Advanced Topics in Algorithms
Computer Science 181b
Spring 2012
Homework 4 "Lite", Due Wednesday, February 15

1. **[20 Points] Union-Find with Partial Path Compression.** In class we examined the union-find data structure using union-by-size and path compression. One disadvantage of path compression is that it is a two-phase process: First we must "climb" the path to find the root. Then, we climb the path a second time and set the parent of each node on that path to be the root.

   The famous Shmorbodian theoretical computer scientist, Professor Y. Woerksohaard, proposes the following simpler approach called *partial path compression*: As we "climb" a path from a node to the root of its tree, each node on the path has its parent pointer changed to point to its grandparent. This process can be done in just one pass as we climb up the path. Professor Woerksohaard claims that this approach still allows us to perform any sequence of n MAKESET, UNION, and FIND operations in time $O(n \log^* n)$ time. Is this true or false? If true, give a short but precise explanation (you may cite any part of the proof that we did in class without replicating it). If false, give a short but precise counter-example.

2. **[15 Points] Union-Find Again.** Consider again the union-find data structure using union-by-size and path compression. Assume that we will perform a sequence of n MAKESET, UNION, and FIND operations where all of the MAKESETs are performed before the UNIONs and all of the UNIONs are done before the FINDs. You should assume that the UNIONs that are performed do not require FINDs; that is, the UNIONs are invoked on the canonical nodes of each set. Show that the total running time is now asymptotically even better than $\Theta(n \log^* n)$.

3. **[30 Points] The Dynamic List Access Problem!** In class we looked at the MTF online algorithm for the List Access Problem. Specifically, we looked at the **Static** List Access Problem where only FINDs were allowed - there were no INSERT and DELETE operations permitted in the request sequence.

   Now, consider the more general **Dynamic** List Access Problem which differs from the static version in three ways:

   (a) Any FIND operation may fail. That is, the item might not be found. In this case, the actual cost is $\ell + 1$ where ℓ was the length of the list at the

time that the FIND was performed.

(b) INSERT operations are permitted in the request sequence. An inserted object is placed at the end of the list. If the length of the list prior to the INSERT was ℓ, then the actual cost of this operation is $\ell + 1$. After inserting the element at cost $\ell+1$, you may move it to any position earlier in the list at no cost (just like in a successful FIND operation).

(c) DELETE operations are permitted in the request sequence. The cost of the DELETE is simply the cost of finding the object. (However, the potential function changes as a consequence of the DELETE, so a DELETE differs from a FIND in this way!)

Describe a modification of the MTF algorithm to handle all three of these operations. Show that when your modified MTF algorithm is applied to the Dynamic List Access Problem, it still maintains a competitive ratio of 2. (That is, it is 2-competitive with respect to an optimal offline algorithm.)