You are given a set H of horizontal line segments and a set V of vertical line segments such that $|H| + |V| = n$. Each horizontal line segment is specified by two x-coordinates and one y-coordinate $\{x_{i_1}, x_{i_2}, y_i\}$ and each vertical line segment is specified by two y-coordinates and one x-coordinate $\{y_{i_1}, y_{i_2}, x_i\}$.

In all three subparts, you may assume that all of the x and y coordinates are distinct; in other words, no pair of horizontal lines intersect and no pair of vertical lines intersect.

(a) Firstly, suppose that each vertical line segment is unbounded from above and below. In other words, each vertical line segment stretches from $-\infty$ to $+\infty$ in its y-coordinate.

Describe an $O(n \log n)$ algorithm to return the number of intersections between line segments in H and *lines* in V.

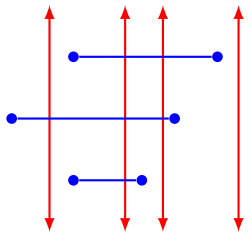

Figure 1: *There are six intersections between the horizontal line segments and the vertical lines.*

Hint. *Sort...*

(b) Now, suppose that each vertical line is bounded from above. In other words, each vertical line has a top endpoint but each vertical line stretches downwards to $-\infty$. We call these *rays* and each ray can be specified by one x-coordinate and one y-coordinate $\{x_i, y_i\}$.

Describe an $O(n \, (\log n)^2)$ algorithm that returns the number of intersections between line segments in H and vertical *rays* in V.

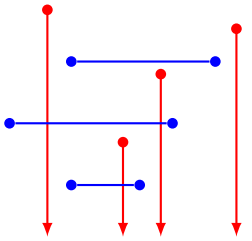

Figure 2: *There are three intersections between the horizontal line segments and the vertical rays.*

(c) We now revisit the original problem. Suppose that each vertical ray is bounded from below. In other words, each vertical ray has a top and bottom endpoint. Each of these vertical line segments can be specified by one x-coordinate and two y-coordinates $\{x_i, y_{i_1}, y_{i_2}\}$.

Describe an $O(n(\log n)^3)$ algorithm that returns the number of intersections between line segments in H and line segments in V.

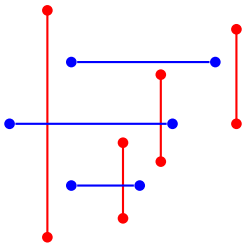

Figure 3: *There are three intersections between the horizontal line segments and the vertical line segments.*

Note. *You can obtain an $O(n(\log n)^2)$ and $O(n \log n)$-time algorithm by doing a pre-processing step, either in part (b) or part (c).*

Rubric.

- Your solution should clearly outline which subpart you're answering.
- If your solution is a simple modification of a previous solution, you do not need to restate the solution; you can simply refer to the previous part in your solution.
- As usual, you should argue the correctness of the algorithm and its time complexity.
- This task will form part of the portfolio.
- Ensure that your argument is clear and keep reworking your solutions until your lab demonstrator is happy with your work.

Solution.

(a) We'll define a set list of events, which will either be a starting point, ending point or a vertical (unbounded) line. We will store these in a list and sort them by their x coordinates. We now initialise a AVL tree of events sorted by their y coordinates. Then, for each event in order, we run the following:

- *If event is a starting point:* add the corresponding horizontal line segment to the AVL tree;
- *If event is an ending point:* remove the corresponding horizontal line segment from the AVL tree;
- *If event is an vertical line:* add the current number of elements in the AVL tree to the total intersection points.

Since we are sorting this first, our algorithm will be $O(n \log n)$, then it's a matter of handling each event, which will be $O(n \log n)$, thus our final algorithm will be $O(n \log n)$.

(b) Using a similar solution to before, we can add an additional step for when the event is a ray (no longer a vertical line). If the event is a ray, count the number of lines in the AVL tree that are below the given ray's endpoint's y value. This additional check is $O(\log n)$ (assuming we have a count in each of the subtrees of it's nodes), so we will have $O(n \log^2 n)$.

(c) Similar to before, we create a sorted list of events and cycle through them. However, if we come across a vertical line segment, we need to perform a range check for both the top and the bottom, counting everything that falls between those values (again assuming we have stored the count of nodes in the subtree). This additional check will be $O(\log n)$ and hence the time complexity of this will be $O(n \log^3 n)$.