Question 7

(a)

This can be solved using a divide and conquer problem. First you can sort by the x value and then recursively partition the set of intervals in half. This gives a $O(n \log n)$ runtime.

You then need to merge them back together at each recursive call to calculate the total number of overlaps contained. As you go you can do merge sort to then sort it by Y coordinates. Which can be done in O(n) for each recursive call.

We know the left half of the partition has X values smaller than those of the right partition. When we subsort each side by Y we then can then scan over all of the intervals in linear time comparing the y values on the left side to those on the right side. If the y value on the left side is bigger than those on the right side you increment the number of overlapping pairs. If not, you go to the next value on the left side and try again.

(b)

You can use the interval overlap algorithm to do element uniqueness by first transforming all the input values into intervals. $\forall x_i \in X, (x_i, x_i)$. This can be done in $\Theta(n)$ time. We can then use the interval overlap algorithm to count the number of overlaps. If all elements are unique, there should be zero overlaps. However, if there are any duplicates there will be more.

Since element uniqueness using a linear decision tree model is $\Omega(n \log n)$, this means that any algorithm solving interval overlap in the linear decision tree model must take $\Omega(n \log n)$, thus showing this implementation is optimal.