CS 535 Design and Analysis of Algorithms **Spring Semester, 2019**

Homework 3 Version 1.11

Assigned: Feb. 21 *Due: March 7*

The late penalty is changed for this homework. In order to allow partial solutions to be posted before midterm, the latest time to submit (with penalty) is March 12.

Problem 1 Prove that a Fibonacci heap with n elements can have a tree of height $\Omega(n)$. That is, find a positive constant c such that, for any $n > 1$, there is a sequence of operations (INSERT, EXTRACT-MIN, DECREASE-KEY, etc) such that, after these operations, the Fibonacci heap has n elements and a tree of height at least $c \cdot n$.

Problem 2 Consider the Union-Find data structure presented in class. Suppose that we wish to add the operation PRINT-SET(x), which is given a node x and prints all the members of x's set, in any order. Show how we can add just a single attribute to each node in a disjoint-set forest so that PRINT-SET(x) takes time linear in the number of members of x's set and the asymptotic running times of the other operations are unchanged. Assume that we can print each member of the set in $O(1)$ time.

Give pseudocode as in the textbook for MAKE-SET, UNION, FIND, PRINT-SET (if exactly the same, just say so). You do not have to prove correctness for this problem (but the pseudocode must be correct). Do argue the bounds on the running time.

Problem 3 Consider the Union-Find data structure presented in class. Assume there are in total m operations: Make-Set(i), Union(S_i, S_j), and Find(i), and all the Union operations preceed all the the Find operations. Each element i appears in at most one Make-Set(i) operations; therefore the sets are disjoint.

Assume both union-by-rank and path compression are used. Prove that the total running time of the data structure is $O(m)$.

Let us rephrase the problem: Suppose we implement the data structure for Disjoint Sets using trees as discussed in class, but WITHOUT path compression. The time spent so far was longer this way, but we do not care about it in this problems. From now on, we implement Find(i) operations WITH path compression. Prove that q such Find() operations take time $O(q + n)$, where n is the total number of elements.

Hint: before the Find() operations start, give each element 1 credit. Then get a $O(1)$ amortized time for each Find() operation.

Problem 4 Prove that, if a node in a binary-search-tree has two children, its succesor has no left child and its predecessor no right-child.