

# ECS 32B: Exam #2 Details

Instructor: Aaron Kaloti

Summer Session #1 2020

## Contents

<b>1 Changelog</b>	<b>1</b>
<b>2 Exam Format</b>	<b>1</b>
2.1 Regarding Specific Exam Problems . . . . .	2
<b>3 List of Topics</b>	<b>2</b>
3.1 Slide Deck #1: Computational Complexity . . . . .	2
3.2 Slide Deck #2: Python Concepts . . . . .	2
3.3 Slide Deck #3: Searching . . . . .	2
3.4 Slide Deck #4: Linear Data Structures . . . . .	3
3.5 Slide Deck #5: Recursion . . . . .	3
3.6 Slide Deck #6: Trees - Intro . . . . .	3
3.7 Slide Deck #7: Trees - Advanced . . . . .	3

## 1 Changelog

You should always refer to the latest version of the syllabus.

- v.1: Initial version.

## 2 Exam Format

This exam will be a timed Gradescope quiz that will be proctored over Zoom. While in the Zoom meeting, you are required to have your face visible through the webcam at all times. **If you fail to meet such requirements, e.g. you don't join the Zoom meeting or you leave your webcam off even after I try to tell you to turn it on**, then I may have to give you a zero on the exam. The purpose of proctoring through this Zoom meeting is to ensure that it is truly you who is taking the exam and that you are not communicating with any other student as you take the exam. The Zoom meeting will be open at 9:55 AM. The exam (i.e. the Gradescope quiz) should become available around 10:00 AM. Once you start it, you will have 70 minutes to finish. (Thus, if you start at 10:02 AM instead of 10:00 AM, you will still have 70 minutes.) If you start the exam after 10:10 AM, then I expect some sort of explanation over email; if you do not provide one ASAP (as in, around the time you start the exam), then I may have to reject your exam submission.

Some of the questions on the exam may ask you to write code and encourage that you use some Python development environment of your choosing (e.g. Python IDLE, Mu editor, PyCharm) as you do so. On such questions, I will permit you to submit a Python file for that question instead of pasting your code into the answer box. The reason for this is that sometimes, students have trouble pasting code into the answer box on Gradescope quizzes, perhaps due to a bug on Gradescope's end. In case you wish to test the functionalities of the Gradescope quiz, I have created a sample Gradescope quiz called "Exam File Upload Test" that will always be open.

All audio will be muted during the meeting. This means that you can use headphones or ear buds and listen to music, if you want. I am allowing this because I know that for some students, they may not have access to a quiet environment or control over how loud the others they live with are. Please do not use any of the disruptive features such as the "Raise Hand" feature during the Zoom meeting; if you need my attention for whatever reason, use email, as I will constantly check my email during the exam.

---

\*This content is protected and may not be shared, uploaded, or distributed.

The exam is open note, open keyboard (since you need to type your answers somehow), etc. **You are allowed to view the lecture slides on Canvas; I consider that as being part of your notes. You are allowed to view notes that you have on a tablet/iPad as well.** I might also recommend that you have scratch paper ready, just in case. **You are not allowed to search for answers online. You are not allowed to ask each other for help. Cheating on an exam will be caught and punished, almost certainly with an F in the course.** You are allowed – and for some problems, *advised* – to use an editor such as the Mu editor or Python IDLE, so do make sure that you have an editor that you are comfortable with. You are *not* allowed to consult me or the TAs for hints on the correct answers, but you *are* allowed to ask *me* for clarifications on the questions, over email.

If you finish the exam early, leave the Zoom meeting; no need to let me know. Zoom generates a log of all of those who enter and exit the meeting.

If you want, you can set a virtual background on Zoom.

If you are having connection issues due to being in the Zoom meeting, please shoot me an email immediately (yes, during the exam).

If you are a student who has accommodations that you have informed me about, then I will contact you soon about how the exam might be handled differently in your case. Please email me if I do not contact you soon.

## 2.1 Regarding Specific Exam Problems

The exam will have one coding question at the beginning, similar to the two that were at the beginning of the previous exam. The rest of the questions will be conceptual, although there is a small chance that a question could ask you to provide a few lines of code to accomplish a certain goal.

The coding question at the beginning could involve topics such as conditional statements, loops (including `break` and `continue`), strings, lists, dictionaries, and tuples. It could also involve sets. It will *not* involve file I/O, user-defined classes, references, or unit testing.

I will try to avoid asking you to type math in any of the questions. If I end up having to, however, then there will not be a strict expectation as to how you format certain mathematical operations (e.g. you could denote exponentiation with either `**` or `^`), so long as your answer is unambiguous. For denoting big-Theta or big-Omega, you can use `Theta(...)` or `Omega(...)`. There will be a reminder about all of this on the exam, if it is relevant.

## 3 List of Topics

Below are a list of topics that you should make sure that you understand before taking exam #2. Not all of these topics will appear on the exam.

### 3.1 Slide Deck #1: Computational Complexity

- Big- $O$ . Formal definition. Proving that a big- $O$  holds. Big- $O$  is an upper-bound.
- You will not be asked to create a cost function.
- Definitions of big- $\Omega$  and big- $\Theta$ . You will not be asked anything about proving big- $\Omega$  or big- $\Theta$ .
- Be able to identify the worst-case time complexity (with each of big- $O$ , big- $\Omega$ , and big- $\Theta$ ) of a given segment of code or function.
- Common categories of functions: constant, logarithmic, polynomial, exponential, factorial. You will not be expected to tell if a given segment of code runs in exponential or factorial time. However, you may be expected to tell if a mathematical function is big- $O$  (or big- $\Omega$  or big- $\Theta$ ) of an exponential or factorial function (without having to prove it).
- Space complexity. You will only ever be asked about auxiliary space.
- Best-case time/space complexity and average-case time/space complexity will not be tested.
- You do not need to understand the different notations for saying that a given function  $T(n)$  is  $O(f(n))$ .
- You will not be tested on the symmetric or transpose symmetric properties of big- $O$  and the like.

### 3.2 Slide Deck #2: Python Concepts

Nothing from this slide deck will be tested.

### 3.3 Slide Deck #3: Searching

- Linear search.
- Binary search. How the algorithm works. Its worst-case time/space complexity.

### 3.4 Slide Deck #4: Linear Data Structures

- Stacks. How they work, implementation, etc. Understand the examples.
- Understand infix/prefix/postfix notations.
- Queue. How they work, implementation, etc.
- Deque. How they work, implementation, etc.
- Linked lists. Using a linked list to implement an unordered list and an ordered list.
- You do not need to understand the distinction between an ADT and how it could be implemented.
- You don't need to understand why appending to a Python list takes linear time in the worst case.
- Amortized analysis will *not* be on the exam.

### 3.5 Slide Deck #5: Recursion

- Understand how recursion works and how to use it.
- Understand the activation stack, i.e. how variables local to each specific recursive call are maintained. (You don't need to understand slide #12, the operating systems concepts.) Recursive implementations of linear search and binary search.

### 3.6 Slide Deck #6: Trees - Intro

- Terminology.
- Binary tree. Understand the two implementations that I lectured about.
- Binary tree traversals (all three of them).
- Binary search trees. Understand how find, insert, and delete work. Binary tree vs. binary search tree.
- Why the worst-case time complexity of find, insert, and delete are linear for a binary search tree.

### 3.7 Slide Deck #7: Trees - Advanced

- Computing the height and balance factor of a node.

