University of Waterloo
CS240 Spring 2020
Assignment 2 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 2. 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

Several questions asked for justifications for why the requested algorithms were correct. Many students did not provide justifications of correctness for their algorithms. Some students' provided justifications only gave another description of the exact steps of the algorithm, which doesn't prove that it's correct.

Problem 1

Many students claimed that each call to fix-up takes $\Theta(\log(n))$ in the worst case. While this is true for an individual insertion, it just means that it takes logarithmic time in the **current** size of the tree, which is changing as items are inserted. This yields a total number of operations of $\sum_{i=1}^{n} \log(i)$, rather than an incorrect, but easier to evaluate, $\sum_{i=1}^{n} \log(n)$.

A couple of students only gave an example for a fixed value of n, rather than giving a general construction method that works for any valid value of n.

Problem 2

Many students correctly swapped the ith key with the last key in the heap, but then only attempted to fix-down the swapped key. Remember that heaps only guarantee relative values between parent and children keys (and by extension, ancestors or descendents of keys). If two keys are in different branches of the heap, they may be vastly larger or smaller than each other. This means that the swapped key could potentially be larger than it's parent and require a fix-up operation.

Some students forgot to account for the case when i=0, and thus won't have a parent to compare values to.

Problem 3

Many students forgot to add new values to T_1 and T_2 when inserting, or remove values from them when deleting.

When deleting the max/min element, many students who chose to rewrite their algorithm from Problem 2, rather than just citing it, performed a fix-down on the swapped element rather than a fixup. Because the root of one heap will always correspond to a leaf in the other heap, the position of the swapped element will always be a leaf, and thus have no children (meaning a fix-down would not change the heap at all).

Some students did not notice that deleting an element from H_1 would require deleting the corresponding element in H_2 (and vice versa).

Problem 4

There were three components to the answer required for Problem 4.3. Some students only analyzed the provided expression, but didn't provide answers for the other two components.

Correctly setting up and solving the expression in P4.4 was difficult. Many students did not initially set up their expression correctly which lead to an incorrect final answer. Many students forgot to divide by $n!$ to take the average over all possible permutations. Some students did not simplify their expression as much as possible (commonly leaving it as $\frac{\binom{n+1}{s+1}}{\binom{n}{s}}$ when it can be reduced to just $\frac{n+1}{s+1}$.

Problem 5

For insertion, there are two general approaches. The first is to insert the new element into the appropriate heap based on its value relative to the median, and then rebalance the heaps' sizes afterward if necessary. Alternatively, you can choose which heap to initially insert into based on their sizes, and then potentially reinsert the root elements into the opposite heaps if they are misordered. Some students only maintained the size requirement, but not the ordering requirement (or vice versa).

Some students forgot to initially insert an element into H_{hi} when there are no elements in the data structure, since there is no current median to compare with.