Programming Assignment 5: Kd-Trees | kdtree.zip Help Center

Submission	
Submission time	Fri-23-Oct 15:34:55
Raw Score	100.00 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  21/21 tests passed
Memory:       8/8 tests passed
Timing:       41/41 tests passed

Aggregate score: 100.00% [Correctness: 65%, Memory: 10%, Timing: 25
%, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.7K Oct 23 22:35 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 22:35 PointSET.java
-rw-r--r-- 1 3.0K Oct 23 22:35 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********
```

```
% javac PointSET.java
*-----------------------------------------------------------

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

% javac KdTree.java
*-----------------------------------------------------------

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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
================================================================


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

KdTree:

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


**************************************************************************
**********
*  correctness
**************************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
```

```
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
```

```
        p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3=0, p4, p5, and p6, respectively
        (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.
```

```
  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
```

```
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
```

```
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
==> passed

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
```

```
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
==> passed


Total: 13/13 tests passed!

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

****************************************************************
**********
*  memory
****************************************************************
**********

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



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



Computing memory of KdTree
*-----------------------------------------------------------
Running 8 total tests.
```

```
Memory usage of a KdTree with N points (including Point2D and RectH
V objects).
Maximum allowed memory is 312N + 192 bytes.

                 N       student (bytes)    reference (bytes)
--------------------------------------------------------------
=> passed        1           168                 160
=> passed        2           312                 288
=> passed        5           744                 672
=> passed       10          1464                1312
=> passed       25          3624                3232
=> passed      100         14424               12832
=> passed      400         57624               51232
=> passed      800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********

Timing PointSET
*----------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     919620
=> passed   320000    1011152
=> passed   640000     682044
```

```
=> passed  1280000     567393
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     510838
=> passed    20000     529082
=> passed    40000     471348
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2387
=> passed    20000       1151
=> passed    40000        524
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2776
=> passed    20000       1274
=> passed    40000        575
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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

```
Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-------------------------------------------------------------------
-------------------------------------------
=> passed   160000     602605               1.0              22.6
21.6              21.6
=> passed   320000     713042               1.0              23.0
22.0              22.0
=> passed   640000     562774               1.0              24.5
23.5              23.5
=> passed  1280000     460695               1.0              26.6
25.6              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()              y()
equals()
-------------------------------------------------------------------
----------------------------
=> passed    10000     500232               18.5             17.5
18.0
=> passed    20000     507286               19.7             18.7
19.2
=> passed    40000     473583               21.8             20.8
21.3
=> passed    80000     461327               22.0             21.0
21.5
=> passed   160000     370897               23.2             22.2
22.7
=> passed   320000     290211               25.0             24.0
24.5
=> passed   640000     336160               25.7             24.7
```

```
25.2
=> passed  1280000     307228              27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-----------------------------------------------------------------
-----------------------------------------
=> passed    10000     289784               0.0               31.1
81.9              42.5
=> passed    20000     294184               0.0               32.6
85.9              48.8
=> passed    40000     314277               0.0               39.3
103.2              52.7
=> passed    80000     229180               0.0               40.7
106.5              55.0
=> passed   160000     184925               0.0               42.5
113.1              63.2
=> passed   320000     197677               0.0               40.2
105.7              55.7
=> passed   640000     134141               0.0               43.3
113.8              62.6
=> passed  1280000     144969               0.0               47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-----------------------------------------------------------------
-----------------------------------------------
=> passed    10000   298100                  52.0
```

```
20.9                    73.0              72.1
=> passed    20000   288027                 57.3
23.2                    80.6              79.9
=> passed    40000   251119                 67.7
27.6                    96.6              94.8
=> passed    80000   233012                 69.0
28.3                    97.4              97.4
=> passed   160000   180868                 74.9
30.9                   107.1             106.6
=> passed   320000   145681                 77.7
32.4                   111.8             110.4
=> passed   640000   125802                 81.2
33.6                   115.7             115.1
=> passed  1280000   115649                 90.9
37.9                   130.1             129.2
==> 8/8 tests passed

Total: 28/28 tests passed!

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

Submission	
Submission time	Fri-23-Oct 15:23:08
Raw Score	100.00 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  21/21 tests passed
Memory:       8/8 tests passed
Timing:       41/41 tests passed
```

```
Aggregate score: 100.00% [Correctness: 65%, Memory: 10%, Timing: 25
%, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.6K Oct 23 22:23 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 22:23 PointSET.java
-rw-r--r-- 1 3.0K Oct 23 22:23 studentSubmission.zip


================================================================
===========
*  compiling
================================================================
===========


% javac PointSET.java
*------------------------------------------------------------

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

% javac KdTree.java
*------------------------------------------------------------

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




% checkstyle *.java readme.txt
*------------------------------------------------------------
================================================================


% findbugs *.class
*------------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L4 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 216]
Warnings generated: 1
```

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


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

KdTree:

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


*****************************************************************
**********
*  correctness
*****************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
```

```
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
```

```
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed
```

```
Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
```

```
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
```

```
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
==> passed

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
==> passed


Total: 13/13 tests passed!

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

****************************************************************
```

```
***********
*  memory
*************************************************************************
***********

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




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




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

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

                N       student (bytes)    reference (bytes)
------------------------------------------------------------
=> passed       1           168                 160
=> passed       2           312                 288
=> passed       5           744                 672
=> passed      10          1464                1312
=> passed      25          3624                3232
=> passed     100         14424               12832
=> passed     400         57624               51232
=> passed     800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
```

```
)
================================================================



**************************************************************************
***********
*  timing
**************************************************************************
***********

Timing PointSET
*------------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     959973
=> passed   320000    1124733
=> passed   640000     859036
=> passed  1280000     686633
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     526575
=> passed    20000     550554
=> passed    40000     521257
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2366
=> passed    20000       1140
=> passed    40000        518
```

```
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2810
=> passed    20000       1305
=> passed    40000        583
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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

Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
=> passed   160000     631109               1.0              22.6
21.6             21.6
=> passed   320000     690872               1.0              23.0
22.0             22.0
=> passed   640000     593336               1.0              24.5
23.5             23.5
=> passed  1280000     465718               1.0              26.6
25.6             25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
```

```
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()               y()
equals()
-------------------------------------------------------------------
---------------------------
=> passed    10000     492719              18.5              17.5
18.0
=> passed    20000     498015              19.7              18.7
19.2
=> passed    40000     473300              21.8              20.8
21.3
=> passed    80000     442976              22.0              21.0
21.5
=> passed   160000     403214              23.2              22.2
22.7
=> passed   320000     335512              25.0              24.0
24.5
=> passed   640000     339438              25.7              24.7
25.2
=> passed  1280000     303934              27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-------------------------------------------
=> passed    10000     287559               0.0              31.1
81.9              42.5
=> passed    20000     291558               0.0              32.6
85.9              48.8
=> passed    40000     288060               0.0              39.3
103.2              52.7
=> passed    80000     241664               0.0              40.7
```

```
106.5              55.0
=> passed   160000     188672              0.0              42.5
113.1              63.2
=> passed   320000     150897              0.0              40.2
105.7              55.7
=> passed   640000     125001              0.0              43.3
113.8              62.6
=> passed  1280000     106554              0.0              47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-----------------------------------------------------------------
-------------------------------------------------
=> passed    10000   284426                 52.0
20.9                   73.0              72.1
=> passed    20000   233658                 57.3
23.2                   80.6              79.9
=> passed    40000   201419                 67.7
27.6                   96.6              94.8
=> passed    80000   188363                 69.0
28.3                   97.4              97.4
=> passed   160000   144073                 74.9
30.9                  107.1             106.6
=> passed   320000   133150                 77.7
32.4                  111.8             110.4
=> passed   640000   110366                 81.2
33.6                  115.7             115.1
=> passed  1280000    99487                 90.9
37.9                  130.1             129.2
==> 8/8 tests passed

Total: 28/28 tests passed!

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

Submission	
Submission time	Fri-23-Oct 14:32:36
Raw Score	96.90 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  20/21 tests passed
Memory:       8/8 tests passed
Timing:       41/41 tests passed

Aggregate score: 96.90% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.5K Oct 23 21:33 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 21:33 PointSET.java
-rw-r--r-- 1 3.0K Oct 23 21:33 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********
```

```
% javac PointSET.java
*-----------------------------------------------------------

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

% javac KdTree.java
*-----------------------------------------------------------

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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L4 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 216]
Warnings generated: 1
================================================================


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

KdTree:

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


**************************************************************
***********
*  correctness
**************************************************************
***********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.
```

```
Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
```

```
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
```

```
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
```

```
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed
```

```
Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
==> passed
```

```
Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:225)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:225)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:697)
     TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 12/13 tests passed!

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

*****************************************************************
**********
*  memory
*****************************************************************
**********

Computing memory of Point2D
*-----------------------------------------------------------
```

```
Memory of Point2D object = 32 bytes
================================================================



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



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

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

                N       student (bytes)    reference (bytes)
----------------------------------------------------------------
=> passed       1           168                 160
=> passed       2           312                 288
=> passed       5           744                 672
=> passed      10          1464                1312
=> passed      25          3624                3232
=> passed     100         14424               12832
=> passed     400         57624               51232
=> passed     800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
================================================================



****************************************************************
**********
```

```
*  timing
*************************************************************
**********

Timing PointSET
*------------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     947584
=> passed   320000    1006572
=> passed   640000     695066
=> passed  1280000     595101
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     526239
=> passed    20000     541728
=> passed    40000     485915
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2366
=> passed    20000       1141
=> passed    40000        523
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
```

```
=> passed    10000       2793
=> passed    20000       1291
=> passed    40000        540
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
=> passed   160000     585146               1.0              22.6
21.6             21.6
=> passed   320000     651389               1.0              23.0
22.0             22.0
=> passed   640000     532332               1.0              24.5
23.5             23.5
=> passed  1280000     402147               1.0              26.6
25.6             25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()                y()
```

```
equals()
-----------------------------------------------------------------
----------------------------
=> passed    10000     473540              18.5               17.5
18.0
=> passed    20000     469778              19.7               18.7
19.2
=> passed    40000     410235              21.8               20.8
21.3
=> passed    80000     368436              22.0               21.0
21.5
=> passed   160000     318146              23.2               22.2
22.7
=> passed   320000     314656              25.0               24.0
24.5
=> passed   640000     304282              25.7               24.7
25.2
=> passed  1280000     302067              27.2               26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-----------------------------------------------------------------
-----------------------------------------
=> passed    10000     290778               0.0               31.1
81.9              42.5
=> passed    20000     294951               0.0               32.6
85.9              48.8
=> passed    40000     289382               0.0               39.3
103.2             52.7
=> passed    80000     231365               0.0               40.7
106.5             55.0
=> passed   160000     180139               0.0               42.5
113.1             63.2
=> passed   320000     168391               0.0               40.2
105.7             55.7
=> passed   640000     132509               0.0               43.3
113.8             62.6
```

```
=> passed  1280000     115743               0.0               47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()              y()
-------------------------------------------------------------------
-------------------------------------------------
=> passed    10000   300186                52.0
20.9                   73.0             72.1
=> passed    20000   289754                57.3
23.2                   80.6             79.9
=> passed    40000   247396                67.7
27.6                   96.6             94.8
=> passed    80000   246986                69.0
28.3                   97.4             97.4
=> passed   160000   181559                74.9
30.9                   107.1            106.6
=> passed   320000   145918                77.7
32.4                   111.8            110.4
=> passed   640000   115241                81.2
33.6                   115.7            115.1
=> passed  1280000    97985                90.9
37.9                   130.1            129.2
==> 8/8 tests passed

Total: 28/28 tests passed!

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

Submission

Submission	Fri-23-Oct 13:58:16

time	
Raw Score	70.55 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  17/21 tests passed
Memory:       8/8 tests passed
Timing:       13/41 tests passed

Aggregate score: 70.55% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.6K Oct 23 21:14 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 21:14 PointSET.java
-rw-r--r-- 1 3.0K Oct 23 21:14 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********


% javac PointSET.java
*-----------------------------------------------------


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


% javac KdTree.java
```

```
*-----------------------------------------------------------

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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L4 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 218]
Warnings generated: 1
================================================================


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

KdTree:

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


**********************************************************************
**********
*  correctness
**********************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
```

```
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
```

```
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3=0, p4, p5, and p6, respectively
        (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
```

```
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
```

```
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.31978, 0.14682)
      -  reference nearest()    = (0.43796, 0.74664)
      -  student   distanceTo() = 0.6128157586909788
      -  reference distanceTo() = 0.001484082207965573
  *  100000 random general points in 10000-by-10000 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.6399, 0.7646)
      -  reference nearest()    = (0.4889, 0.3254)
      -  student   distanceTo() = 0.46558884222025765
      -  reference distanceTo() = 0.0022022271554554559
  *  100000 random general points in 1000-by-1000 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.176, 0.346)
      -  reference nearest()    = (0.166, 0.23)
      -  student   distanceTo() = 0.11543396380615192
      -  reference distanceTo() = 0.0010000000000000009
  *  100000 random general points in 100-by-100 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.3, 0.59)
      -  reference nearest()    = (0.34, 0.79)
      -  student   distanceTo() = 0.20396078054371147
      -  reference distanceTo() = 0.0
  *  100000 random general points in 10-by-10 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.9, 0.4)
      -  reference nearest()    = (0.7, 0.3)
      -  student   distanceTo() = 0.22360679774997905
      -  reference distanceTo() = 0.0
==> FAILED

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.011, 0.389)
      -  reference nearest()    = (0.091, 0.495)
      -  student   distanceTo() = 0.13020752666641679
```

```
      -  reference distanceTo() = 0.0031622776601683824
  *  10000 random general points in 100-by-100 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.9, 0.71)
      -  reference nearest()    = (0.98, 0.31)
      -  student   distanceTo() = 0.4079215610874227
      -  reference distanceTo() = 0.0
  *  10000 random general points in 10-by-10 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.6, 0.7)
      -  reference nearest()    = (0.7, 0.7)
      -  student   distanceTo() = 0.09999999999999998
      -  reference distanceTo() = 0.0
==> FAILED

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
      -  failed on trial 12 of 20000
      -  student   nearest()    = (0.80539, 0.44889)
      -  reference nearest()    = (0.31384, 0.38097)
      -  student   distanceTo() = 0.43889085898432656
      -  reference distanceTo() = 0.15571361661717328
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
      -  failed on trial 25 of 20000
      -  student   nearest()    = (0.3933, 0.3941)
      -  reference nearest()    = (0.8787, 0.0303)
      -  student   distanceTo() = 0.63560662362816 83
      -  reference distanceTo() = 0.12402781945999047
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
      -  failed on trial 40 of 20000
      -  student   nearest()    = (0.116, 0.769)
      -  reference nearest()    = (0.379, 0.215)
      -  student   distanceTo() = 0.8687951427120205
      -  reference distanceTo() = 0.30572209602840295
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
      -  failed on trial 21 of 20000
      -  student   nearest()    = (0.6, 0.91)
```

```
      -  reference nearest()    = (0.24, 0.97)
      -  student   distanceTo() = 0.34014702703389893
      -  reference distanceTo() = 0.05385164807134499
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
      -  failed on trial 17 of 20000
      -  student   nearest()    = (0.6, 0.8)
      -  reference nearest()    = (0.8, 0.8)
      -  student   distanceTo() = 0.360555127546399
      -  reference distanceTo() = 0.30000000000000004
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
      -  failed on trial 98 of 20000
      -  student   nearest()    = (0.0, 0.0)
      -  reference nearest()    = (0.0, 1.0)
      -  student   distanceTo() = 1.0
      -  reference distanceTo() = 0.0
==> FAILED

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:227)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:227)
```

```
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:697)
     TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 9/13 tests passed!

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

*****************************************************************
**********
*  memory
*****************************************************************
**********

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


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



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

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

                N       student (bytes)    reference (bytes)
---------------------------------------------------------------
=> passed       1          168                160
=> passed       2          312                288
=> passed       5          744                672
```

```
=> passed       10          1464                1312
=> passed       25          3624                3232
=> passed      100         14424               12832
=> passed      400         57624               51232
=> passed      800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
================================================================



*****************************************************************
***********
*  timing
*****************************************************************
***********

Timing PointSET
*----------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
---------------------------------------
=> passed   160000     792239
=> passed   320000     790139
=> passed   640000     608752
=> passed  1280000     631484
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
---------------------------------------
=> passed    10000     500357
```

```
=> passed    20000     522738
=> passed    40000     492612
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2382
=> passed    20000       1127
=> passed    40000        513
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2783
=> passed    20000       1315
=> passed    40000        577
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
```

```
-----------------------------------------------
=> passed   160000     600274               1.0              22.6
21.6              21.6
=> passed   320000     702514               1.0              23.0
22.0              22.0
=> passed   640000     561364               1.0              24.5
23.5              23.5
=> passed  1280000     460418               1.0              26.6
25.6              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()              y()
equals()
-----------------------------------------------------------------
----------------------------
=> passed    10000     509220               18.5             17.5
18.0
=> passed    20000     516603               19.7             18.7
19.2
=> passed    40000     477897               21.8             20.8
21.3
=> passed    80000     472510               22.0             21.0
21.5
=> passed   160000     407754               23.2             22.2
22.7
=> passed   320000     342399               25.0             24.0
24.5
=> passed   640000     351629               25.7             24.7
25.2
=> passed  1280000     315621               27.2             26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-----------------------------------------
=> passed    10000     294290               0.0              31.1
81.9              42.5
=> passed    20000     301981               0.0              32.6
85.9              48.8
=> passed    40000     265049               0.0              39.3
103.2             52.7
=> passed    80000     289842               0.0              40.7
106.5             55.0
=> passed   160000     192030               0.0              42.5
113.1             63.2
=> passed   320000     165947               0.0              40.2
105.7             55.7
=> passed   640000     123675               0.0              43.3
113.8             62.6
=> passed  1280000     144698               0.0              47.0
123.0             60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                          Point2D                Re
ctHV
              N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-------------------------------------------------------------------
-----------------------------------------------
=> passed    10000    56170                488.3
174.0                  633.7             627.3
=> FAILED    20000    32958                792.9   (1.3x)
283.7                  1054.8   (1.3x)   1034.2   (1.3x)
=> FAILED    40000    21164   (0.7x)       1116.5   (1.9x)
397.5   (1.3x)         1503.0   (1.9x)   1465.3   (1.8x)
=> FAILED    80000    14170   (0.5x)       1587.9   (2.6x)
570.7   (1.9x)         2075.3   (2.6x)   2128.7   (2.7x)
=> FAILED   160000     8693   (0.3x)       2282.8   (3.8x)
817.8   (2.7x)         3076.4   (3.8x)   3013.4   (3.8x)
```

```
=> FAILED   320000     6433   (0.3x)       3037.5   (5.1x)
1082.9   (3.6x)         3962.5   (5.0x)   3992.0   (5.0x)
=> FAILED   640000     3207   (0.2x)       3638.7   (6.1x)
1302.9   (4.3x)         4868.6   (6.1x)   4769.3   (6.0x)

Total: 0/28 tests passed: Could not complete tests in allotted
time, which results in a reported score of 0.
================================================================
```

Submission	
Submission time	Fri-23-Oct 13:39:06
Raw Score	83.35 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  17/21 tests passed
Memory:       8/8 tests passed
Timing:       34/41 tests passed

Aggregate score: 83.35% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.6K Oct 23 20:39 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 20:39 PointSET.java
```

```
-rw-r--r-- 1 3.0K Oct 23 20:39 studentSubmission.zip


**********************************************************************
**********
*  compiling
**********************************************************************
**********


% javac PointSET.java
*-----------------------------------------------------------


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

% javac KdTree.java
*-----------------------------------------------------------


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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L4 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 218]
Warnings generated: 1
================================================================


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

KdTree:

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

```
*****************************************************************
**********
*  correctness
*****************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
```

```
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed
```

```
Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
```

```
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
```

```
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
     -  failed on trial 1 of 100000
     -  student   nearest()    = (0.47489, 0.89947)
     -  reference nearest()    = (0.56479, 0.2338)
     -  student   distanceTo() = 0.6711053045536147
     -  reference distanceTo() = 0.001586474077947681
  *  100000 random general points in 10000-by-10000 grid
     -  failed on trial 1 of 100000
     -  student   nearest()    = (0.4139, 0.6325)
     -  reference nearest()    = (0.3726, 0.6095)
     -  student   distanceTo() = 0.04620270554848 49
     -  reference distanceTo() = 0.0017262676501631359
  *  100000 random general points in 1000-by-1000 grid
     -  failed on trial 1 of 100000
     -  student   nearest()    = (0.934, 0.455)
     -  reference nearest()    = (0.842, 0.027)
     -  student   distanceTo() = 0.4397317364030029
     -  reference distanceTo() = 0.0019999999999999983
  *  100000 random general points in 100-by-100 grid
     -  failed on trial 1 of 100000
     -  student   nearest()    = (0.93, 0.1)
     -  reference nearest()    = (0.95, 0.24)
     -  student   distanceTo() = 0.14142135623730948
```

```
      -  reference distanceTo() = 0.0
  *  100000 random general points in 10-by-10 grid
      -  failed on trial 1 of 100000
      -  student   nearest()    = (0.5, 0.9)
      -  reference nearest()    = (0.8, 0.2)
      -  student   distanceTo() = 0.7615773105863908
      -  reference distanceTo() = 0.0
==> FAILED

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.865, 0.465)
      -  reference nearest()    = (0.485, 0.551)
      -  student   distanceTo() = 0.3921033027149861
      -  reference distanceTo() = 0.0036055512754639926
  *  10000 random general points in 100-by-100 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.3, 0.31)
      -  reference nearest()    = (0.31, 0.34)
      -  student   distanceTo() = 0.03162277660168382
      -  reference distanceTo() = 0.0
  *  10000 random general points in 10-by-10 grid
      -  failed on trial 1 of 10000
      -  student   nearest()    = (0.0, 0.2)
      -  reference nearest()    = (0.0, 0.8)
      -  student   distanceTo() = 0.6000000000000001
      -  reference distanceTo() = 0.0
==> FAILED

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
      -  failed on trial 17 of 20000
      -  student   nearest()    = (0.14231, 0.13991)
      -  reference nearest()    = (0.46592, 0.55599)
      -  student   distanceTo() = 0.7208786808472006
      -  reference distanceTo() = 0.3953778913899967
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
```

```
      -  failed on trial 11 of 20000
      -  student   nearest()    = (0.4689, 0.0197)
      -  reference nearest()    = (0.5322, 0.2736)
      -  student   distanceTo() = 0.5368974017445046
      -  reference distanceTo() = 0.47568514797079803
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
      -  failed on trial 10 of 20000
      -  student   nearest()    = (0.821, 0.025)
      -  reference nearest()    = (0.436, 0.515)
      -  student   distanceTo() = 0.6392808459511359
      -  reference distanceTo() = 0.2412985702402731
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
      -  failed on trial 19 of 20000
      -  student   nearest()    = (0.6, 0.16)
      -  reference nearest()    = (0.33, 0.03)
      -  student   distanceTo() = 0.5703507692639679
      -  reference distanceTo() = 0.33541019662496846
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
      -  failed on trial 7 of 20000
      -  student   nearest()    = (0.2, 0.5)
      -  reference nearest()    = (0.4, 0.4)
      -  student   distanceTo() = 0.2
      -  reference distanceTo() = 0.09999999999999998
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
      -  failed on trial 18 of 20000
      -  student   nearest()    = (1.0, 0.0)
      -  reference nearest()    = (0.0, 1.0)
      -  student   distanceTo() = 1.4142135623730951
      -  reference distanceTo() = 0.0
==> FAILED

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
```

```
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:227)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:227)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:697)
     TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 9/13 tests passed!

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


****************************************************************
**********
*  memory
****************************************************************
**********

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



Computing memory of RectHV
*-----------------------------------------------------------
Memory of RectHV object = 48 bytes
================================================================
```

```
Computing memory of KdTree
*-----------------------------------------------------------
Running 8 total tests.

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

                N       student (bytes)    reference (bytes)
---------------------------------------------------------------
=> passed       1           168                160
=> passed       2           312                288
=> passed       5           744                672
=> passed      10          1464               1312
=> passed      25          3624               3232
=> passed     100         14424              12832
=> passed     400         57624              51232
=> passed     800        115224             102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
=================================================================



**************************************************************************
**********
*  timing
**************************************************************************
**********

Timing PointSET
*-----------------------------------------------------------
Running 13 total tests.
```

```
Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     985441
=> passed   320000    1154486
=> passed   640000     840812
=> passed  1280000     708333
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     533199
=> passed    20000     560979
=> passed    40000     525887
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2390
=> passed    20000       1142
=> passed    40000        518
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2747
=> passed    20000       1284
=> passed    40000        580
==> 3/3 tests passed

Total: 13/13 tests passed!

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

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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-------------------------------------------------------------------
-----------------------------------------
=> passed   160000     551877               1.0              22.6
21.6              21.6
=> passed   320000     680247               1.0              23.0
22.0              22.0
=> passed   640000     516393               1.0              24.5
23.5              23.5
=> passed  1280000     432108               1.0              26.6
25.6              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()                y()
equals()
-------------------------------------------------------------------
----------------------------
=> passed    10000     492330               18.5               17.5
18.0
=> passed    20000     502540               19.7               18.7
19.2
=> passed    40000     457815               21.8               20.8
21.3
```

```
=> passed    80000     439080              22.0              21.0
21.5
=> passed   160000     382077              23.2              22.2
22.7
=> passed   320000     326584              25.0              24.0
24.5
=> passed   640000     341062              25.7              24.7
25.2
=> passed  1280000     298350              27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-----------------------------------------------------------------
-----------------------------------------
=> passed    10000     291371               0.0              31.1
81.9              42.5
=> passed    20000     293149               0.0              32.6
85.9              48.8
=> passed    40000     305127               0.0              39.3
103.2              52.7
=> passed    80000     210648               0.0              40.7
106.5              55.0
=> passed   160000     181149               0.0              42.5
113.1              63.2
=> passed   320000     195153               0.0              40.2
105.7              55.7
=> passed   640000     131002               0.0              43.3
113.8              62.6
=> passed  1280000     115935               0.0              47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().
```

```
                                                  Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-------------------------------------------------------------------------
-------------------------------------------------
=> passed    10000    63337                 488.3
174.0                  633.7             627.3
=> FAILED    20000    33634                  792.9   (1.3x)
283.7                  1054.8   (1.3x)   1034.2   (1.3x)
=> FAILED    40000    21473   (0.7x)       1116.8   (1.9x)
397.6   (1.3x)         1503.4   (1.9x)   1465.7   (1.8x)
=> FAILED    80000    16593   (0.6x)       1344.0   (2.2x)
479.1   (1.6x)         1760.7   (2.2x)   1755.5   (2.2x)
=> FAILED   160000    10642   (0.4x)       1801.0   (3.0x)
644.5   (2.1x)         2388.6   (3.0x)   2389.1   (3.0x)
=> FAILED   320000     5842   (0.3x)       2526.0   (4.2x)
903.2   (3.0x)         3377.5   (4.2x)   3320.3   (4.2x)
=> FAILED   640000     2299   (0.1x)       5180.6   (8.6x)
1844.9   (6.1x)        6831.2   (8.5x)   6842.8   (8.6x)
=> FAILED  1280000     2269   (0.1x)       4251.1   (7.1x)
1526.0   (5.1x)        5662.2   (7.1x)   5608.0   (7.0x)
==> 1/8 tests passed

Total: 21/28 tests passed!

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

Submission	
Submission time	Fri-23-Oct 13:12:47
Raw Score	70.55 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
```

```
Style:          PASSED
Findbugs:       Potential bugs found.
API:            PASSED

Correctness:    17/21 tests passed
Memory:         8/8 tests passed
Timing:         13/41 tests passed

Aggregate score: 70.55% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.4K Oct 23 20:13 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 20:13 PointSET.java
-rw-r--r-- 1 3.0K Oct 23 20:13 studentSubmission.zip


================================================================
==========
*  compiling
================================================================
==========


% javac PointSET.java
*-----------------------------------------------------------


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


% javac KdTree.java
*-----------------------------------------------------------


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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
============================================================
```

```
% findbugs *.class
*-----------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L3 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 210]
Warnings generated: 1
================================================================


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

KdTree:

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


**************************************************************
**********
*  correctness
**************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed
```

```
Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
```

```
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3=0, p4, p5, and p6, respectively
        (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!


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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
```

```
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed
```

```
Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:205)
```

```
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   ...
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:218)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:348)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 10000-by-10000 grid
   java.lang.NullPointerException
   KdTree$Node.access$400(KdTree.java:7)
   KdTree.nearest(KdTree.java:199)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   ...
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:218)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:349)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 1000-by-1000 grid
   java.lang.NullPointerException
   KdTree$Node.access$500(KdTree.java:7)
   KdTree.nearest(KdTree.java:205)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   ...
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:218)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:350)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 100-by-100 grid
   java.lang.NullPointerException
   KdTree$Node.access$500(KdTree.java:7)
   KdTree.nearest(KdTree.java:205)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
```

```
   KdTree.nearest(KdTree.java:198)
   ...
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:218)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:351)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 10-by-10 grid
   java.lang.NullPointerException
   KdTree$Node.access$500(KdTree.java:7)
   KdTree.nearest(KdTree.java:205)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:198)
   ...
   KdTree.nearest(KdTree.java:198)
   KdTree.nearest(KdTree.java:218)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:352)
   TestKdTree.main(TestKdTree.java:737)

==> FAILED

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:205)
     KdTree.nearest(KdTree.java:198)
     KdTree.nearest(KdTree.java:198)
     KdTree.nearest(KdTree.java:198)
     ...
     KdTree.nearest(KdTree.java:198)
     KdTree.nearest(KdTree.java:218)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6b(TestKdTree.java:361)
     TestKdTree.main(TestKdTree.java:738)

  *  10000 random general points in 100-by-100 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:205)
```

```
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:198)
    ...
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testNearest(TestKdTree.java:318)
    TestKdTree.test6b(TestKdTree.java:362)
    TestKdTree.main(TestKdTree.java:738)

  *  10000 random general points in 10-by-10 grid
    java.lang.NullPointerException
    KdTree$Node.access$500(KdTree.java:7)
    KdTree.nearest(KdTree.java:205)
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:198)
    ...
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testNearest(TestKdTree.java:318)
    TestKdTree.test6b(TestKdTree.java:363)
    TestKdTree.main(TestKdTree.java:738)

==> FAILED

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$400(KdTree.java:7)
    KdTree.nearest(KdTree.java:199)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testAll(TestKdTree.java:637)
    TestKdTree.test7(TestKdTree.java:676)
    TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$500(KdTree.java:7)
```

```
    KdTree.nearest(KdTree.java:205)
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testAll(TestKdTree.java:637)
    TestKdTree.test7(TestKdTree.java:677)
    TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$500(KdTree.java:7)
    KdTree.nearest(KdTree.java:205)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testAll(TestKdTree.java:637)
    TestKdTree.test7(TestKdTree.java:678)
    TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$400(KdTree.java:7)
    KdTree.nearest(KdTree.java:199)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testAll(TestKdTree.java:637)
    TestKdTree.test7(TestKdTree.java:679)
    TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$500(KdTree.java:7)
    KdTree.nearest(KdTree.java:205)
    KdTree.nearest(KdTree.java:218)
    TestKdTree.testAll(TestKdTree.java:637)
    TestKdTree.test7(TestKdTree.java:680)
    TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
    java.lang.NullPointerException
    KdTree$Node.access$500(KdTree.java:7)
    KdTree.nearest(KdTree.java:205)
    KdTree.nearest(KdTree.java:198)
    KdTree.nearest(KdTree.java:218)
```

```
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:681)
     TestKdTree.main(TestKdTree.java:739)

==> FAILED

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:218)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$400(KdTree.java:7)
     KdTree.nearest(KdTree.java:218)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:697)
     TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 9/13 tests passed!

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

****************************************************************
**********
```

```
*  memory
*************************************************************************
**********

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



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



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

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

                 N       student (bytes)    reference (bytes)
------------------------------------------------------------
=> passed        1           168                 160
=> passed        2           312                 288
=> passed        5           744                 672
=> passed       10          1464                1312
=> passed       25          3624                3232
=> passed      100         14424               12832
=> passed      400         57624               51232
=> passed      800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
```

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


****************************************************************
**********
*  timing
****************************************************************
**********

Timing PointSET
*------------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     888731
=> passed   320000     786692
=> passed   640000     762509
=> passed  1280000     552062
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     526700
=> passed    20000     535517
=> passed    40000     486701
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2347
=> passed    20000       1095
=> passed    40000        493
==> 3/3 tests passed
```

```
Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2765
=> passed    20000       1281
=> passed    40000        565
==> 3/3 tests passed

Total: 13/13 tests passed!

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



Timing KdTree
*------------------------------------------------------------
Exception in thread "main" java.lang.NullPointerException
        at KdTree$Node.access$500(KdTree.java:7)
        at KdTree.nearest(KdTree.java:205)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:198)
        at KdTree.nearest(KdTree.java:218)
        at TimeKdTree.nearest(TimeKdTree.java:355)
        at TimeKdTree.nearestTest(TimeKdTree.java:580)
        at TimeKdTree.main(TimeKdTree.java:600)
Running 28 total tests.
```

```
Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-------------------------------------------------------------------
-----------------------------------------
=> passed   160000     572997               1.0              22.6
21.6              21.6
=> passed   320000     681980               1.0              23.0
22.0              22.0
=> passed   640000     538417               1.0              24.5
23.5              23.5
=> passed  1280000     425274               1.0              26.6
25.6              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()              y()
equals()
-------------------------------------------------------------------
----------------------------
=> passed    10000     499938               18.5             17.5
18.0
=> passed    20000     503316               19.7             18.7
19.2
=> passed    40000     439507               21.8             20.8
21.3
=> passed    80000     442138               22.0             21.0
21.5
=> passed   160000     345381               23.2             22.2
22.7
=> passed   320000     312431               25.0             24.0
24.5
=> passed   640000     301581               25.7             24.7
```

```
25.2
=> passed  1280000     282222              27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-----------------------------------------------------------------
-----------------------------------------
=> passed    10000     293817               0.0               31.1
81.9              42.5
=> passed    20000     273933               0.0               32.6
85.9              48.8
=> passed    40000     266240               0.0               39.3
103.2              52.7
=> passed    80000     215918               0.0               40.7
106.5              55.0
=> passed   160000     170636               0.0               42.5
113.1              63.2
=> passed   320000     178061               0.0               40.2
105.7              55.7
=> passed   640000     127607               0.0               43.3
113.8              62.6
=> passed  1280000     145041               0.0               47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-----------------------------------------------------------------
-----------------------------------------------
```

```
Total: 0/28 tests passed: Could not complete tests in allotted
time, which results in a reported score of 0.
================================================================
```

Submission	
Submission time	Fri-23-Oct 07:44:27
Raw Score	70.55 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  17/21 tests passed
Memory:       8/8 tests passed
Timing:       13/41 tests passed

Aggregate score: 70.55% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 16K
-rw-r--r-- 1 7.8K Oct 23 14:44 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 14:44 PointSET.java
-rw-r--r-- 1 2.9K Oct 23 14:44 studentSubmission.zip


**************************************************************
***********
```

```
*  compiling
************************************************************************
**********


% javac PointSET.java
*------------------------------------------------------------


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


% javac KdTree.java
*------------------------------------------------------------


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



% checkstyle *.java readme.txt
*------------------------------------------------------------
================================================================


% findbugs *.class
*------------------------------------------------------------
M D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L3 in KdTree.nearest(P
oint2D, KdTree$Node)  At KdTree.java:[line 188]
Warnings generated: 1
================================================================


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

KdTree:

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


************************************************************************
**********
*  correctness
************************************************************************
**********
```

```
Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
```

```
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

```
Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
```

```
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
```

```
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     ...
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6a(TestKdTree.java:348)
     TestKdTree.main(TestKdTree.java:737)

  *  100000 random general points in 10000-by-10000 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     ...
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6a(TestKdTree.java:349)
     TestKdTree.main(TestKdTree.java:737)
```

```
*  100000 random general points in 1000-by-1000 grid
   java.lang.NullPointerException
   KdTree$Node.access$500(KdTree.java:7)
   KdTree.nearest(KdTree.java:183)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   ...
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:196)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:350)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 100-by-100 grid
   java.lang.NullPointerException
   KdTree$Node.access$500(KdTree.java:7)
   KdTree.nearest(KdTree.java:183)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   ...
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:196)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:351)
   TestKdTree.main(TestKdTree.java:737)

*  100000 random general points in 10-by-10 grid
   java.lang.NullPointerException
   KdTree$Node.access$100(KdTree.java:7)
   KdTree.nearest(KdTree.java:177)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:176)
   ...
   KdTree.nearest(KdTree.java:176)
   KdTree.nearest(KdTree.java:196)
   TestKdTree.testNearest(TestKdTree.java:318)
   TestKdTree.test6a(TestKdTree.java:352)
   TestKdTree.main(TestKdTree.java:737)

==> FAILED
```

```
Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
     java.lang.NullPointerException
     KdTree$Node.access$100(KdTree.java:7)
     KdTree.nearest(KdTree.java:177)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     ...
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6b(TestKdTree.java:361)
     TestKdTree.main(TestKdTree.java:738)

  *  10000 random general points in 100-by-100 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     ...
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6b(TestKdTree.java:362)
     TestKdTree.main(TestKdTree.java:738)

  *  10000 random general points in 10-by-10 grid
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testNearest(TestKdTree.java:318)
     TestKdTree.test6b(TestKdTree.java:363)
     TestKdTree.main(TestKdTree.java:738)

==> FAILED
```

```
Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$100(KdTree.java:7)
     KdTree.nearest(KdTree.java:177)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:676)
     TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:184)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:677)
     TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:184)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:678)
     TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
```

```
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:679)
     TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$500(KdTree.java:7)
     KdTree.nearest(KdTree.java:183)
     KdTree.nearest(KdTree.java:176)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:680)
     TestKdTree.main(TestKdTree.java:739)

  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$100(KdTree.java:7)
     KdTree.nearest(KdTree.java:177)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test7(TestKdTree.java:681)
     TestKdTree.main(TestKdTree.java:739)

==> FAILED

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree$Node.access$100(KdTree.java:7)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
```

```
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree$Node.access$100(KdTree.java:7)
     KdTree.nearest(KdTree.java:196)
     TestKdTree.testAll(TestKdTree.java:637)
     TestKdTree.test8(TestKdTree.java:697)
     TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 9/13 tests passed!

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

****************************************************************
**********
*  memory
****************************************************************
**********

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



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



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

Memory usage of a KdTree with N points (including Point2D and RectH
```

```
V objects).
Maximum allowed memory is 312N + 192 bytes.

                 N       student (bytes)    reference (bytes)
-------------------------------------------------------------
=> passed        1           168                 160
=> passed        2           312                 288
=> passed        5           744                 672
=> passed       10          1464                1312
=> passed       25          3624                3232
=> passed      100         14424               12832
=> passed      400         57624               51232
=> passed      800        115224              102432
==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student   memory (bytes) = 144.00 N + 24.00  (R^2 = 1.000
)
Estimated reference memory (bytes) = 128.00 N + 32.00  (R^2 = 1.000
)
================================================================



****************************************************************
***********
*  timing
****************************************************************
***********

Timing PointSET
*----------------------------------------------------------
Running 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
-----------------------------------------
=> passed   160000     986847
=> passed   320000    1091976
=> passed   640000     831942
=> passed  1280000     550630
```

```
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     522955
=> passed    20000     558265
=> passed    40000     523358
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2345
=> passed    20000       1118
=> passed    40000        509
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2760
=> passed    20000       1268
=> passed    40000        571
==> 3/3 tests passed

Total: 13/13 tests passed!

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



Timing KdTree
*------------------------------------------------------------
Exception in thread "main" java.lang.NullPointerException
        at KdTree$Node.access$500(KdTree.java:7)
        at KdTree.nearest(KdTree.java:183)
        at KdTree.nearest(KdTree.java:176)
```

```
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:176)
        at KdTree.nearest(KdTree.java:196)
        at TimeKdTree.nearest(TimeKdTree.java:355)
        at TimeKdTree.nearestTest(TimeKdTree.java:580)
        at TimeKdTree.main(TimeKdTree.java:600)
Running 28 total tests.


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
=> FAILED   160000     408640              22.6  (11.3x)     33.2
31.7             21.6
=> FAILED   320000     394969              23.0  (11.5x)     33.8
32.3             22.0
=> FAILED   640000     350741              24.5  (12.3x)     36.1
34.6             23.5
=> FAILED  1280000     314026              26.6  (13.3x)     39.2
37.7             25.6
==> 0/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
```

```
to contain().


Point2D
               N      ops per second       x()               y()
equals()
-------------------------------------------------------------------
---------------------------
=> passed    10000     435025             18.5              17.5
18.0
=> passed    20000     429213             19.7              18.7
19.2
=> passed    40000     431243             21.8              20.8
21.3
=> passed    80000     376253             22.0              21.0
21.5
=> passed   160000     327351             23.2              22.2
22.7
=> passed   320000     342134             25.0              24.0
24.5
=> passed   640000     339186             25.7              24.7
25.2
=> passed  1280000     292933             27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-----------------------------------------
=> passed    10000     272908              0.0              31.1
81.9              42.5
=> passed    20000     272615              0.0              32.6
85.9              48.8
=> passed    40000     238603              0.0              39.3
103.2              52.7
=> passed    80000     205473              0.0              40.7
106.5              55.0
=> passed   160000     187034              0.0              42.5
```

```
113.1              63.2
=> passed   320000     196006              0.0              40.2
105.7              55.7
=> passed   640000     193719              0.0              43.3
113.8              62.6
=> passed  1280000     155739              0.0              47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-----------------------------------------------------------------
-------------------------------------------------

Total: 0/28 tests passed: Could not complete tests in allotted
time, which results in a reported score of 0.
================================================================
```

Submission	
Submission time	Thu-22-Oct 21:36:31
Raw Score	96.34 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED
```

```
Correctness:  21/21 tests passed
Memory:       8/8 tests passed
Timing:       35/41 tests passed

Aggregate score: 96.34% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 20K
-rw-r--r-- 1 8.4K Oct 23 04:37 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 04:37 PointSET.java
-rw-r--r-- 1 2.8K Oct 23 04:37 studentSubmission.zip


*****************************************************************
**********
*  compiling
*****************************************************************
**********


% javac PointSET.java
*-----------------------------------------------------------


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


% javac KdTree.java
*-----------------------------------------------------------


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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
```

```
*-----------------------------------------------------------
============================================================


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

KdTree:

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


************************************************************
**********
*  correctness
************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
```

```
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
```

```
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed
```

```
Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
```

```
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
```

```
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
==> passed

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
==> passed


Total: 13/13 tests passed!

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

```
*****************************************************************
**********
*  memory
*****************************************************************
**********

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



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



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

Memory usage of a KdTree with N points (including Point2D and RectH
V 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 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     994471
=> passed   320000    1100679
=> passed   640000     827056
=> passed  1280000     693216
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     534604
=> passed    20000     565001
=> passed    40000     533952
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2333
=> passed    20000       1125
=> passed    40000        513
```

```
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2772
=> passed    20000       1302
=> passed    40000        585
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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

Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
=> passed   160000     860798               0.0              22.1
21.1              21.6
=> passed   320000     915591               0.0              22.5
21.5              22.0
=> passed   640000     704231               0.0              24.0
23.0              23.5
=> passed  1280000     562342               0.0              26.1
25.1              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
```

```
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()               y()
equals()
-------------------------------------------------------------------
---------------------------
=> passed    10000     509426              18.5              17.5
18.0
=> passed    20000     518866              19.7              18.7
19.2
=> passed    40000     484059              21.8              20.8
21.3
=> passed    80000     455387              22.0              21.0
21.5
=> passed   160000     459331              23.2              22.2
22.7
=> passed   320000     358965              25.0              24.0
24.5
=> passed   640000     324772              25.7              24.7
25.2
=> passed  1280000     279550              27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-------------------------------------------
=> passed    10000     286219               0.0              31.1
81.9              42.5
=> passed    20000     286505               0.0              32.6
85.9              48.8
=> passed    40000     270041               0.0              39.3
103.2              52.7
=> passed    80000     252541               0.0              40.7
```

```
106.5              55.0
=> passed   160000     225904              0.0              42.5
113.1              63.2
=> passed   320000     182450              0.0              40.2
105.7              55.7
=> passed   640000     148281              0.0              43.3
113.8              62.6
=> passed  1280000     121363              0.0              47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()              y()
-----------------------------------------------------------------
-------------------------------------------------
=> passed    10000   112135                 172.8
86.9                  398.8             305.9
=> passed    20000    91534                 225.2
113.1                 514.1              390.1
=> FAILED    40000    50659                 448.8
224.9                 980.3   (1.2x)    739.0
=> FAILED    80000    46887                 463.2
232.1                 1017.1   (1.3x)    770.7
=> FAILED   160000    21668   (0.7x)        964.1   (1.6x)
482.6   (1.6x)        2034.4   (2.5x)   1526.9   (1.9x)
=> FAILED   320000    24890                 735.7   (1.2x)
368.4   (1.2x)        1571.0   (2.0x)   1179.3   (1.5x)
=> FAILED   640000    11433   (0.6x)       1321.9   (2.2x)
661.4   (2.2x)        2762.0   (3.5x)   2073.2   (2.6x)
=> FAILED  1280000     8286   (0.4x)       1855.2   (3.1x)
928.1   (3.1x)        3849.8   (4.8x)   2890.9   (3.6x)
==> 2/8 tests passed

Total: 22/28 tests passed!

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

Submission	
Submission time	Thu-22-Oct 19:04:51
Raw Score	96.34 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        PASSED
Findbugs:     No potential bugs found.
API:          PASSED

Correctness:  21/21 tests passed
Memory:       8/8 tests passed
Timing:       35/41 tests passed

Aggregate score: 96.34% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 16K
-rw-r--r-- 1 8.0K Oct 23 02:05 KdTree.java
-rw-r--r-- 1 1.9K Oct 23 02:05 PointSET.java
-rw-r--r-- 1 2.8K Oct 23 02:05 studentSubmission.zip


********************************************************************
***********
*  compiling
********************************************************************
***********
```

```
% javac PointSET.java
*-----------------------------------------------------------

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

% javac KdTree.java
*-----------------------------------------------------------

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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
================================================================


% findbugs *.class
*-----------------------------------------------------------
================================================================

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

KdTree:

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


**************************************************************************
**********
*  correctness
**************************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
```

```
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
```

```
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
        contains(), range(), and nearest() with probabilities
        p1, p2, p3=0, p4, p5, and p6, respectively
        (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
  *  10 random general points in 1-by-1 grid
==> passed

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3a: Insert N distinct points and call contains() with random q
uery points
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
```

```
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
```

```
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
0.1, 0.2, 0.2
==> passed

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
```

```
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
==> passed


Total: 13/13 tests passed!

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

****************************************************************
**********
*  memory
****************************************************************
**********

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



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



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

Memory usage of a KdTree with N points (including Point2D and RectH
```

```
V 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 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     967652
=> passed   320000    1091435
=> passed   640000     813296
=> passed  1280000     677230
==> 4/4 tests passed
```

```
Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     534461
=> passed    20000     562116
=> passed    40000     532471
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2356
=> passed    20000       1127
=> passed    40000        518
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2829
=> passed    20000       1314
=> passed    40000        581
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
```

```
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-------------------------------------------------------------------
-----------------------------------------
=> passed   160000     878426               0.0              22.1
21.1              21.6
=> passed   320000     900660               0.0              22.5
21.5              22.0
=> passed   640000     689475               0.0              24.0
23.0              23.5
=> passed  1280000     553580               0.0              26.1
25.1              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()                y()
equals()
-------------------------------------------------------------------
----------------------------
=> passed    10000     498972               18.5               17.5
18.0
=> passed    20000     511570               19.7               18.7
19.2
=> passed    40000     483792               21.8               20.8
21.3
=> passed    80000     447618               22.0               21.0
21.5
=> passed   160000     463755               23.2               22.2
22.7
=> passed   320000     370929               25.0               24.0
24.5
=> passed   640000     312330               25.7               24.7
25.2
=> passed  1280000     271230               27.2               26.2
```

```
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-----------------------------------------
=> passed    10000     278822              0.0              31.1
81.9             42.5
=> passed    20000     296914              0.0              32.6
85.9             48.8
=> passed    40000     270203              0.0              39.3
103.2             52.7
=> passed    80000     247734              0.0              40.7
106.5             55.0
=> passed   160000     215609              0.0              42.5
113.1             63.2
=> passed   320000     178680              0.0              40.2
105.7             55.7
=> passed   640000     143647              0.0              43.3
113.8             62.6
=> passed  1280000     158866              0.0              47.0
123.0             60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
              N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-------------------------------------------------------------------
-----------------------------------------------
=> passed    10000   111386                230.8
86.9                  398.8             305.9
=> passed    20000    88463                299.6
```

```
113.1                    514.1              390.1
=> FAILED    40000    49325                 591.5
224.9                    980.3   (1.2x)     739.0
=> FAILED    80000    43430                 610.8
232.1                   1017.1   (1.3x)     770.7
=> FAILED   160000    20769   (0.7x)       1253.1   (2.1x)
478.2   (1.6x)          2016.5   (2.5x)   1513.4   (1.9x)
=> FAILED   320000    21114                1079.1   (1.8x)
411.5   (1.4x)          1748.4   (2.2x)   1323.0   (1.7x)
=> FAILED   640000    12473   (0.6x)       1598.5   (2.7x)
610.0   (2.0x)          2559.2   (3.2x)   1923.3   (2.4x)
=> FAILED  1280000    10885   (0.5x)       2028.8   (3.4x)
777.7   (2.6x)          3277.9   (4.1x)   2426.2   (3.0x)
==> 2/8 tests passed

Total: 22/28 tests passed!

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

Submission	
Submission time	Thu-22-Oct 14:51:41
Raw Score	87.06 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        FAILED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  18/21 tests passed
Memory:       8/8 tests passed
Timing:       35/41 tests passed

Aggregate score: 87.06% [Correctness: 65%, Memory: 10%, Timing: 25%
```

```
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 16K
-rw-r--r-- 1 8.0K Oct 22 21:52 KdTree.java
-rw-r--r-- 1 2.3K Oct 22 21:52 PointSET.java
-rw-r--r-- 1 2.9K Oct 22 21:52 studentSubmission.zip


================================================================
===========
*  compiling
================================================================
===========


% javac PointSET.java
*-----------------------------------------------------------


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


% javac KdTree.java
*-----------------------------------------------------------


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



% checkstyle *.java readme.txt
*-----------------------------------------------------------
KdTree.java:8:17: Variable 'point' should be private.
KdTree.java:9:17: Variable 'vertical' should be private.
KdTree.java:10:14: Variable 'left' should be private.
KdTree.java:11:14: Variable 'right' should be private.
KdTree.java:12:13: Variable 'count' should be private.
KdTree.java:198:46: ',' is not followed by whitespace.
KdTree.java:198:48: ',' is not followed by whitespace.
KdTree.java:198:50: ',' is not followed by whitespace.
KdTree.java:208:54: ',' is not followed by whitespace.
Checkstyle ends with 9 errors.
```

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


% findbugs *.class
*---------------------------------------------------------
H D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L2 in PointSET.main(St
ring[])  At PointSET.java:[line 66]
Warnings generated: 1
================================================================


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

KdTree:

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


**************************************************************
***********
*  correctness
**************************************************************
***********

Testing methods in PointSET
*---------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed
```

```
Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed
```

```
Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
```

```
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 1b: Insert N points and check size() after each insertion
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
      -  failed on trial 10963 of 100000
      -  student   size() = 10963
      -  reference size() = 10962
  *  100000 random general points in 1000-by-1000 grid
      -  failed on trial 251 of 100000
      -  student   size() = 251
      -  reference size() = 250
  *  100000 random general points in 100-by-100 grid
      -  failed on trial 169 of 100000
      -  student   size() = 169
      -  reference size() = 168
  *  100000 random general points in 10-by-10 grid
      -  failed on trial 8 of 100000
      -  student   size() = 8
      -  reference size() = 7
  *  10 random general points in 1-by-1 grid
      -  failed on trial 4 of 10
      -  student   size() = 4
      -  reference size() = 3
==> FAILED

Test 2: Test isEmpty() by checking that it returns the right result
s for 0, 1, and 2 points
  *  zero points
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
     TestKdTree.test2(TestKdTree.java:128)
     TestKdTree.main(TestKdTree.java:730)

==> FAILED

Test 3a: Insert N distinct points and call contains() with random q
uery points
```

```
  *  100000 random distinct points in 100000-by-100000 grid
  *  100000 random distinct points in 10000-by-10000 grid
  *  100000 random distinct points in 1000-by-1000 grid
  *  10000 random distinct points in 100-by-100 grid
  *  100 random distinct points in 10-by-10 grid
  *  1 random distinct points in 1-by-1 grid
==> passed

Test 3b: Insert N points and call contains() with random query poin
ts
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
  *  10000 random general points in 1-by-1 grid
==> passed

Test 4: Test whether two KdTree objects can be created at the same
time
==> passed

Test 5a: Insert N distinct points and call range() for random query
rectangles
  *  4000 random rectangles and 4000 distinct points in 100000-by-1
00000 grid
  *  4000 random rectangles and 4000 distinct points in 10000-by-10
000 grid
  *  4000 random rectangles and 4000 distinct points in 1000-by-100
0 grid
  *  4000 random rectangles and 4000 distinct points in 100-by-100
grid
  *  40 random rectangles and 40 distinct points in 10-by-10 grid
  *  1 random rectangles and 1 distinct points in 1-by-1 grid
==> passed

Test 5b: Insert N points and call range() for random query rectangl
es
  *  4000 random rectangles and 4000 random general points in 10000
-by-10000 grid
  *  4000 random rectangles and 4000 random general points in 1000-
by-1000 grid
  *  4000 random rectangles and 4000 random general points in 100-b
y-100 grid
  *  4000 random rectangles and 4000 random general points in 10-by
-10 grid
```

```
  *  4000 random rectangles and 4000 random general points in 1-by-
1 grid
==> passed

Test 5c: Insert N points and call range() for tiny
rectangles enclosing each point.
  *  4000 tiny rectangles and 4000 points in 100000-by-100000 grid
  *  4000 tiny rectangles and 4000 points in 10000-by-10000 grid
  *  4000 tiny rectangles and 4000 points in 1000-by-1000 grid
  *  4000 tiny rectangles and 4000 points in 100-by-100 grid
  *  4000 tiny rectangles and 4000 points in 10-by-10 grid
==> passed

Test 6a: Insert N distinct points and call nearest() with random qu
ery points
  *  100000 random general points in 100000-by-100000 grid
  *  100000 random general points in 10000-by-10000 grid
  *  100000 random general points in 1000-by-1000 grid
  *  100000 random general points in 100-by-100 grid
  *  100000 random general points in 10-by-10 grid
==> passed

Test 6b: Insert N points and call nearest() with random query point
s
  *  10000 random general points in 1000-by-1000 grid
  *  10000 random general points in 100-by-100 grid
  *  10000 random general points in 10-by-10 grid
==> passed

Test 7: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3, p4, p5, and p6, respectively
  *  20000 calls in 100000-by-100000 grid with probabilties 0.05, 0
.05, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 10000-by-10000 grid with probabilties 0.05, 0.0
5, 0.3, 0.1, 0.2, 0.2
  *  20000 calls in 1000-by-1000 grid with probabilties 0.05, 0.05,
0.3, 0.1, 0.2, 0.2
  *  20000 calls in 100-by-100 grid with probabilties 0.05, 0.05, 0
.3, 0.1, 0.2, 0.2
  *  20000 calls in 10-by-10 grid with probabilties 0.05, 0.05, 0.3
, 0.1, 0.2, 0.2
  *  20000 calls in 1-by-1 grid with probabilties 0.05, 0.05, 0.3,
```

```
0.1, 0.2, 0.2
==> passed

Test 8: test intermixed sequence of calls to isEmpty(), size(), ins
ert(),
        contains(), range(), and nearest() with probabilities
        p1, p2, p3 = 0, p4, p5, and p6, respectively
        (a data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with probabilties 0.5, 0.5, 0.
0, 0.0, 0.0, 0.0
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
     TestKdTree.testAll(TestKdTree.java:580)
     TestKdTree.test8(TestKdTree.java:693)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.6, 0.0, 0.0
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
     TestKdTree.testAll(TestKdTree.java:580)
     TestKdTree.test8(TestKdTree.java:694)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.6, 0.0
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
     TestKdTree.testAll(TestKdTree.java:580)
     TestKdTree.test8(TestKdTree.java:695)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.0, 0.0, 0.6
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
     TestKdTree.testAll(TestKdTree.java:580)
     TestKdTree.test8(TestKdTree.java:696)
     TestKdTree.main(TestKdTree.java:740)

  *  1000 calls in 1000-by-1000 grid with probabilties 0.2, 0.2, 0.
0, 0.2, 0.2, 0.2
     java.lang.NullPointerException
     KdTree.size(KdTree.java:41)
```

```
    TestKdTree.testAll(TestKdTree.java:580)
    TestKdTree.test8(TestKdTree.java:697)
    TestKdTree.main(TestKdTree.java:740)

==> FAILED


Total: 10/13 tests passed!

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

*****************************************************************
**********
*  memory
*****************************************************************
**********

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


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



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

Memory usage of a KdTree with N points (including Point2D and RectH
V 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 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
----------------------------------------
=> passed   160000     985642
=> passed   320000    1127064
=> passed   640000     858418
=> passed  1280000     624460
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
----------------------------------------
=> passed    10000     520352
=> passed    20000     542277
```

```
=> passed    40000     485054
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
----------------------------------------
=> passed    10000       2389
=> passed    20000       1139
=> passed    40000        516
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
ET.

               N      ops per second
----------------------------------------
=> passed    10000       2781
=> passed    20000       1303
=> passed    40000        585
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
```

```
=> passed   160000     942159               0.0               22.1
21.1              21.6
=> passed   320000     956118               0.0               22.5
21.5              22.0
=> passed   640000     723987               0.0               24.0
23.0              23.5
=> passed  1280000     581800               0.0               26.1
25.1              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().


Point2D
               N      ops per second       x()               y()
equals()
-------------------------------------------------------------------
---------------------------
=> passed    10000     506452               18.5              17.5
18.0
=> passed    20000     518306               19.7              18.7
19.2
=> passed    40000     482472               21.8              20.8
21.3
=> passed    80000     455493               22.0              21.0
21.5
=> passed   160000     476675               23.2              22.2
22.7
=> passed   320000     375030               25.0              24.0
24.5
=> passed   640000     319949               25.7              24.7
25.2
=> passed  1280000     279213               27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()              y()
-------------------------------------------------------------------
-----------------------------------------
=> passed    10000     281506               0.0               31.1
81.9              42.5
=> passed    20000     285826               0.0               32.6
85.9              48.8
=> passed    40000     267599               0.0               39.3
103.2             52.7
=> passed    80000     245310               0.0               40.7
106.5             55.0
=> passed   160000     210372               0.0               42.5
113.1             63.2
=> passed   320000     185980               0.0               40.2
105.7             55.7
=> passed   640000     136455               0.0               43.3
113.8             62.6
=> passed  1280000     123713               0.0               47.0
123.0             60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                          Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-------------------------------------------------------------------
-----------------------------------------------------
=> passed    10000   115383                 230.8
86.9                   398.8             305.9
=> passed    20000    92716                 299.6
113.1                  514.1             390.1
=> FAILED    40000    50388                 591.5
224.9                  980.3   (1.2x)    739.0
=> FAILED    80000    44161                 610.8
232.1                  1017.1   (1.3x)    770.7
=> FAILED   160000    18747   (0.6x)       1260.4   (2.1x)
481.0   (1.6x)         2027.6   (2.5x)   1521.9   (1.9x)
=> FAILED   320000    21730                 1042.4   (1.7x)
```

```
397.3   (1.3x)          1691.0   (2.1x)   1268.0   (1.6x)
=> FAILED   640000    10785   (0.5x)       1786.9   (3.0x)
683.0   (2.3x)          2850.4   (3.6x)   2140.0   (2.7x)
=> FAILED  1280000    11979   (0.6x)       1854.3   (3.1x)
711.1   (2.4x)          3012.4   (3.8x)   2225.3   (2.8x)
==> 2/8 tests passed

Total: 22/28 tests passed!

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

Submission	
Submission time	Thu-22-Oct 13:58:23
Raw Score	56.10 / 100.00
Feedback	See the Assessment Guide for information on how to interpret this report.

Assessment Summary

```
Compilation:  PASSED
Style:        FAILED
Findbugs:     Potential bugs found.
API:          PASSED

Correctness:  8/21 tests passed
Memory:       8/8 tests passed
Timing:       35/41 tests passed

Aggregate score: 56.10% [Correctness: 65%, Memory: 10%, Timing: 25%
, Style: 0%]
```

Assessment Details

```
The following files were submitted:
----------------------------------
total 16K
```

```
-rw-r--r-- 1 7.8K Oct 22 21:14 KdTree.java
-rw-r--r-- 1 2.3K Oct 22 21:14 PointSET.java
-rw-r--r-- 1 2.8K Oct 22 21:14 studentSubmission.zip


**********************************************************************
**********
*  compiling
**********************************************************************
**********


% javac PointSET.java
*-----------------------------------------------------------


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


% javac KdTree.java
*-----------------------------------------------------------


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


% checkstyle *.java readme.txt
*-----------------------------------------------------------
KdTree.java:8:17: Variable 'point' should be private.
KdTree.java:9:17: Variable 'vertical' should be private.
KdTree.java:10:14: Variable 'left' should be private.
KdTree.java:11:14: Variable 'right' should be private.
KdTree.java:192:46: ',' is not followed by whitespace.
KdTree.java:192:48: ',' is not followed by whitespace.
KdTree.java:192:50: ',' is not followed by whitespace.
KdTree.java:202:54: ',' is not followed by whitespace.
Checkstyle ends with 8 errors.
================================================================


% findbugs *.class
*-----------------------------------------------------------
H D DLS_DEAD_LOCAL_STORE DLS: Dead store to $L2 in PointSET.main(St
ring[])  At PointSET.java:[line 66]
Warnings generated: 1
================================================================
```

```
Testing the APIs of your programs.
*-----------------------------------------------------------
PointSET:

KdTree:

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


*****************************************************************
**********
*  correctness
*****************************************************************
**********

Testing methods in PointSET
*-----------------------------------------------------------
Running 8 total tests.

Test 1: Test size() by inserting N random points
        (size may be less than N because of duplicates)
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed

Test 2: Test isEmpty() by checking for N = 0, 1, and 2 points
  *  zero points
  *  one point
  *  two points
==> passed

Test 3: Insert N random points and check contains() for random quer
y points
  *  100000 random points in 100000-by-100000 grid
  *  100000 random points in 10000-by-10000 grid
  *  100000 random points in 1000-by-1000 grid
  *  100000 random points in 100-by-100 grid
  *  100000 random points in 10-by-10 grid
==> passed
```

```
Test 4: Insert N random points and check nearest() for random query
points
  *  3000 random points in 100000-by-100000 grid
  *  3000 random points in 10000-by-10000 grid
  *  3000 random points in 1000-by-1000 grid
  *  3000 random points in 100-by-100 grid
  *  3000 random points in 10-by-10 grid
==> passed

Test 5: Insert N random points and check range() for random query r
ectangles
  *  1000 random rectangles and points in 100000-by-100000 grid
  *  1000 random rectangles and points in 10000-by-10000 grid
  *  1000 random rectangles and points in 1000-by-1000 grid
  *  1000 random rectangles and points in 100-by-100 grid
  *  1000 random rectangles and points in 10-by-10 grid
==> passed

Test 6: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3, p4, p5, and p6, respectively
  *  10000 calls in 10000-by-10000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1000-by-1000 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 100-by-100 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 10-by-10 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
  *  10000 calls in 1-by-1 grid with random points
     and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2
==> passed

Test 7: Intermixed sequence of calls to isEmpty(), size(), insert()
,
       contains(), range(), and nearest() with probabilities
       p1, p2, p3=0, p4, p5, and p6, respectively
       (data structure with 0 points)
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0
```

```
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6
  *  1000 calls in 1000-by-1000 grid with random points
     and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2
==> passed

Test 8: Test whether two PointSET objects can be created at the sam
e time
==> passed


Total: 8/8 tests passed!

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

Testing methods in KdTree
*-----------------------------------------------------------
In the tests below, we consider three classes of points and rectang
les.

  *  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-coor
dinates
     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

Running 13 total tests.

Test 1a: Insert N distinct points and check size() after each inser
tion
  *  100000 random distinct points in 100000-by-100000 grid

Total: 0/13 tests passed: Could not complete tests in allotted
time, which results in a reported score of 0.
================================================================

****************************************************************
**********
*  memory
```

```
**********************************************************************
**********

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



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



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

Memory usage of a KdTree with N points (including Point2D and RectH
V 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 13 total tests.


Inserting N points into a PointSET.

               N      ops per second
---------------------------------------
=> passed   160000     819659
=> passed   320000     874370
=> passed   640000     674247
=> passed  1280000     556875
==> 4/4 tests passed

Performing contains() queries after inserting N points into a Point
SET.

               N      ops per second
---------------------------------------
=> passed    10000     513132
=> passed    20000     520422
=> passed    40000     475594
==> 3/3 tests passed

Performing range() queries after inserting N points into a PointSET
.

               N      ops per second
---------------------------------------
=> passed    10000       2357
=> passed    20000       1104
=> passed    40000        495
==> 3/3 tests passed

Performing nearest() queries after inserting N points into a PointS
```

```
ET.

               N      ops per second
-----------------------------------------
=> passed    10000       2679
=> passed    20000       1288
=> passed    40000        582
==> 3/3 tests passed

Total: 13/13 tests passed!

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



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


Inserting N points into a 2d tree. The table gives the average numb
er of calls to methods
in RectHV and Point per call to insert().


Point2D
               N      ops per second       RectHV()           x()
y()             equals()
-----------------------------------------------------------------
-----------------------------------------
=> passed   160000     970953               0.0              22.1
21.1              21.6
=> passed   320000     887996               0.0              22.5
21.5              22.0
=> passed   640000     724492               0.0              24.0
23.0              23.5
=> passed  1280000     564393               0.0              26.1
25.1              25.6
==> 4/4 tests passed

Performing contains() queries after inserting N points into a 2d tr
ee. The table gives
the average number of calls to methods in RectHV and Point per call
to contain().
```

```
Point2D
               N      ops per second       x()               y()
equals()
-------------------------------------------------------------------
---------------------------
=> passed    10000     400130             18.5              17.5
18.0
=> passed    20000     476116             19.7              18.7
19.2
=> passed    40000     419053             21.8              20.8
21.3
=> passed    80000     364833             22.0              21.0
21.5
=> passed   160000     370251             23.2              22.2
22.7
=> passed   320000     335856             25.0              24.0
24.5
=> passed   640000     300404             25.7              24.7
25.2
=> passed  1280000     276179             27.2              26.2
26.7
==> 8/8 tests passed

Performing 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()      contai
ns()        x()               y()
-------------------------------------------------------------------
-----------------------------------------
=> passed    10000     299152              0.0              31.1
81.9              42.5
=> passed    20000     298861              0.0              32.6
85.9              48.8
=> passed    40000     286077              0.0              39.3
103.2              52.7
=> passed    80000     249673              0.0              40.7
106.5              55.0
=> passed   160000     219475              0.0              42.5
113.1              63.2
```

```
=> passed   320000     171795               0.0              40.2
105.7              55.7
=> passed   640000     135637               0.0              43.3
113.8              62.6
=> passed  1280000     142432               0.0              47.0
123.0              60.1
==> 8/8 tests passed

Performing nearest() queries after inserting N points into a 2d tre
e. The table gives
the average number of calls to methods in RectHV and Point per call
to nearest().

                                        Point2D                 Re
ctHV
               N      ops per second     distanceSquaredTo()     di
stanceSquaredTo()        x()               y()
-------------------------------------------------------------------
-------------------------------------------------
=> passed    10000   113227                230.8
86.9                  398.8             305.9
=> passed    20000    92334                299.6
113.1                 514.1             390.1
=> FAILED    40000    45519                591.5
224.9                 980.3   (1.2x)    739.0
=> FAILED    80000    37473                610.8
232.1                 1017.1   (1.3x)   770.7
=> FAILED   160000    18169   (0.6x)       1269.1   (2.1x)
484.3   (1.6x)        2040.8   (2.6x)   1532.1   (1.9x)
=> FAILED   320000    16373   (0.8x)       1259.9   (2.1x)
481.1   (1.6x)        2039.2   (2.5x)   1525.7   (1.9x)
=> FAILED   640000    19134                1172.0   (2.0x)
446.5   (1.5x)        1895.9   (2.4x)   1420.0   (1.8x)
=> FAILED  1280000     5141   (0.3x)       3107.1   (5.2x)
1189.4   (4.0x)        4893.2   (6.1x)   3678.3   (4.6x)
==> 2/8 tests passed

Total: 22/28 tests passed!

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

Submission	
Submission time	Thu-22-Oct 13:52:44
Raw Score	0.00 / 100.00
Feedback	Compilation: PASSED API: FAILED KdTree: The following fields should be made private: * KdTree$Node root