.2              19.2
=> passed    40000     809363              55.9              45.8              21.3
=> passed    80000     701103              56.5              46.1              21.5
=> passed   160000     613453              58.1              47.9              22.7
=> passed   320000     527647              64.9              53.0              24.5
=> passed   640000     469359              66.1              54.4              25.2
=> passed  1280000     394907              70.4              57.5              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     545296              49.4              31.1              50.1              12.1
=> passed    20000     488310              51.7              32.6              53.3              16.2
=> passed    40000     426541              63.9              39.3              63.1              14.1
=> passed    80000     364380              66.1              40.7              65.2              14.9
=> passed   160000     313971              69.0              42.5              70.9              20.4
=> passed   320000     259338              66.0              40.2              65.2              15.7
=> passed   640000     199791              71.0              43.3              70.7              19.2
=> passed  1280000     192354              77.7              47.0              74.8              14.2
==> 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   485405                158.3                     40.5                   88.6              87.5
=> passed    20000   499097                174.4                     44.8                   97.7              97.0
=> passed    40000   428070                205.7                     53.1                  116.9             114.7
=> passed    80000   385724                210.3                     54.4                  117.4             119.0
=> passed   160000   327496                228.5                     59.3                  129.7             129.0
=> passed   320000   258849                238.5                     62.1                  135.0             135.1
=> passed   640000   208986                247.6                     64.4                  139.4             140.2
=> passed  1280000   194174                277.5                     72.4                  158.4             155.7
==> 8/8 tests passed


Total: 28/28 tests passed!

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