Homework 1

CS 252, Winter-Spring 2005

Due in class on **Monday, March 7, 2005**

Hand-in Instructions:

- Hand in the written portions of the assignment to the TA in class.
- E-mail a gzipped tar file containing the source files for your solution to Problem 5 to glencora@cs.brown.edu as an *attachment*.

Late Policy:

- Hand in this homework by **3:00 pm, Monday, March 7, 2005** to receive **full credit**.
- You can also hand in this homework after the deadline to the TA's mailbox (labelled Glencora Borradaile on the 4th floor by the restroooms). However, you will be assessed a **5% penalty for each late day**.

Problem 1 (10 points)

1. Draw a segment tree for the following intervals:

   $a = (1, \infty)$, $b = (-8, 0)$, $c = (1, 6)$, $d = (4, 8)$, $e = (-\infty, 0)$.
2. On copies of the above drawing, illustrate the execution of point enclosure queries for the following values of the query point x:

   $x = 1$, $x = -5$, $x = 5$, and $x = 0.5$.

Problem 2 (10 points)

1. Describe the difference between a segment-range tree (SR-tree) and a range-segment tree (RS-tree) and what problems each may be used to solve.
2. For each of the following problems, give the data structure would you use to efficiently solve it. Briefly analyze its space requirement and query time.
   (a) A two-dimensional range searching problem?
   (b) A two-dimensional point enclosure problem?
   (c) A one-dimensional interval intersection problem?

Problem 3 (20 points)

1. Draw the **separator tree** data structure (chain method) for the subdivision in Figure 1.
2. On copies of your drawing, illustrate the execution of two point locations queries, for two points of your choice that are in different regions of the subdivision.

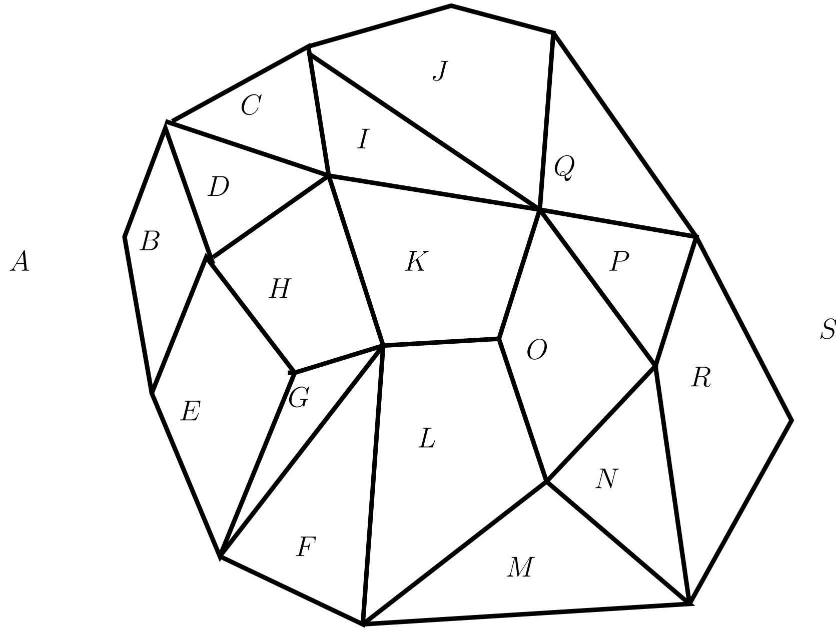

Figure 1: Subdivision for Problem 3.

Problem 4 (60 points)

In this problem, you will be using Java and JDSL to implement a priority search tree to handle three-sided queries. The archive `hw1.tar` provides the interface files and an example input file.

Part 1

Implement two classes to do three-sided range searches on `Point2D` objects:

`OpenBottom2DRangeSearch`**:** Searches between two x coordinates and below a y coordinate.

`OpenRight2DRangeSearch`**:** Searches between two y coordinates and above an x coordinate.

The priority search tree can be implemented once: For example, you could implement the interface `PrioritySearchTree`, provided to you in the `hw1` package, in a class named `PrioritySearchTreeImpl`. This implementation should not depend on the type of object (ie, it is not for 2D integer points).

As a starting point, `OpenBottom2DRangeSearch` and `OpenRight2DRangeSearch` could implement the interface `ThreeSided2DRangeSearch` provided to you in the `hw1` package.

Part 2

Write a tester for your priority tree class. The tester should include brute-force (linear-time) implementations for the two three-sided range query classes that you created in part 2. The tester compares the outputs of the brute force and priority-tree implementations for several test cases: random, fixed points and ranges, boundary cases (with small n and small k).

The tester should be able to read in an input file of data points such as `input_file.txt`. The first line of this file has a number representing the number of data points. Each line thereafter has two space delimited points representing the x and y coordinates of the point. Also include a `main` function for command-line invocation to prompt the user to enter the location of such a data file, and ask for what type of search will be performed (OpenBottom or OpenRight) and what the input query range is.

Part 3

Time your `OpenRight2DRangeSearch` program for the preprocessing and query time dependence on n (using random test cases). Test enough values of n to illustrate the expected time dependence on n. You should plot the results. To do this correctly, consider the following:

- Remove the dependence on k: one way to do this is to have $k = O(1)$ or $k = O(n)$. This can be done by using input points that are uniformly distributed over some region and choosing your query range appropriately.
- Time the query separately (ie, after the preprocessing step): `System.currentTimeMillis()` can be used to query the run time.
- Since the entire process is very efficient, several identical examples should be run in succession to get a significant time.
- Garbage collection may overpower the run time of the algorithm. Garbage collection can be forced through a call to `System.gc` and should be made before any timing begins.

Notes:

- Use javadoc to document the purpose, parameters, return values, and exceptions of your classes' members.
- Use JDSL's `GeomTester2D` for geometric comparisons, and use `GeomConstructor2D` for constructing new geometric objects. If you need comparisons or constructions not included, extend these interfaces as needed. Don't forget to pass any instances of `GeomTester2D` or `GeomConstructor2D` needed into the constructors of `OpenBottom2DRangeSearch` and `OpenRight2DRangeSearch`, as well as those of any other necessary classes.
- Call your top-level package "`prioritysearch`". It is not necessary to divide the package into separate `ref` and `api` subpackages for this assignment.