**Social network connectivity.** Given a social network containing *n* members and a log file containing *m* timestamps at which times pairs of members formed friendships, design an algorithm to determine the earliest time at which all members are connected (i.e., every member is a friend of a friend of a friend ... of a friend). Assume that the log file is sorted by timestamp and that friendship is an equivalence relation. The running time of your algorithm should be *m*log*n* or better and use extra space proportional to *n*.

Hint: union−find.

Store the friendship data between friends using a weighted quick find union. This will use extra space proportional to n (for the size array). Iterate through the timestamp log and for every timestamp create a union between the two friends in the timestamp. This union operation is log n due to root finding being log n and so iterating through the log will take m log n. To solve the problem, iterate through the log until any index in the size array is equal to n. This would mean that there is a (single) tree containing all the objects (n friends).

**Union-find with specific canonical element.** Add a method find() to the union-find data type so that find(i) returns the largest element in the connected component containing *i*. The operations, union(), connected(), and find() should all take logarithmic time or better.

For example, if one of the connected components is {1,2,6,9}, then the find()method should return 9 for each of the four elements in the connected components.

**Successor with delete**. Given a set of *N* integers *S*={0,1,...,*N*−1} and a sequence of requests of the following form:

* Remove *x* from *S*
* Find the *successor* of *x*: the smallest *y* in *S* such that *y*≥*x*.

design a data type so that all operations (except construction) should take logarithmic time or better.

Hint: use the modification of the union−find data discussed in the previous question.