Version 1.0 ***due:*** *Sun, Nov 24nd @ 11:59pm*

1 Introduction

In this assignment you will explore an application of binary search trees to develop efficient algorithms in computational geometry, which also happen to be useful in various database queries. First you will implement an ordered table using binary search trees that conforms to the ordered table signature. Then you will create a data structure that enables answering efficiently queries that report the number of 2-dimensional points within a specified rectangle. A common application of such 2-dimensional *range queries* is for counting the number of elements in a data set that are within two ranges. For example if a database maintained people by height and weight, one could ask for the number of people between 95 and 100 pounds who are between $5'6''$ and $5'9''$. Range queries can also return all elements in the range, but we don't require this for the homework.

Note: For this lab, it is a good idea to test your code after the completion of each task. Instructions for testing are at the end of this document.

2 Files

After downloading the assignment tarball from Autolab, extract the files from it by running

```
tar -xvf rangelab-handout.tgz
```

from a terminal window. You should see the following files:

1. `sources.cm`
2. `support/`
3. * `MkBSTOrderedTable.sml`
4. * `MkRangeCount.sml`
5. **FINISH WITH SUBMISSION SCRIPT**

You should only modify the last 2 files, denoted by *. Additionally, you should create a file called `written.pdf` which contains the answers to the written part of the assignment.

3 Submission

To submit your assignment: open a terminal, `cd` to the `rangelab-handout` folder, and run `make`. Alternatively, run `make package`, open the Autolab webpage and submit the `handin.tgz` file via the "Handin your work" link.

4 Ordered Tables

The abstract data types Table and Set allow for implementations based on types that have no natural ordering. Implementations can be based, for example, on hash tables or on binary search trees. The `BSTTable` implementation in the 15-210 library is based on a binary search tree and contains only the core tree functions needed to implement tables and sets. Tables that have keys from a totally ordered universe of elements $\mathbb{K}$, however, can have additional operations that take advantage of the ordering. You will implement an ordered table based on binary search trees that conforms to the abstract data type *ordered table*. Then, in the next part, you will use the operations provided by ordered tables to support efficient range queries.

Documentation for the BST signature is on the course website at `http://www.cs.cmu.edu/~15210/docs/`. Your implementations should be as efficient as possible.

Task 4.1 (6%). Implement the functions

```
fun first (T : 'a table) :  (key * 'a) option
fun last (T : 'a table) :  (key * 'a) option
```

Given an ordered table T, `first T` should evaluate to `SOME (k,v)` iff $(k, v) \in T$ and k is the minimum key in T. Analagously, `last T` should evaluate to `SOME (k,v)` iff $(k, v) \in T$ and k is the maximum key in T. Otherwise, they evaluate to `NONE`.

Task 4.2 (8%). Implement the functions

```
fun previous (T : 'a table) (k :  key) :  (key * 'a) option
fun next (T : 'a table) (k :  key) :  (key * 'a) option
```

Given an ordered table T and a key k, `previous T k` should evaluate to `SOME (k',v)` if $(k', v) \in T$ and k' is the greatest key in T strictly less than k. Otherwise, it evaluates to `NONE`. Similarly, `next T k` should evaluate to `SOME (k',v)` iff k' is the least key in T strictly greater than k.

Task 4.3 (2%). Implement the function

```
fun join (L : 'a table, R : 'a table) :  'a table
```

Given ordered tables L and R, where all the keys in L are strictly less than those in R, `join (L, R)` should evaluate to an ordered table containing all the keys from *both* L and R.

Task 4.4 (2%). Implement the function

```
fun split (L : 'a table, k :  key) :  'a table * 'a option * 'a table
```

Given an ordered table T and a key k, `split` should evaluate to a triple consisting of

1. an ordered table containing every $(k', v) \in T$ such that $k' < k$,

2. `SOME v` if $(k, v) \in T$ and `NONE` otherwise, and
3. an ordered table containing every $(k', v) \in T$ such that $k' > k$.

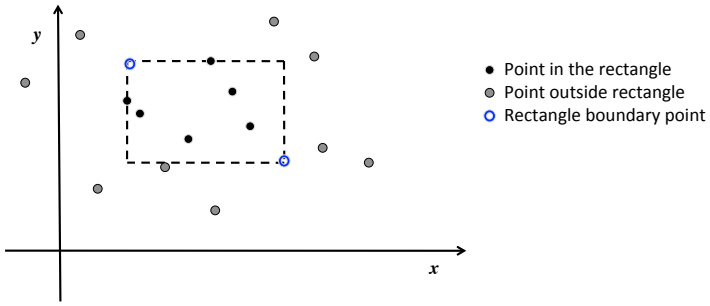
Task 4.5 (7%). Implement the function

```
fun getRange (T : 'a table) (low : key, high : key) : 'a table
```

Given an ordered table T and keys l and h, `getRange T (l, h)` should evaluate to an ordered table containing every $(k, v) \in T$ such that $l \leq k \leq h$.

5 Range Query

Suppose we are given a set of points $P = \{p_1, p_2, p_3, \cdots, p_n\}$, where $p_i \in \mathbb{Z}^2$. That is to say, these points are on the integer lattice. We want to be able to answer questions about the points within axis-aligned rectangular regions quickly. For example, we may want the number of points in the region, or the total "mass" of the points in the region. We will define a region by two diagonally opposing corner points of the rectangle: the upper-left corner and lower-right corner.



For example, given the points $\{(0,0),(1,2),(3,3),(4,4),(5,1)\}$, there are 2 points in the rectangle defined by $\{(2,4),(4,2)\}$ and 3 points in $\{(1,3),(5,1)\}$.

There is an obvious $O(n)$ algorithm to answer such queries. In this lab, you will need to create a data structure (using ordered tables) that will allow for $O(\log n)$ work queries on a given set of points.

Common to many computational geometry problems is the concept of a *sweep line*: Each point is considered when a vertical line crosses the point as it sweeps across the plane by increasing x coordinate (or by a horizontal line that sweeps across the point set by increasing y coordinate).

In this task, you will build a data structure from a sequence of two-dimensional points that will enable us to answer range queries efficiently. You might want to consider using the sweep line concept when building your data structure. In particular consider incrementally constructing your data structure - given a valid data structure for the first i points seen, how can you update this structure to be valid for the first $i+1$ many points? It might also be helpful to first think about how to answer a three sided query that given $(x_{top}, y_{bot}, y_{top})$ returns the number of points inside the rectangle that goes out to negative infinity on the x coordinate.

Task 5.1 (25%). In the `MkRangeQuery` functor, define the `countTable` type and implement the function

```
fun makeCountTable:  point seq -> countTable
```

The type `point` is defined to be `OrdTable.Key.t * OrdTable.Key.t` where `OrdTable` is an ordered table structure provided to you. You should choose the type of `countTable` such that you can implement `count` (range queries) in $O(\log n)$ work and span.
For full credit, your `makeCountTable` must run within $O(n \log n)$ expected work.

Note: You may assume that the n input points have unique x coordinates and unique y coordinates and these integer coordinates are of type `int`.

Task 5.2 (10%). Briefly describe how you would parallelize your code so that it runs in $O(\log^2 n)$ span. Does the work remain the same? You don't need to formally prove the bounds, just briefly justify them.

Task 5.3 (5%). What is the expected space complexity of your `countTable` in terms of n the number of input points? That is, how many nodes in the underlying binary search tree(s) does your `countTable` use in expectation? Explain in a few short sentences.

Task 5.4 (25%). Implement the function

```
count:  countTable -> point * point -> int
```

As described earlier, `count T ((x_1,y_1), (x_2, y_2))` will report the number of points within the rectangle with the top-left corner (x_1, y_1) and bottom-right corner (x_2, y_2). Your function should return the number of points *within and on* the boundary of the rectangle. You may find the `OrdTable.size` function useful here. Your implementation should have $O(\log n)$ work and span.

6 Testing

You do *not* have to submit test cases for this homework. However, it is in your best interest to test your code thoroughly before submission. A testing framework has been provided for you to make it easy to test your code.

In the file `Tests.sml`, add your tests to the appropriate list (look at the sample test cases for reference). After defining your test cases, you may run them from the REPL with

```
Tester.testFirst ();
Tester.testLast ();
Tester.testPrev ();
Tester.testNext ();
Tester.testJoin ();
Tester.testSplit ();
Tester.testRange ();
Tester.testCount ();
```