Rules: Work in groups of 3. Submit one set of solutions per group, and remember to put all your names on it. We'd prefer if you typed your answers, but if your handwriting is neat and legible, written work is fine. You may refer to whatever sources you want, but *CITE* your references! If you get a great idea from somebody, give appropriate credit: it won't lower your grade, and it's the right thing to do. If you have any questions or errors to report, email `ryanw@cs.cmu.edu`.

1 Fun with Binomial Heaps.

1. Show how to implement `delete(`h`,`i`)` in a binomial heap, in $O(\log n)$ amortized time. That is, give an algorithm, prove its correctness, and prove its runtime. The `delete` should not affect the amortized cost of the other operations; i.e. the credit invariants should be preserved.

2. Can this be improved to *worst case* $O(\log n)$? If so, show how. If not, give an intuitive sketch as to why this is true.

2 Tree Genealogy.

Given a tree T with root r, and two nodes u and v of T, the *least common ancestor* of u and v is the node w with maximum depth in T that is an ancestor of both u and v. (Note in a tree, this is unique.)

The problem we wish to solve is finding *multiple least common ancestors*: given a list S of p pairs of nodes in T (i.e. S is $\{u_1, v_2\}, \ldots, \{u_p, v_p\}$), return a list of p nodes $w_1, \ldots, w_p$ where w_i is the least common ancestor of $\{u_i, v_i\}$ in T, for all $i = 1, \ldots, p$.

Describe an algorithm for *multiple least common ancestors* that runs in $O((n + p)\alpha(n))$ time. (It could have better runtime than this bound, but your bound should be no more than this. Thus the big-O.) Outline a proof of correctness.

Hint: One possible algorithm takes two stages: a preprocessing stage, and a stage where you compute the least common ancestor for each pair.

3 Conjectures about Fibonacci.

This problem is just to settle a conjecture in class. Let x be a node in a Fibonacci heap, and let y_0, …, y_i be the children of x, in increasing order of rank.

Prove or disprove: for all $j = 1, \ldots, i$, is it always the case that the rank of y_j is at most j? In other words, is it possible to create a scenario (through various operations on a Fibonacci heap) where the jth child of some node x has rank more than j?

4 Open problem of the week.

No open problem this week... we ran out (?).