CSE143 Lecture Questions for Friday, 5/14/21

Question	Answer
I love the fun anagram demonstration at the start! :-) Haha yeah it is quite nifty.	I presented this assignment at a conference in a panel called "nifty assignments." I think you can tell something is nifty when people really want to run the program themselves to type in their own name or other items to turn into anagrams.
How would we change this program to work with 3D chess? I'm guessing it would be much more complicated. I see; does backtracking work only with Qubic just because it has much fewer possibilities than 3d chess would have? I see so backtracking is not necessarily efficient, but it is easier than other search approaches, so it is a good option for problems that have "few" options? Ok, thank you.	The basic backtracking idea would work with chess where you consider all possible moves and just explore where it leads, but chess would lead to too many options to explore in a meaningful way. I've given an assignment in the past where I have a program find a guaranteed win in a game called Qubic which is 3-dimensional tic-tac-toe on a 4x4x4 board. Yes, Qubic is "small" enough that simple backtracking works. Problems like chess require more efficient search approaches. Yes, exactly.
So this 8 queens problem is different from exhaustive search because, as soon as one pathway doesn't work, you go backwards to a different choice? Does that sound right? Ok, and by contrast, in exhaustive search, you go through every single possibility right? Ok I think I get it, thanks.	Yes, the big difference is the recognition of dead ends that you don't explore further. Yes, in exhaustive search you'd explore everything.
Is recursion zen used because it is good practice/easier to read OR because it is necessary for the code to work? Ok, thank you. Ohh is it because it avoids running into the remove method? Because if you don't do explore, the next line of code to run is remove. What is the other way to make it work? Have the if statement after the explore method? Ah ok.	We apply those ideas to simplify the code. You can generally get it to work either way, but it's helpful to be able to make things simpler. That also tends to avoid bugs because the code is easier to verify for correctness (fewer cases). You can make it work either way. I'm not going to figure out how to write bad code. :-)

I'm afraid to know, but what's the big O notation for a program like AnagramSolver? It seems that there's a lot of lines of code running. Yeah, it feels super long and expensive	Theoretically it's n^m where n is the size of the dictionary and m is the max number of words allowed (potentially VERY expensive).
If our letter inventory code is sufficiently functional, do we need to worry about making corrections to it based on the feedback we got for it? Or do we just need to focus on the task we're given for homework 6? Ohh ok, I misunderstood. I understand now, thank you.	I provide a compiled version of LetterInventory that you should be using. It's guaranteed to behave properly.