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



返回到第 5 周**课程**

[上一个](https://www.coursera.org/learn/algorithms-part1/lecture/mNiwq/rectangle-intersection)

[下一个](https://www.coursera.org/learn/algorithms-part1/supplement/iHcBs/overview)

### Balanced Search Trees

### Geometric Applications of BSTs

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

##### [1d Range Search](https://www.coursera.org/learn/algorithms-part1/lecture/wSISD/1d-range-search)

[8 分](https://www.coursera.org/learn/algorithms-part1/lecture/wSISD/1d-range-search)

##### [Line Segment Intersection](https://www.coursera.org/learn/algorithms-part1/lecture/HeK99/line-segment-intersection)

[5 分](https://www.coursera.org/learn/algorithms-part1/lecture/HeK99/line-segment-intersection)

##### [Kd-Trees](https://www.coursera.org/learn/algorithms-part1/lecture/Yionu/kd-trees)

[29 分](https://www.coursera.org/learn/algorithms-part1/lecture/Yionu/kd-trees)

##### [Interval Search Trees](https://www.coursera.org/learn/algorithms-part1/lecture/ot9vw/interval-search-trees)

[13 分](https://www.coursera.org/learn/algorithms-part1/lecture/ot9vw/interval-search-trees)

##### [Rectangle Intersection](https://www.coursera.org/learn/algorithms-part1/lecture/mNiwq/rectangle-intersection)

[8 分](https://www.coursera.org/learn/algorithms-part1/lecture/mNiwq/rectangle-intersection)

##### ****[编程作业:](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees)****

##### [Kd-Trees](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees)

[5小时](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees)

## 编程作业: Kd-Trees

已通过 · 89/100 分

|  |  |
| --- | --- |
| **截止时间** | Pass this assignment by 二月 26, 11:59 晚上 PST |

1. [说明](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees)
2. [**我提交的作业**](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees/submission)
3. [讨论](https://www.coursera.org/learn/algorithms-part1/programming/wuF0a/kd-trees/discussions)

  Create submission

### Your Submissions

**日期**

**分数**

**通过了吗？**

1 二月 2017 在 12:13 中午

89/100

是

Kd-Trees

89/100

隐藏 评分反馈

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

ASSESSMENT SUMMARY

Compilation: PASSED

API: PASSED

Findbugs: FAILED (1 warning)

Checkstyle: FAILED (15 warnings)

Correctness: 23/27 tests passed

Memory: 8/8 tests passed

Timing: 36/42 tests passed

Aggregate score: 88.25%

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

ASSESSMENT DETAILS

The following files were submitted:

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

7.2K Feb 1 04:13 KdTree.java

2.5K Feb 1 04:13 PointSET.java

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

\* COMPILING

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

% javac PointSET.java

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

% javac KdTree.java

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

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

Checking the APIs of your programs.

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

PointSET:

KdTree:

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

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

\* CHECKING STYLE AND COMMON BUG PATTERNS

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

% findbugs \*.class

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

M P URF\_UNREAD\_FIELD UrF: The instance (or static) variable 'tree' is never read. Consider removing it from the class. At KdTree.java:[line 10]

Warnings generated: 1

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

% checkstyle \*.java

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

PointSET.java:3:8: Unused import statement for 'edu.princeton.cs.algs4.StdDraw'. [UnusedImports]

KdTree.java:4: Do not use .\* in import statements. [AvoidStarImport]

KdTree.java:16:17: The instance (or static) variable 'isOdd' must be private. [VisibilityModifier]

KdTree.java:17:17: The instance (or static) variable 'point' must be private. [VisibilityModifier]

KdTree.java:18:14: The instance (or static) variable 'left' must be private. [VisibilityModifier]

KdTree.java:19:14: The instance (or static) variable 'right' must be private. [VisibilityModifier]

KdTree.java:58:35: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

KdTree.java:89:40: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

KdTree.java:111:37: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

KdTree.java:132:33: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

KdTree.java:153: Line is longer than 128 characters (currently 131). [LineLength]

KdTree.java:153:45: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

KdTree.java:197:5: Declare instance variables after static variables but before constructors and methods. [DeclarationOrder]

KdTree.java:198:5: Declare instance variables after static variables but before constructors and methods. [DeclarationOrder]

KdTree.java:200:45: The local (or parameter) variable 'root' has the same name as an instance variable. Use a different name. [HiddenField]

Checkstyle ends with 15 errors.

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

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

\* TESTING CORRECTNESS

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

Testing correctness of PointSET

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

Running 8 total tests.

A point in an m-by-m grid means that it is of the form (i/m, j/m),

where i and j are integers between 0 and m

Test 1: Test size() by inserting n random points

(size may be less than n because of duplicates)

\* 100000 random points in a 100000-by-100000 grid

\* 100000 random points in a 10000-by-10000 grid

\* 100000 random points in a 1000-by-1000 grid

\* 100000 random points in a 100-by-100 grid

\* 100000 random points in a 10-by-10 grid

==> passed

Test 2: Test isEmpty() by checking for n = 0, 1, and 2 points

\* 0 points

\* 1 point

\* 2 points

==> passed

Test 3: Insert n random points and check contains() for random query points

\* 100000 random points in a 100000-by-100000 grid

\* 100000 random points in a 10000-by-10000 grid

\* 100000 random points in a 1000-by-1000 grid

\* 100000 random points in a 100-by-100 grid

\* 100000 random points in a 10-by-10 grid

==> passed

Test 4: Insert n random points and check nearest() for random query points

\* 1000 random points in a 100000-by-100000 grid

- failed on trial 1 of 1000

- student nearest() = (0.86872, 0.70041)

- reference nearest() = (0.85176, 0.26652)

- student distanceTo() = 0.41991044307090053

- reference distanceTo() = 0.02151192460009109

\* 1000 random points in a 10000-by-10000 grid

- failed on trial 1 of 1000

- student nearest() = (0.373, 0.699)

- reference nearest() = (0.3722, 0.3604)

- student distanceTo() = 0.31670006315124094

- reference distanceTo() = 0.0219082176363117

\* 1000 random points in a 1000-by-1000 grid

- failed on trial 1 of 1000

- student nearest() = (0.93, 0.688)

- reference nearest() = (0.927, 0.101)

- student distanceTo() = 0.5670008818335294

- reference distanceTo() = 0.02039607805437113

\* 1000 random points in a 100-by-100 grid

- failed on trial 1 of 1000

- student nearest() = (0.84, 0.05)

- reference nearest() = (0.83, 0.96)

- student distanceTo() = 0.9199999999999999

- reference distanceTo() = 0.014142135623730963

\* 1000 random points in a 10-by-10 grid

- failed on trial 1 of 1000

- student nearest() = (0.7, 0.0)

- reference nearest() = (0.7, 0.9)

- student distanceTo() = 0.9

- reference distanceTo() = 0.0

==> FAILED

Test 5: Insert n random points and check range() for random query rectangles

\* 1000 random rectangles and points in a 100000-by-100000 grid

- failed on trial 2 of 1000

- rectangle query [0.15482, 0.90331] x [0.0805, 0.46905]

- student range() does not contain (0.20004, 0.0805)

- reference range() contains (0.20004, 0.0805)

\* 1000 random rectangles and points in a 10000-by-10000 grid

- failed on trial 9 of 1000

- rectangle query [0.4382, 0.8717] x [0.2498, 0.6911]

- student range() does not contain (0.4382, 0.3867)

- reference range() contains (0.4382, 0.3867)

\* 1000 random rectangles and points in a 1000-by-1000 grid

- failed on trial 290 of 999

- rectangle query = [0.9999999999, 1.0] x [0.2739999999, 0.27400000010000003]

- student range() size = 0

- reference range() size = 1

\* 1000 random rectangles and points in a 100-by-100 grid

- failed on trial 1 of 947

- rectangle query = [0.06999999990000001, 0.0700000001] x [0.0, 1.0E-10]

- student range() size = 0

- reference range() size = 1

\* 1000 random rectangles and points in a 10-by-10 grid

- failed on trial 1 of 121

- rectangle query = [0.0, 1.0E-10] x [0.0, 1.0E-10]

- student range() size = 0

- reference range() size = 1

==> FAILED

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 with random points in a 10000-by-10000 grid

and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2

- failed on trial 7 of 10000

- student nearest() = (0.8165, 0.2132)

- reference nearest() = (0.882, 0.3957)

- student distanceTo() = 0.9059145710275335

- reference distanceTo() = 0.8918005438437454

\* 10000 calls with random points in a 1000-by-1000 grid

and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2

- failed on trial 15 of 10000

- student nearest() = (0.863, 0.945)

- reference nearest() = (0.898, 0.71)

- student distanceTo() = 0.43161672812809276

- reference distanceTo() = 0.25956309444911463

\* 10000 calls with random points in a 100-by-100 grid

and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2

- failed on trial 12 of 10000

- student nearest() = (0.6, 0.44)

- reference nearest() = (0.86, 0.9)

- student distanceTo() = 0.431740662898458

- reference distanceTo() = 0.16492422502470647

\* 10000 calls with random points in a 10-by-10 grid

and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2

- failed on trial 10 of 10000

- student nearest() = (1.0, 0.4)

- reference nearest() = (1.0, 1.0)

- student distanceTo() = 0.5385164807134504

- reference distanceTo() = 0.2236067977499789

\* 10000 calls with random points in a 1-by-1 grid

and probabilities 0.05, 0.05, 0.3, 0.2, 0.2, 0.2

- student solution has 0 non-null entries

- reference solution has 1 non-null entries

- 1 missing entry in student solution: (1.0, 0.0)

- failed on trial 7 of 10000

- rectangle = [1.0, 1.0] x [0.0, 1.0]

==> FAILED

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 with random points in a 1000-by-1000 grid

and probabilities 0.5, 0.5, 0.0, 0.0, 0.0, 0.0

\* 1000 calls with random points in a 1000-by-1000 grid

and probabilities 0.2, 0.2, 0.0, 0.6, 0.0, 0.0

\* 1000 calls with random points in a 1000-by-1000 grid

and probabilities 0.2, 0.2, 0.0, 0.0, 0.6, 0.0

- failed on trial 1 of 1000

- student range() = null

- reference range() = Iterable<Point2D> of length 0

\* 1000 calls with random points in a 1000-by-1000 grid

and probabilities 0.2, 0.2, 0.0, 0.0, 0.0, 0.6

\* 1000 calls with random points in a 1000-by-1000 grid

and probabilities 0.2, 0.2, 0.0, 0.2, 0.2, 0.2

- failed on trial 3 of 1000

- student range() = null

- reference range() = Iterable<Point2D> of length 0

==> FAILED

Test 8: Test whether two PointSET objects can be created at the same time

==> passed

Total: 4/8 tests passed!

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

Testing correctness of KdTree

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

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

\* Non-degenerate points: no two points (or rectangles) share either an

x-coordinate or a y-coordinate

\* Distinct points: no two points (or rectangles) share both an

x-coordinate and a y-coordinate

\* General points: no restrictions on the x-coordinates or y-coordinates

of the points (or rectangles)

A point in an GRID-by-GRID grid means that it is of the form (i/GRID, j/GRID),

where i and j are integers between 0 and GRID

Running 19 total tests.

Test 1a: Insert n non-degenerate points and check size() after each insertion

\* 50000 random non-degenerate points in a 100000-by-100000 grid

\* 5000 random non-degenerate points in a 10000-by-10000 grid

\* 500 random non-degenerate points in a 1000-by-1000 grid

\* 50 random non-degenerate points in a 100-by-100 grid

\* 10 random non-degenerate points in a 10-by-10 grid

\* 1 random non-degenerate points in a 1-by-1 grid

==> passed

Test 1b: Insert n distinct points and check size() after each insertion

\* 100000 random distinct points in a 100000-by-100000 grid

\* 100000 random distinct points in a 10000-by-10000 grid

\* 100000 random distinct points in a 1000-by-1000 grid

\* 10000 random distinct points in a 100-by-100 grid

\* 100 random distinct points in a 10-by-10 grid

\* 1 random distinct points in a 1-by-1 grid

==> passed

Test 1c: Insert n general points and check size() after each insertion

\* 100000 random general points in a 100000-by-100000 grid

\* 100000 random general points in a 10000-by-10000 grid

\* 100000 random general points in a 1000-by-1000 grid

\* 100000 random general points in a 100-by-100 grid

\* 100000 random general points in a 10-by-10 grid

\* 10 random general points in a 1-by-1 grid

==> passed

Test 2: Test size() and isEmpty() for n = 0, 1, and 2 points

\* 0 points

\* 1 point

\* 2 points

==> passed

Test 3a: Insert n non-degenerate points and call contains() with random query points

\* 50000 random non-degenerate points in a 100000-by-100000 grid

\* 5000 random non-degenerate points in a 10000-by-10000 grid

\* 500 random non-degenerate points in a 1000-by-1000 grid

\* 50 random non-degenerate points in a 100-by-100 grid

\* 5 random non-degenerate points in a 10-by-10 grid

\* 1 random non-degenerate points in a 1-by-1 grid

==> passed

Test 3b: Insert n distinct points and call contains() with random query points

\* 100000 random distinct points in a 100000-by-100000 grid

\* 100000 random distinct points in a 10000-by-10000 grid

\* 100000 random distinct points in a 1000-by-1000 grid

\* 10000 random distinct points in a 100-by-100 grid

\* 100 random distinct points in a 10-by-10 grid

\* 1 random distinct points in a 1-by-1 grid

==> passed

Test 3c: Insert n general points and call contains() with random query points

\* 10000 random general points in a 1000-by-1000 grid

\* 10000 random general points in a 100-by-100 grid

\* 10000 random general points in a 10-by-10 grid

\* 10000 random general points in a 1-by-1 grid

==> passed

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

==> passed

Test 5a: Insert n non-degenerate points and call range() for n random query rectangles

\* 5000 random non-degenerate points and rectangles in a 100000-by-100000 grid

\* 5000 random non-degenerate points and rectangles in a 10000-by-10000 grid

\* 500 random non-degenerate points and rectangles in a 1000-by-1000 grid

\* 50 random non-degenerate points and rectangles in a 100-by-100 grid

\* 5 random non-degenerate points and rectangles in a 10-by-10 grid

==> passed

Test 5b: Insert n distinct points and call range() for n random query rectangles

\* 5000 random distinct points and rectangles in a 100000-by-100000 grid

\* 5000 random distinct points and rectangles in a 10000-by-10000 grid

\* 1000 random distinct points and rectangles in a 1000-by-1000 grid

\* 1000 random distinct points and rectangles in a 100-by-100 grid

\* 5 random distinct points and rectangles in a 10-by-10 grid

==> passed

Test 5c: Insert n general points and call range() for n random query rectangles

\* 5000 random general points and rectangles in a 10000-by-10000 grid

\* 5000 random general points and rectangles in a 1000-by-1000 grid

\* 5000 random general points and rectangles in a 100-by-100 grid

\* 5000 random general points and rectangles in a 10-by-10 grid

\* 5000 random general points and rectangles in a 1-by-1 grid

==> passed

Test 5d: Insert n points and call range() for tiny rectangles enclosing each point.

\* 4000 tiny rectangles and 4000 points in a 100000-by-100000 grid

\* 4000 tiny rectangles and 4000 points in a 10000-by-10000 grid

\* 4000 tiny rectangles and 4000 points in a 1000-by-1000 grid

\* 4000 tiny rectangles and 4000 points in a 100-by-100 grid

\* 4000 tiny rectangles and 4000 points in a 10-by-10 grid

==> passed

Test 6a: Insert n non-degenerate points and call nearest() with random query points

\* 50000 random non-degenerate points in a 100000-by-100000 grid

\* 5000 random non-degenerate points in a 10000-by-10000 grid

\* 500 random non-degenerate points in a 1000-by-1000 grid

\* 50 random non-degenerate points in a 100-by-100 grid

\* 5 random non-degenerate points in a 10-by-10 grid

==> passed

Test 6b: Insert n distinct points and call nearest() with random query points

\* 50000 random distinct points in a 100000-by-100000 grid

\* 10000 random distinct points in a 10000-by-10000 grid

\* 10000 random distinct points in a 1000-by-1000 grid

\* 5000 random distinct points in a 100-by-100 grid

\* 50 random distinct points in a 10-by-10 grid

==> passed

Test 6c: Insert n general points and call nearest() with random query points

\* 10000 random general points in a 1000-by-1000 grid

\* 10000 random general points in a 100-by-100 grid

\* 10000 random general points in a 10-by-10 grid

==> passed

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

contains(), range(), and nearest() with probabilities

(p1, p2, p3, p4, p5, p6), respectively

\* 20000 calls with non-degenerate points in a 100000-by-100000 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

\* 20000 calls with non-degenerate points in a 10000-by-10000 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

\* 20000 calls with non-degenerate points in a 1000-by-1000 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

\* 20000 calls with non-degenerate points in a 100-by-100 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

\* 20000 calls with non-degenerate points in a 10-by-10 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

\* 20000 calls with non-degenerate points in a 1-by-1 grid

and probabilities (0.3, 0.05, 0.05, 0.1, 0.2, 0.2)

==> passed

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

contains(), range(), and nearest() with probabilities

(p1, p2, p3, p4, p5, p6), respectively

\* 20000 calls with distinct points in a 100000-by-100000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with distinct points in a 10000-by-10000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with distinct points in a 1000-by-1000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with distinct points in a 100-by-100 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with distinct points in a 10-by-10 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with distinct points in a 1-by-1 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

==> passed

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

contains(), range(), and nearest() with probabilities

(p1, p2, p3, p4, p5, p6), respectively

\* 20000 calls with general points in a 100000-by-100000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with general points in a 10000-by-10000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with general points in a 1000-by-1000 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with general points in a 100-by-100 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with general points in a 10-by-10 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

\* 20000 calls with general points in a 1-by-1 grid

and probabilities (0.3, 0.05, 0.05, 0.2, 0.2, 0.2)

==> passed

Test 8: Intermixed sequence of calls to insert(), isEmpty(), size(),

contains(), range(), and nearest() with probabilities

(p1=0, p2, p3, p4, p5, p6), respectively

(data structure with 0 points)

\* 1000 calls with no points in a 1000-by-1000 grid

and probabilities (0.0, 0.5, 0.5, 0.0, 0.0, 0.0)

\* 1000 calls with no points in a 1000-by-1000 grid

and probabilities (0.0, 0.2, 0.2, 0.6, 0.0, 0.0)

\* 1000 calls with no points in a 1000-by-1000 grid

and probabilities (0.0, 0.2, 0.2, 0.0, 0.6, 0.0)

\* 1000 calls with no points in a 1000-by-1000 grid

and probabilities (0.0, 0.2, 0.2, 0.0, 0.0, 0.6)

\* 1000 calls with no points in a 1000-by-1000 grid

and probabilities (0.0, 0.2, 0.2, 0.2, 0.2, 0.2)

==> passed

Total: 19/19 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 RectHV objects).

Maximum allowed memory is 312n + 192 bytes.

n student (bytes) reference (bytes)

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

=> passed 1 248 160

=> passed 2 336 288

=> passed 5 600 672

=> passed 10 1040 1312

=> passed 25 2360 3232

=> passed 100 8960 12832

=> passed 400 35360 51232

=> passed 800 70560 102432

==> 8/8 tests passed

Total: 8/8 tests passed!

Estimated student memory (bytes) = 88.00 n + 160.00 (R^2 = 1.000)

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

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

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

\* TIMING

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

Timing PointSET

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

Running 14 total tests.

Inserting n points into a PointSET

n ops per second

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

=> passed 160000 1692786

=> passed 320000 1862042

=> passed 640000 1512420

=> passed 1280000 1244084

==> 4/4 tests passed

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

n ops per second

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

=> passed 160000 506580

=> passed 320000 481779

=> passed 640000 433416

=> passed 1280000 386956

==> 4/4 tests passed

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

n ops per second

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

=> passed 10000 3959

=> passed 20000 1911

=> passed 40000 875

==> 3/3 tests passed

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

n ops per second

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

=> passed 10000 1070

=> passed 20000 527

=> passed 40000 255

==> 3/3 tests passed

Total: 14/14 tests passed!

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

Timing KdTree

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

Running 28 total tests.

Inserting n points into a 2d tree. The table gives the average number of calls to methods

in RectHV and Point per call to insert().

Point2D

n ops per second RectHV() x() y() equals()

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

=> passed 160000 1057429 0.0 44.3 42.3 21.6

=> passed 320000 1264607 0.0 45.1 43.1 22.0

=> passed 640000 1052334 0.0 48.1 46.1 23.5

=> passed 1280000 881447 0.0 52.3 50.3 25.6

==> 4/4 tests passed

Performing contains() queries after inserting n points into a 2d tree. The table gives

the average number of calls to methods in RectHV and Point per call to contain().

Point2D

n ops per second x() y() equals()

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

=> passed 10000 538800 18.5 17.5 18.0

=> passed 20000 560742 19.7 18.7 19.2

=> passed 40000 557400 21.8 20.8 21.3

=> passed 80000 518360 22.0 21.0 21.5

=> passed 160000 494459 23.2 22.2 22.7

=> passed 320000 457902 25.0 24.0 24.5

=> passed 640000 408243 25.7 24.7 25.2

=> passed 1280000 408612 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() contains() x() y()

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

=> passed 10000 233455 49.4 31.1 97.2 57.4

=> passed 20000 186849 51.7 32.6 102.2 64.1

=> passed 40000 252085 63.9 39.3 122.5 71.7

=> passed 80000 244678 66.1 40.7 126.5 74.6

=> passed 160000 244336 69.0 42.5 134.5 83.3

=> passed 320000 190073 66.0 40.2 125.7 74.9

=> passed 640000 184413 71.0 43.3 135.5 83.1

=> passed 1280000 136375 77.7 47.0 146.0 83.2

==> 8/8 tests passed

Performing nearest() queries after inserting n points into a 2d tree. The table gives

the average number of calls to methods in RectHV and Point per call to nearest().

Point2D RectHV

n ops per second distanceSquaredTo() distanceSquaredTo() x() y()

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

=> passed 10000 88792 190.2 232.1 758.0 716.0

=> passed 20000 90787 183.2 226.1 718.1 700.4

=> FAILED 40000 45240 419.4 504.6 (1.7x) 1680.0 (2.1x) 1555.3 (1.9x)

=> FAILED 80000 47878 372.8 454.4 (1.5x) 1427.0 (1.8x) 1465.7 (1.8x)

=> FAILED 160000 24978 (0.8x) 757.1 (1.3x) 914.1 (3.0x) 3016.5 (3.8x) 2812.7 (3.5x)

=> FAILED 320000 27190 534.5 657.7 (2.2x) 2090.2 (2.6x) 2125.0 (2.7x)

=> FAILED 640000 20891 823.0 (1.4x) 1004.3 (3.3x) 3324.5 (4.2x) 3184.2 (4.0x)

=> FAILED 1280000 10450 (0.5x) 1329.3 (2.2x) 1620.1 (5.4x) 5494.7 (6.9x) 5134.2 (6.4x)

==> 2/8 tests passed

Total: 22/28 tests passed!

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

1 二月 2017 在 11:39 中午

74/100

否

Kd-Trees

74/100

显示 评分反馈

1 二月 2017 在 11:06 上午

77/100

否

Kd-Trees

77/100

显示 评分反馈

31 一月 2017 在 10:59 晚上

36/100

否

31 一月 2017 在 10:57 晚上

5/100

否

31 一月 2017 在 10:55 晚上

5/100

否