University of Waterloo
CS240 Spring 2020
Assignment 3 Post Mortem

We normally publish a post-mortem for an assignment after it has been marked and released. Here is a list of common errors provided by the graders for assignment 3. Please look at this feedback, as well as the individual feedback provided for your assignment, to get an idea of where you can improve in the future.

General

If a question asks you to design an algorithm, you should provide an english description of the main idea of the algorithm, pseudocode, a justification of correctness for the algorithm, and an analysis of runtime. Many students forget to include the justification of correctness (re-explaining the steps of the algorithm does not prove that it's correct).

Problem 1

Many students did not convert the given numbers into base n so they could be sorted efficiently by radix sort, or didn't explain how this could be done.

A few students tried to use radix sort with $R = 10$, which will not be fast enough as this could inflate m to $\log_{10} n^{43}$ which will make radix sort run in $\Theta(n \log n)$.

While we didn't deduct marks for it, many students claimed that 42 digits would be needed, but it's only possible to represent up to $n^{42} - 1$ with 42 digits in base n.

Problem 2

Several students misunderstood that $FastMerge$ was the name of the supposed merging algorithm and not the name of the modified $Mergesort$ algorithm you were asked to construct. Students that did this often skimmed over how the merging of their $\sqrt{n}$ sorted subarrays was done.

Solving the recurrence relation was generally done well, though some students were unclear about some of the steps of their derivation.

Some students correctly provided the modified $Mergesort$ algorithm and analyzed its runtime, but didn't finish their proof by concluding that $FastMerge$'s existence results in a contradiction.

Problem 3

This problem was done very well.

Problem 4

Most students got the general approach for the algorithms, but some made small indexing mistakes when updating their i value when recursing (particularly when moving up the tree in $ithSuccessor$).

Problem 5

Since AVL trees are recursively defined, it is generally necessary to use recursive proofs (i.e., induction) when proving properties of them. Many non-inductive proofs were often either imprecise or missing important details.

Some students only showed why the root node of T_h was balanced, but forgot to also show that every other node in T_h is also balanced.

Some students successfully proved that it's possible to delete a leaf in T_h and cause $\lfloor h/2 \rfloor$ rotations, but forgot to justify why the resulting height shrinks to $h - 1$.

Many students only included one base case for their inductive proofs. However, many of the inductive cases for these proofs required referring to the previous two trees, which means a second base case would be needed in order to start recursively building up an inductive result.

Problem 6

For part 6.2, many students were not clear about how to handle if n isn't a perfect cube. The number of keys in each level will need to be rounded in some way and you need to show why this rounding will not affect the worst case running time. We didn't deduct any marks

if these details were omitted, but this would be necessary for a complete proof.

For part 6.2, a few students correctly stated the number of keys necessary on each level, but weren't clear about how these keys should be distributed/selected.