University of Waterloo
CS240 Spring 2020
Assignment 1 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 1. 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.

Problem 1

It's very easy to accidently make claims that only hold true for certain values of n. For example, it's not safe to divide by $n \log \log n$ unless you specify that $n > 2$. You should always keep track of and state what values of n your derivation holds for (this will also help you in choosing your value of n_0 for first principle proofs).

Several student didn't use strict inequalities ($>$ rather than $\geq$) when working with little-o and little-ω proofs.

For 1.4 and 1.5, it was often a common error to forget the "+1" for values of n_0 to ensure a strict inequality.

1.5 had several attempts of substituting values of n in a manner that did not justify the claimed inequality. For example, with some intermediate function $f(n)$, the claim $f(n) \geq n^{1/n}$ often involved substituting $n = n_0$ (or one component of n_0), but this implies $(n_0)^{1/n_0} \geq n^{1/n}$, which is not valid under $n \geq n_0$.

When writing backward-style proofs where you start with some conclusion inequality (such as with first principle proofs), it is important that you only ever progress with if-and-only-if ($\iff$) steps whose implications hold in both directions. After writing a backward-style proof, you should double-check that you can start at a statement that you know to be true and then progress through a series of valid implications to arrive at the final statement that you're trying to prove.

Problem 2

This question was done well, but many students didn't justify how they evaluated their limits. For example, claiming that $\lim_{x\to\infty} \frac{\sqrt(n)}{\log(n)} = \infty$ should be backed up with an application of l'Hopital's rule.

Problem 3

While a well-written analysis in words was accepted, it is typically much cleaner to set up a summation that represents the exact running time of a loop and then solve that summation. This won't always work for complicated loops, but is very effective for for-loops.

For 3.4, it was very common for students to only analyze either the fastest the loop could run, or the slowest. To show a Θ bound here, you should separately derive an upper and lower bound on the runtime of the loop and show that these bounds are asymptotically equivalent.

This was not penalized, but many attempts at 3.4 referred to a "worst-case runtime" or "best-case runtime". The concept of worst-case and best-case does not apply to Problem 3, since there is no notion of input instances. Each code segment has only a single runtime expression as a function of n. What the students were actually referring to were the upper and lower bounds respectively for this single function of n. This misunderstanding may have contributed to the absence of lower bound analyses.

3.4 has several students fail to justify why the upper bound involves dividing by 2 after every 2nd iteration. The key observation that s becomes even after multiplying by 2 was not stated, despite being an essential component of the reasoning presented.

For 3.4, a few students tried a sort of "average" argument along the lines of "half of the integers are even and half are odd, so this averages to dividing s by 2 in each iteration, which gives our runtime of $\Theta(\log n)$". Analyzing an average case doesn't help when proving an overall Θ bound, as we are interested in showing that the fastest and slowest the program can run is asymptotically the same.

3.5 had some students giving a handwavy explanation of how many iterations would be required for the loop to terminate. This frequently resulted in $\log n$ iterations, whereas a precise mathematical derviation would yield the correct $\log \log n$ iterations.

Problem 4

4.2 had many students ignore the n_0 values of their premises. You need to use these values to derive a n_0 value to satisfy the first principles definition that you're trying to prove.

4.2 had a number of students trying to utilize the limit rule, which violates the requirement of proving the statement from the first principles definitions of order notation.

Problem 5

For 5.1, many students answered with "Priority Queue" which is an ADT and not a data structure like was required. ADTs only provide a list of of required operations, but make no guarantees about how fast these operations will perform. Another common answer was "Array" which, without additional properties, would not effectively support the needed operations.

5.3 had a number of basic math errors when calculating $(x + y - 1)$.

Some answers for 5.5 incorrectly evaluated geometric series due to the summation not beginning at 0.

Problem 6

It is important to remember that $f(n) \in \Theta(g(n))$ does not imply anything about $\lim f(n)/g(n)$. It is possible that the limit doesn't exist. If the limit exists, it can be used to derive an order notation relationship, but the opposite direction does not always hold.

6.1 Had many cases of neglecting n_0 constants. Most commonly, the same n_0 was used for both premises, inequality derviations were made without establishing the necessary n_0-relation assumptions, and a final n_0 value was not provided to satisfy the first principles definition.

6.1 also had several cases of using incorrect bounds for the denominator. Increasing the denominator decreases the fraction, so an upper bound on the denominator gives a lower bound on the fraction, and vice versa.

Many students assumed (usually implicitly) that certain constants are positive when they aren't necessarily so. For example, $\log c_1$, $\log c_2$, and $\log(g(n_0))$ are all negative for fractional c_1, c_2, and $g(n_0)$.

6.2 had some students stating that $\log(c * g(n))$ is equal to either $c * \log(g(n))$ or $\log c * \log(g(n))$. The result should be $\log(c * g(n)) = \log c + log(g(n))$.

Some general advice (that would have helped for 6.2): If the premise is symmetric, such as a Θ bound, then you should expect that the proof can be symmetric as well. After all, if $f(n) \in \Theta(g(n))$, this implies $g(n) \in \Theta(f(n))$. The techniques used for a lower bound proof should also be applicable for an upper bound proof and vice versa (the details may vary of course). Many approaches here tried to prove the lower bound in a manner that clearly could not have been applied to the upper bound, which should have hinted that something was wrong.