Due: April 11th (Thursday), 6pm, submit to courseworks.

You may use "standard" arguments without a formal proof—either cite from class or add one sentence showing the main idea.

Problem 1: 2D Range-Counting via Alternating Trees

In class we saw that range trees solve the 2D orthogonal range problem with $s = O(n \lg n)$ words of space and query time $t = O(\lg^2 n)$ (where the query time can be further reduced to $O(\lg n)$ via Fractional Cascading). In this exercise, we shall explore a different data structure, which has (significantly) higher query time, but has cheaper space.

An *Alternating tree* is a binary tree which stores points on the plane (though it can be extended to higher dimensions). Each node v of T stores one of the points $v.p$ and an index $v.i \in \{1, 2\}$, so that $l_{v.i} \leq v.p_{v.i} < r_{v.i}$, for any points l and r from the left and right subtrees of v respectively. The coordinate indices alternate on every path: The root node stores index 1, its children store index 2, each of its grandchildren stores index 1, and so on. Observe that this corresponds to cutting the plane by lines parallel to axes, alternating the choice of the axis, so each node corresponds to a (possibly unbounded) rectangle, and these rectangles cover the plane without intersections. You may assume that all of the points have distinct x and y coordinates.

1. Show how to construct a balanced Alternating tree (i.e. every two subtrees of a node have almost equal sizes) in time $O(n \log n)$. How much space does this data structure require? Prove your answer.
2. Show how to extend alternating trees to answer orthogonal range counting queries (i.e. count number of points in an axis-parallel rectangle R) in time $O(\sqrt{n})$.

   (Hint: for any vertical line, bound the number of rectangles in the tree that intersect it. Then bound the number of rectangles in the tree that intersect the query rectangle R but are not fully covered by it, and connect this number to the complexity of your proposed algorithm.)

Problem 2: Alternative Proof of Fractional Cascading

(Courtesy of Victor Lecomte) In this exercise we will see an alternative Fractional Cascading data structure for searching $Pred(x)$ in a collection of k arbitrary arrays $A_1, \ldots, A_k$ of size N each, in total time $O(k + \lg N)$ and linear $s = O(kN)$ space, that avoids actually cascading/sampling elements as we saw in class.

Consider the following data structure: In the preprocessing phase, merge all arrays in sorted order to form the merged array A^*. For each element a_i in the merged array, store k pointers p_{ij} corresponding to the predecessor of a_i in A_j. Note that this requires space $O(k^2 N)$. To reduce the space, we shall store these pointers only for *every kth* element in A^*. All the rest of the elements in A^* can be removed. Call the resulting array $\tilde{A}$. (What is the overall space now?)

Given the predecessor of a search key x in $\tilde{A}$, how can we find all predecessors $Pred_{A_j}(x)$ for all $j \in [k]$?

Prove that the the *total* search time of this data structure for outputting all k predecessors still takes $O(k + \lg N)$ time (i.e., the *amortized* search times for $Pred_{A_j}(x)$ is $O(1)$, even though some searches may take longer).

Problem 3: Speeding up updates for dynamic 2D Range Counting (ORC)

In class we saw how Buffer Trees can be used to speed up updates in range trees for the 1D dynamic range counting problem (Partial Sums). In the same spirit, show how this technique can be used to speed up the 2D case:

1. Design a dynamic 2D ORC data structure with $\hat{t}_u = O(\lg^{3/2} n)$ (amortized) update time, and $t_q = O(\lg^2 n)$ query time. (Recall that standard 2D range trees have update time $t_u = O(\lg^2 n)$).
2. Recall that layered range trees (fractional cascading) can be dynamized to achieve $t_u = O(\lg^2 n)$ and query time $t_q = O(\lg n)$ for dynamic 2D ORC. Is it possible to speed up the update time of *layered* range trees as in Part (a), which would result in a $t_u = O(\lg^{3/2} n)$ and $t_q = O(\lg n)$ dynamic 2D ORC data structure?