University of Waterloo
CS240R Spring 2017
Assignment 2 Post mortem

Written Friday, June 9

Problem 1

A common mistake on this question was to iterate from i=0 to n-1 and only swapping a[i] with a[a[i]] if a[i] != i, but doing nothing more. This will not work on all input, as the element that was swapped into a[i] may not be at the right position still. The correct solution was to either decrement i so that the same i is examined again on the next iteration of the loop, or to use a while loop and only increase i if a[i] == [i] is satisfied.

Problem 2

This question was done well, but there were some recurring errors. Some students were careless in writing their pseudocode and missed edge checks that would crash their algorithm. A few students misinterpreted the structure of the matrix, thinking that the number of 1's in each row increased as you went down the matrix.

Problem 3

The quality of solutions to this question was mixed. Many students got the idea of partitioning the chargers and devices into groups of smaller chargers and smaller devices and groups of larger chargers and larger devices around a pivot device-charger pair. However, some of them did not explain this idea clearly and only stated "partition the devices and chargers similar to Quicksort", which is an insufficient explanation as the solution must be expressed in terms of comparisons between chargers and devices.

The analysis of the algorithm was often done poorly. Many students did not do a complete runtime analysis by deriving a recurrence relation, either

explained in words or as a formula, for the runtime of their algorithm and relating the recurrence back to the recurrence of Quicksort that was discussed in class.

Some students wrote a solution which involved partitioning the devices into groups, then trying out each charger one by one on devices, creating a sequence of pivots with each charger. This strategy still works, as it does the same number of expected comparisons as quicksort. It is, however, more similar to binary search than quicksort, and the runtime analysis should reflect that.

Very few students answered the bonus correctly. Many students simply stated that it was trivial that the lower bound for the number of comparisons was $\Omega(n \log n)$ because it was proven in class. However, this is an insufficient analysis, as they must draw a decision tree to show the relation between this problem and the problem of sorting, describing what the nodes and leaves in the tree means.

Problem 4

Many students got full marks on this question. The most common source of failed tests was a series of tests that inserted random or repeated numbers and then called deleteMax() until the entire heap was empty. Students should check their code thoroughly to make sure that they have implemented bubble down correctly and that the case where the last element is deleted from the heap is handled.

Students generally did well on the efficiency tests, with most students passing the long tests where there were tens of thousands of inserts and the majority passed the mods tests, where the number of changes to the heap was counted after an insert or delete operation.