University of Waterloo
CS240 Spring 2020
Midterm Post Mortem

Here is a list of common errors provided by the graders for Midterm. Please look at this feedback, as well as the individual feedback provided for your midterm, to get an idea of where you can improve in the future.

Problem 1

This problem is generally done well.

Problem 2

2.1 and 2.2 were done well. Some students did not provide justifications for the specified run-time in their solution.

2.3 has some students not dividing the frequency by 7 which results in incorrect expected access cost. Also, some students did not provide the right optimal static order for the dictionary.

For 2.4, several students only proved an upper bound while the question asked for a $\Theta()$ expression. It was also common for students to change the $\log(i)$ in the inner loop into $log(n)$, which makes the outer loop analysis trivial. Other mark deductions are mainly caused by skipping steps in the analysis.

2.5 has some students fail to realise the expression $1 + \frac{1}{2^2} + \frac{1}{4^2} + ... + \frac{1}{(\frac{n}{2})^2}$ is between 1 and 2 and thus is constant time. Some students assumed this is $log(n)$ time, which yields the incorrect $\Theta(n^3 log(n))$ instead of $\Theta(n^3)$.

For 2.6, some students calculated the lower bound for inserting n elements into a BST one by one. The main idea of this problem is that one can do an in-order traversal with BST and thus the run-time of Professor X's algorithm should be related to the lower bound of comparison-based sorting algorithm. Also, some students did not state explicitly what the contradiction is or did not provide enough details.

Problem 3

3.1 has some students attempting the question with limit rules, which violates the requirement of proving the given statement using first principles. Also, some students define c_1 and

c_2 as functions of n, which is incorrect for Θ bounds. Other errors include only proving one side of the inequality, with the big-O side being the most common to do and big-Omega was often forgotten.

For 3.2, most students had the right idea for this question but some did not explicitly define n_0 in terms of c, which is required for small-o proof. Also, many forgot to add the +1 term in n_0 to make the inequality strict. Some students attempt to show Big-O instead of small-o; for small-o proof one need to show for any c > 0, there exist some n_0 that works, not just for one particular value of c.

3.3 was done well.

Problem 4

4.1 had some students drawing the heap instead of writing out the array as asked in part (b) and (c).

For 4.2, some students did not include the root node when calculating the maximum number of e' such that e' > e. Also, some students only stated the answer without providing any justifications.

4.3 had several students designed an algorithm to min-heapify the first $\sqrt{n}$ elements, and then compare each of the remaining elements with the current to decide whether to replace it or not. In the worst case, each of the $n - \sqrt{n}$ elements might have to be inserted, resulting in a runtime of $(n - \sqrt{n})log(\sqrt{n}) \in \Theta(nlogn)$.

Several students tried to use QuickSelect to solve the problem. The worst-case runtime for both variants of QuickSelect shown in lectures is $\Theta(n^2)$. Runtime constraints in algorithm design refer to the worst-case by default. The lectures did mention that QuickSelect can be done in Θn worst-case runtime, but its details were not covered, so using this QuickSelect requires proving that it satisfies the problem constraints (no heaps with greater than $\sqrt{n}$ elements, no non-array/heap structures).

Several students claimed that inserting $\sqrt{n}$ elements into a heap takes $\Theta(\sqrt{n})$ time, but this actually takes $\Theta(\sqrt{n}log(\sqrt{n}) = \Theta(\sqrt{n}logn)$ time. Achieving $\Theta(\sqrt{n})$ time to construct the heap requires using the heapify algorithm, which involves fix-downs instead of inserting elements.

Several students that correctly found the maximum from $\sqrt{n}$ heaps neglected the runtime for deleteMax. Although the deleteMax runtime is dominated by the cost of finding the maximum, this should be included in the runtime analysis.

A few students suggested that, instead of deleting the maximum element of a heap, that the left and right subtrees of the root node should be added as separate heaps in place of the previous heap. This would've been easy if the heaps were implemented as tree structures, but the heaps covered in this course are implemented as arrays, where it is relatively difficult and expensive to isolate subtrees like this.

Problem 5

For P5, many students argued that transforming A to A' will ensure A' has only unique elements, but you still need to argue that the transformation preserves the relative ordering of the elements in A, as otherwise sorting A' doesn't help with sorting A

Some students tried to change the behaviour of the provided MySort function. This was publicly clarified on Piazza to not be allowed. Almost all of these proposed changes made assumptions about how the MySort algorithm would work, which isn't safe to do if you don't know the implementation of MySort.

To undo the transformation on A' after sorting it, many students tried to do 'for i from 1 to n: A'[i] = A[$\frac{(A'[i]-i)}{n}$]'. However, the original index i used to transform an element in A will not necessarily be the same index of A' that the element is in after sorting. This can result in accidentally selecting the wrong element of A in the untransformation.

Problem 6

Some students get $\Theta(m(n+R)) = \Theta(2mn)$ and did not simplify using m=n or drop the constant term. In LSD-radix sort, m represents the number of digits and m=n in this question since the entries of A are in 1, 2, 3, ..., $n^n - 1$, which means each entry has at most n digits in base n.

Problem 7

For 7.1, some students wrote height in their AVL tree instead of balance. Also, some students have the wrong sign for the balance factor, which is defined to be height(right subtree) - height(left subtree).

7.2 had some students deleting more than one node from the AVL tree. We fix the balances by doing rotations in the resulting tree, not deleting unrelated nodes.

Problem 8

For P8, some students missed the $\Theta(1)$ term in the first branch of if statement. Some students assumed 1 for the run-time of "return B"; although it does take constant time, $\Theta(1)$ would be more appropriate here.

Some students wrote 1 - α instead of α for is-correct(B), this is a simple misreading of the question.

Problem 9

The most common mistake is assuming the skiplist is sorted in non-increasing / descending order. The question stated that the priority queue (ADT) is max-oriented, not the skiplist (implementation) itself. We do not allow changing the order of nodes in level 0.

Some students stored the largest node in a variable max and called skipSearch on max to find the second largest node. This is incorrect as skipSearch either gives the node found or return "not found, but would be after p0". Note that the error message does not give you the node, hence you should call getPredecessors instead of skipSearch. Also, note that getPredecessors return a stack of nodes, not just one node. You need to call .top() on the returned stack to obtain the second largest node in the skiplist.