1. Suppose we are given an *undirected, unrooted* tree T with n vertices, represented using an adjacency list data structure. The tree T necessarily has $n-1$ edges.

   For any two vertices u and v of T, let $path_T(u,v)$ denote the unique path from u to v in T. For any three vertices u,v,w of T, let $meet_T(u,v,w)$ denote the unique vertex of T that lies on all three paths $path_T(u,v)$ and $path_T(u,w)$ and $path_T(v,w)$.

   Describe and analyze a data structure that supports the following query:

   - MEET(u,v,w): return the vertex $meet_T(u,v,w)$.

   For full credit, your solution should have the following components:

   - A description of your actual data structure
   - An analysis of the space used by your data structure
   - A preprocessing algorithm that builds your data structure from an adjacency list for T
   - An analysis of the running time of your preprocessing algorithm.
   - A query algorithm that implements MEET.
   - A brief argument that your query algorithm is correct.
   - An analysis of the running time of MEET.

   For full credit, your algorithm should use $O(n)$ space, your preprocessing algorithm should run in $O(n)$ time, and your query algorithm should run in $O(1)$ time; however, larger and/or slower data structures are worth significant partial credit *if they are correct*.

   [Hint: Make liberal use of results from both the regular lecture and the honors lecture. ***Do not reinvent the wheel.*** *If you use a result from class, just explain how to use it; don't regurgitate the details. Most of the components of your solution should be very short, because we've seen the details elsewhere. In particular, give the tree T a root and use the LCA data structure described in the honors lecture on Monday. What can you say about the tree lowest common ancestors* $\mathrm{lca}_T(u,v)$ *and* $\mathrm{lca}_T(u,w)$ *and* $\mathrm{lca}_T(v,w)$*?]*

The remaining problems are for you play with on your own.
Discussion in office hours or on Discord is welcome, but don't submit solutions!

---

2. Let $A[1..n]$ be an array of numbers. Recall that a *Cartesian tree* for A is a rooted binary tree with n vertices, each with a numerical value satisfying two properties:

   - Listing the vertex values according to an inorder traversal of T yields the original array A.
   - Values in T satisfy the *min-heap property*: If v is a child of p, then $v.value > p.value$.

   (a) Prove that if the numbers in A are distinct, then A has a *unique* Cartesian tree.

   (b) Describe an algorithm that constructs the Cartesian tree of A, and show that it runs in $O(n)$ time.

3. Suppose we are given a set P of n points in the plane, each specified by an x-coordinate and a y-coordinate. A *Cartesian tree* for P is a rooted binary tree T with n vertices, each associated with a unique point of P, satisfying two properties:

   - An inorder traversal of T lists the points in P in sorted order by increasing x-coordinate.
   - The y-coordinates in T satisfy the *min-heap property*: If v is a child of p, then $v.y > p.y$.

   Equivalently, after sorting P by increasing x-coordinate, the Cartesian tree for P is precisely the Cartesian tree of the y-coordinates of P.

   The reduction from RMQ to LCA implies that we can answer the following query in $O(\log n)$ time:

   - LowestBetween(l, r): Return the lowest point $p \in P$ (if any) such that $l < p.x < r$.

   Specifically, let p_l be the leftmost point in P such that $p_l.x > l$, and let p_r be the rightmost point in P such that $p_r.x < r$. To answer LowestBetween(l, r), we find p_l and p_r in $O(\log n)$ time using a binary search over the x-coordinates of P, after which we find and return $lca_T(p_l, r_l)$ in $O(1)$ time.

   Describe how to answer the following *3-sided range query* in $O(\log n + k)$ time, where k is the size of the output:

   - ListAllBelow(l, r, t): Return a list of all points $p \in P$ such that $l < p.x < r$ and $p.y < t$.

   [Hint: Finding p_l and p_r still takes $O(\log n)$ time. The rest of the query algorithm takes $O(1)$ time, plus $O(1)$ time per output point.]