University of Waterloo
CS240R Spring 2017
Midterm Post mortem

Written Friday, June 23

Problem 1

1a) was done well by most students.

The majority of students chose to justify a.1) for part b), but there were recurring errors for this question. Many students just stated definitions of big O and little o and proceeded to say that the definitions alone were enough to show that a.1) was false, which is insufficient. Other students said that the existence of a c in big O means that the "for all c" part of the little o definition is false. This is not true, as all c could still be valid for big O. Finally, there were some students who wrote the definition of big O, then proceeded to say let c be a constant such that $f(n) = c * g(n)$. This is not a valid proof.

When justifying a.2), the majority of the students didn't know what preorder traversal meant, although their counter examples were still correct. Preorder traversal means printing the data of the current node first, then recursing on the left subtree, then recursing on the right subtree. Many students mentioned that for a max heap, a preorder traversal will not yield keys in decreasing order, because the left child is less than the root, but a max heap of 1 element does not satisfy this criteria, and hence would produce a decreasing order of keys in a preorder traversal.

Not many students chose to justify a.3), but of the ones that did, some said that best case comparisons for insertion sort was O(1) comparisons, which is false. The best case comparisons for insert sort is O(n).

a.4), a.6), a.7), and a.8) were all generally well done. For a.5), many students said let $f(n) = n$, and since n is in $O(n^2)$, then that means every single function in $O(n) \in O(n^2)$. This is an invalid proof. A proof by example

only works when you want to prove either the existence of an element or to disprove a "for all" statement.

Problem 2

2a) had mixed performance. Students should make sure that they understand logarithm laws as well as how a logarithm raised to a high power will have a different growth rate than a logarithm which is only raised to a power of 1. For $n(1 + (-1)^n)$, students should be careful about understanding the growth rate of a function that alternates between values of 0 and $2n$. Because it is always equal to 0 for odd values of n, the function is not actually in any of the given order notations.

For 2b), students did well to separate the polynomial and exponential functions into their own groups. However, in the group of exponential functions, there were some students who did not get the order right, and there were many who thought that $n * 2^n > e^n$, which is untrue since $e^n = (\frac{e}{2})^n * 2^n$ where $(\frac{e}{2})^n$ is an exponential function with base > 1, which grows much faster than n.

Problem 3

Most students did this question well. For part a), some students forgot to refer to the index in the array that the new key k would be located at.

Problem 4

There were many recurring errors in this problem, such as answers that were missing justification, describing running time instead of comparisons and swaps, and students being imprecise with their expression (not using Θ-notation). Quite a few answers also wrote (n - 1)! comparisons for parts a) and b), which is quite far off.

Part (a) was mostly done well.

Most students did not realize that for even iterations in part (b), the pivot is the smallest element in the partition. A lot of answers said n^2 swaps were required without an attempt at justification.

In part c), common errors were: not recognizing that random input is similar to the randomized algorithm or equivalent to the average case, and not recognizing that more than one swap can occur in an iteration.

Part d): If all values are the same, Quicksort does not move the pivot to the middle. Also, each partition of Quicksort only sorts the results into two groups ($\leq$ and $>$ OR $<$ and $\geq$). It therefore does not stop after one iteration. The solution could require 0 swaps if $A[i] < A[0]$ and $A[j] \geq A[0]$ was used in the partition algorithm (and assuming swap(A[i],A[i]) is ignored), but this is not what the lecture notes described. However, students still received full marks if they stated their assumptions about how values equal to the pivot were treated.

e) was in general very well done. Common errors were giving the wrong value for the median and failing to explain why the value 9 was chosen.

Problem 5

The most common mistake involved students believing that Move to Middle meant that the element was moved to the middle of the array instead of being moved to a position halfway between its current index and the front of the array. The question explicitly states that an element found at index i is moved to index $\lfloor \frac{i}{2} \rfloor$.

Many answers also failed to write the correct number of comparisons done for the last search, perhaps because students failed to take into account the comparison that determines that the key was found or that they wrote the number of comparisons that would be done on the array when searching for 5 after the last search(5) instead of on the last search.

Problem 6

Quite a few of the students chose the leaf-nodes in the AVL tree as their answer for part a), but leaf nodes cannot be invalid in an AVL tree because their balance is always zero.

Parts b) and c) were mostly done well. Most of the wrong answers appeared to be because the student did not understand how to do all four types of rotations. Also, there were some answers in c) where students put perfect trees, which would have a balanced root and the question specifically asks for an unbalanced root.

Problem 7

Due to a high number of answers that lacked justification and only gave the algorithm alone, the number of marks that this question is worth has been reduced. Students who did include a correct justification, however, will still get the number of marks that the original question was worth.

Many algorithms given involved traversing the tree in some way and enforcing at most one rotation per node, which wouldn't work. Several answers involved creating right-paths before turning it into a left-path, which either doesn't produce a left-path or involves too many rotations.

In part b), many students gave trees that wouldn't require $\Omega(n)$ rotations upon deletion or simply drew a fixed number of nodes for their right path, which does not imply $\Omega(n)$ rotations unless a brief justification is given.

Problem 8

Most students did this question well. Many understood that they needed to include an extra variable containing the sum of all the values in the array.

Problem 9

Several answers involved denoting the number of tails as the height instead of heads. They were not penalized if they stated their assumption, but a few answers did not state this assumption. Many answers involved circling the node that is reached from a "below" operation, even though comparisons are only done with "after" nodes. Some answers had the nodes in the insertion order instead of alphabetical order. Many students somehow missed circling the node right after where "M" should have been inserted ("O").

Part c) was difficult, with most students getting zero or one marks. The most common answer, by far, was to let the field store the position of the node (either the position from the front, or from the back). This would take $\Omega(n)$ time to insert/delete since many nodes would need to be updated. A less common answer was to have a field that indicates whether a node is before the $p-th$ element, or whether the node is the $p-th$ element. The analysis would argue that only a logarithmic number of nodes would need to update the field during insert/delete, but it did not seem to consider the implication that each node would need to have n such fields, all of which would need to be updated.