Problem 1. You are given the coordinates of three points in $\mathbb{R}^2$. Describe an algorithm to calculate in constant time the area of the triangle that has the three points as vertices. You should note that $\sqrt{x}$ is not an atomic operation of the real-RAM model.

Solution. Let $A = (x_1, y_1), B = (x_2, y_2), C = (x_3, y_3)$ be the three points in the $\mathbb{R}^2$ plane.

$$S_\triangle = \frac{1}{2}\left|\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}\right| = \frac{1}{2}|x_1y_2 - x_2y_1 - x_1y_3 + x_3y_1 + x_2y_3 - x_3y_2|.$$

We just have to deal with the operation of taking the absolute value of real numbers. Using the power of the *floor* atomic operation, we may decide whether to invert the sign ahead of the evaluation result. Suppose that we are taking the absolute value of x, we let $y \leftarrow \lfloor x \rfloor$, and $x \leftarrow -x$ if $y < 0$ (comparison is an atomic operation, remind that we are loading a 0 into a register and comparing y to that). Then, all other calculations are just simple atomic arithmetic operations.

Problem 2. Let S be a set of n vertical line segments in $\mathbb{R}^2$ (i.e. each segment has the form $x \times [y_1, y_2]$. Also, let P be a set of m points in $\mathbb{R}^2$. For each segment $s \in S$, we want to output a pair (s, p) where p is the first point in P that is hit by s if s moves left; if p does not exist, output (s, nil).
Use the planesweep approach to design an algorithm to solve the above problem in $O(n \log n + m \log m)$ time, subject to the constraint that your algorithm should sweep a horizontal line from $y = -\infty$ to $y = \infty$. You may assume that no two segments in S share the same x-coordinate.

Solution.

We first sort the points in P by their y-coordinate in ascending order. Then, sort all endpoints of the segments by their y-coordinate in ascending order, taking note of whether it's the left endpoint (beginning of a line segment) or the right endpoint (end of a line segment). This is done in $O(n \log n + m \log m)$ time.

We then are ready to sweep a line through the points and segments endpoints. Maintain a sweepline status structure recording all segments crossing the sweepline. We shall implement a data structure supporting the following query with respect to the event queue Q.

- Insert a segment and record its x-coordinate;
- Delete a segment;
- Update with query (x_0, y_0) for those segments with x-coordinate larger or equal to x_0 in the sweepline status and covering the point y_0.

The problem is to maintain a BBST to support the above queries. We can assign each node in the BBST with a *lazytag*, which allows us to not iterate through all possible segments. For each insertion, we release the *lazytag* on the parent node and pushdown to its children, then we are assured to insert the corresponding segment safely. On segment deletion, we are ready to finish the line-sweeping for the respective segment, reporting the point hit by the segment. This can be done by referring to the lazytags on the nodes along the BBST path. These operations can be done in $O(\log n)$ each.

Overall, the algorithm can be finished within $O(n \log n + m \log m)$ time.

Here is a solution of an output-sensitive algorithm similar to the line-intersection algorithm taught in class.

We consider every point in P shoots a ray toward the positive x-axis, these rays will possibly intersect with the vertical segments in S. Let the number of intersections be k. The time complexity of this proposed algorithm is $O(n \log n + m \log m + k)$.

We first sort the points in P by their y-coordinate in ascending order. Then, sort all endpoints of the segments by their y-coordinate in ascending order, taking note of whether it's the left endpoint (beginning of a line segment) or the right endpoint (end of a line segment). This is done in $O(n \log n + m \log m)$ time.

We then are ready to sweep a line through the points and segments endpoints. Maintain a sweepline status structure recording all segments crossing the sweepline. We shall implement a data structure supporting the following query with respect to the event queue Q.

- Insert a segment and record its x-coordinate;
- Delete a segment;
- Report all segments having their x-coordinate larger than or equal to a given query x_0, and update their values respectively.

This can be done by a balanced binary search tree (e.g. Splay). Whenever we encounter a left endpoint, insert the segment number with its x-coordinate in the tree; and delete it when we have arrived to its endpoint, these operations can be done in $O(\log n)$ time. The report operation is simply an augmented ordinary successor search which can be done in $O(\#$ of reported segments$) = O(k)$ time, since the total number of reported segments is simply k.

Then the algorithm can be done in $O(n \log n + m \log m + k)$ time in total.

Problem 3. Let S be a set of n real numbers. Each number $v \in S$ is associated with a real valued weight. Given a range $[x, y]$, a query returns an element in $S \cap [x, y]$ with the maximum weight. For example, if $S = \{(1, 15), (3, 7), (7, 12), (10, 9)\}$, where each pair has the form $(v, weight(v))$. Then, a query with range $[2, 15]$ returns $(7, 12)$. Design a data structure to answer such queries in $O(\log n)$ time. Your structure should also support insertions and deletions in $O(\log n)$ time.

Solution. A BBST (e.g. Splay) is used to support such queries. The key observation is that each rotate operation only affect a small number of nodes, and for any of these nodes $v \in V$, the maximum value in $subtree(v)$ can still be easily maintained in $O(1)$ time accompanied with the splay operation.

Problem 4. Consider again Problem 2. Design another planesweep algorithm to solve the above problem in $O(n \log n + m \log m)$ time. This time, your algorithm must sweep a vertical line from $x = -\infty$ to $x = \infty$. You may assume that no two points in P have the same y-coordinate.

Solution. We utilize the data structure as described in Problem 3, where S consists of the pair (y, x) corresponding to the $(v, weight(v))$ element form of the data structure described above.

We first sort all points in P by their x-coordinate in ascending order. Then, sort all vertical segments in S by their x-coordinate in ascending order. These two operations are done in

$O(n \log n + m \log m)$ time. A line ℓ is ready to sweep from $x = -\infty$ to $x = \infty$. Whenever we encounter a point p, we insert it to the data structure in $O(\log m)$ time. We do not have to delete these points inserted. Furthermore, whenever a vertical segment $v = x \times [y_1, y_2]$ is encountered, we make queries to the data structure with range $[y_1, y_2]$, and get the maximum x lying in the range. It is ensured that the point reported is the one nearest, on the left to the vertical segment, answering in $O(\log m)$ time for each segment.

The overall running time is $O(n \log n + m \log m)$ as required.

Problem 5. Let S be a set of n disjoint line segments in $\mathbb{R}^2$ (these segments can have arbitrary "slopes"), and P be a set of m points in $\mathbb{R}^2$ such that no point in P falls on any segment in S. For each point $p \in P$, we want to output the segment $s \in S$ that is immediately above p, namely, s is the first segment hit by p if p moves up. Design an algorithm to achieve the purpose in $O(n \log n + m \log m)$ time.

Solution. We apply the planesweep approach and sweep a horizontal line from $y = -\infty$ to $y = \infty$.

Carefully note that all segments are disjoint and their relative order viewed from the up to the bottom is always preserved along the sweepline. This easen the maintanence for the event queue (note that we can maintain the sweep-line status "on the fly" given the x-coordinate of the current sweep line by storing the line equation).

We first sort the points in P by their x-coordinate in ascending order. Then, sort all endpoints of the segments by their x-coordinate in ascending order, taking note of whether it's the left endpoint or tthe right endpoint. This can be done in $O(n \log n + m \log m)$ time.

We then are ready to sweep a line through the points and segments endpoints. The sweep line status can be maintained by using a BBST which supports the following operations:

- Insert a new line segment;
- Delete an existing line segment;
- Report the line segment immediately above a given y-coordinate at $x = x_0$;

Whenever we encounter a left endpoint, insert the segment; and delete it when we have arrived to its endpoint, this can be done in $O(\log n)$ time. When we encounter a point $p \in P$, we make a query and obtain the segment $s \in S$ that is immediately above p. This is also done in $O(\log n)$ time.

Problem 6. Let S be a set of n disjoint line segments in the plane, and let p be a point not on any of the line segments in S. We want to determine all line segments of S that p can see, that is, all line segments of S that contain some point q so the segment pq does not intersect any segment in S (except at q, of course). Give an $O(n \log n)$ time algorithm to solve the problem. For example, in the following figure, you should output all segments but s_4 and s_6.

Solution. We first sort all the endpoints of the segments by counterclockwise order from p. We can either use the cross product or the `atan2` function in the C++ `cmath` library to perform this operation.

Our planesweep approach is then modified as if we are dynamically shooting a ray to a specified angle from p. Whenever we have encountered a left/right endpoint of a segment, we add/remove it to the sweepline status like what we have done in Problem 5. The only difference is that we

always make query to the sweepline status whenever an event is encountered so that we can get the nearest line segment with respect to p from the perspective of a polar angle θ. We then mark this reported line segment as visible. The insertion/delection/query all takes $O(\log n)$ each, and thus the overall running time of this algorithm is $O(n \log n)$.