Name:

ID:

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Explain-It-Back 4 **Spring 2019, CU-Boulder**

The ecology department is planning a large-scale fish migration study that involves electronically tagging and releasing millions of fish across North America, waiting six months, then trapping the fish and recording the where they found each fish, according to its tracking number. In previous smaller-scale experiments, the field scientists used a hand-held device that had a sensor for reading the electronic sensor and a small onboard hard drive that used a predefined table for storing the tag ID, timestamp, and current GPS. In this table, every possible tag had a preset row which allowed for very fast (constant-time) insertions and lookups. While the team would like to re-use this hardware, they do not think that there is enough hard drive space to account for a table with millions of rows. Help them figure out another solution that provides fast insertions and lookups without requiring large memory allocations. HINT: an individual scientist will only tag a few thousand fish at a time.

Name:

ID:

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Explain-It-Back 4 **Spring 2019, CU-Boulder**

To solve this problem we can use hash tables. A has table is essentially an array that assigns an index to new entries in the table using some sort of hashing algorithm. if the index is known, like the old system you were using, you can easily look up a solution without having to go through every single element from the beginning of the array until you reach the element you are looking for. Let's say the element was at the nth element in the array. Then it would take n operations to find the element. In contrast, if you know the index of the array that contains the information you are looking for then you can simply jump straight to what you are looking for taking only 1 operation to reach the desired element. If we give one hash table to each scientist, and we assume that they only tagging lets say 10,000 fish each max (which is far less than the total amount of fish), then we would only need 10,000 entries to that scientists hash table. Then, once we have the array of 10,000 elements, we can begin assigning indexes to each of the fish caught. These indexes will range from 0 to 10,000. To assign indexes we can take their tag number and then divide that number by the total number of elements in the array, so in this case 10,000, and take the remainder. This will be our "hashing algorithm". So for instance if we assign fairly large tags such as 1000000000 being the first and the reset being greater than that, then we divide that tag number by 10,000 and take the remainder, we will get easily manageable indexes. so the fish with tag 1000000001 then divide this by 10,000 and take the remainder, we will get an index of 1. The tag 1000000002 will giver an index of 2 and so on. As long as the number of fish that one scientist must tag remains under 1 million (which must be true considering we only allocated 10,000 rows for a single scientist) than there will be no overlap in indexes. Now each scientist has a 10,000 element hash table that is very quickly accessible for insertions as well as look ups. Then, to combing all of this data into a single source we can make a hash table where each of the elements contain another hash table. In other words, we can assign each scientist a row in the array. We can give the scientists so that we can assign them indexes similar to the fish. So for example lets say Scientist Joe gets the first index and scientist Sarah has the second and so on. Joe's findings, including his hash table will then be stored at his index and same with Sarah, a hash table within a hash table. Now we have a hash table where you can easily insert more scientists and just as easily lookup scientists to retrieve their data on the fish also stored in a hash table.