Jeffrey Wan/ Module 1/ Spring 2020/ Data Structures
Answers to prompt questions for this module are:

__**What is the design value in writing an ADT?**__
- ADTs support abstraction by hiding the underlying implementation details and providing a well-defined set of operations for using the ADT. It allows the user of the ADT to interact and use the API (the defined uses) of the ADT without having to worry about the implementation of the ADT. Programmers can focus on behavior of the ADT too which allows for more efficient programming of the ADT.

__**When assessing complexity, what do we measure (or, what do we not measure)?**__
Algorithm efficiency is typically measured by the algorithm's computational complexity. Computational complexity is the amount of resources used by the algorithm. The most common resources considered are the runtime (elementary operations made by the cpu) and memory usage (fixed-size memory units needed to run the algo). So when assessing complexity, we measure the amount of resources the algo uses. We do not actually measure time an algo takes for assessing complexity.

__**Does an upper bound for a function apply in all cases? Explain.**__
. These lower and upper bounds provide a general picture of the runtime, while using simpler functions than the exact runtime. The upper bound does not apply in all cases. The upper bound is an approximation that is most accurate for very large N (the input size). But closer to the origin, the upper bound may actually direct us to pick the the worse performing algorithm since close to the origin, the actual algorithm performance may differ from the performance of the algorithm when N is large. For different input sizes, the upper bound may actually be better performing than the actual algo itself.

It depends on what applies means lol. In small input sizes, teh upper bound may actually be lower than the actual algo itself. The upper bound only says that it performs worse than the algo after a certain number N. So the upper bound isn't an upper bound for all N values where N is the input size.

__**Why might it be important to have different measure of complexity (e.g. upper bounds, lower bounds, …)?**__

– It is useful to know what the worse case scenario might be because we do not have control over the input data size. It gives a sense of the bounds of time that an algorithm might take to run.

__**Under what conditions might you not be concerned about upper and lower bounds?**__

If I knew I'd be working with a small input size, I wouldn't be concerned about upper and lower bounds. In fact, the labels and estimates might suggest using an algo that was worse performing at a small N since the estimates are accurate only at very large N.

I would be concerned about the bounds I was say concerned about cost and if above a certain T(N) threshold, I'd run out of money.

__**The lecture notes state, “Sometimes the ‘worst’ algorithm is the best choice.” What does this mean, and how might it apply in specific situation?**__

At a small N, the worse estimate might actually be better than the estimate that has the lower T(N) over large N. At a small N, the actual algorithm might actually perform much worse than its long-run performance and so its estimate might suggest that an algorithm is actually better over an N when it actually is the worse of the two choices.

__**How important is an understanding of space complexity in an era of cheap memory?**__

Not that important? Memory is cheap? I'm not sure if it s important. Time complexity seems more important of the two.