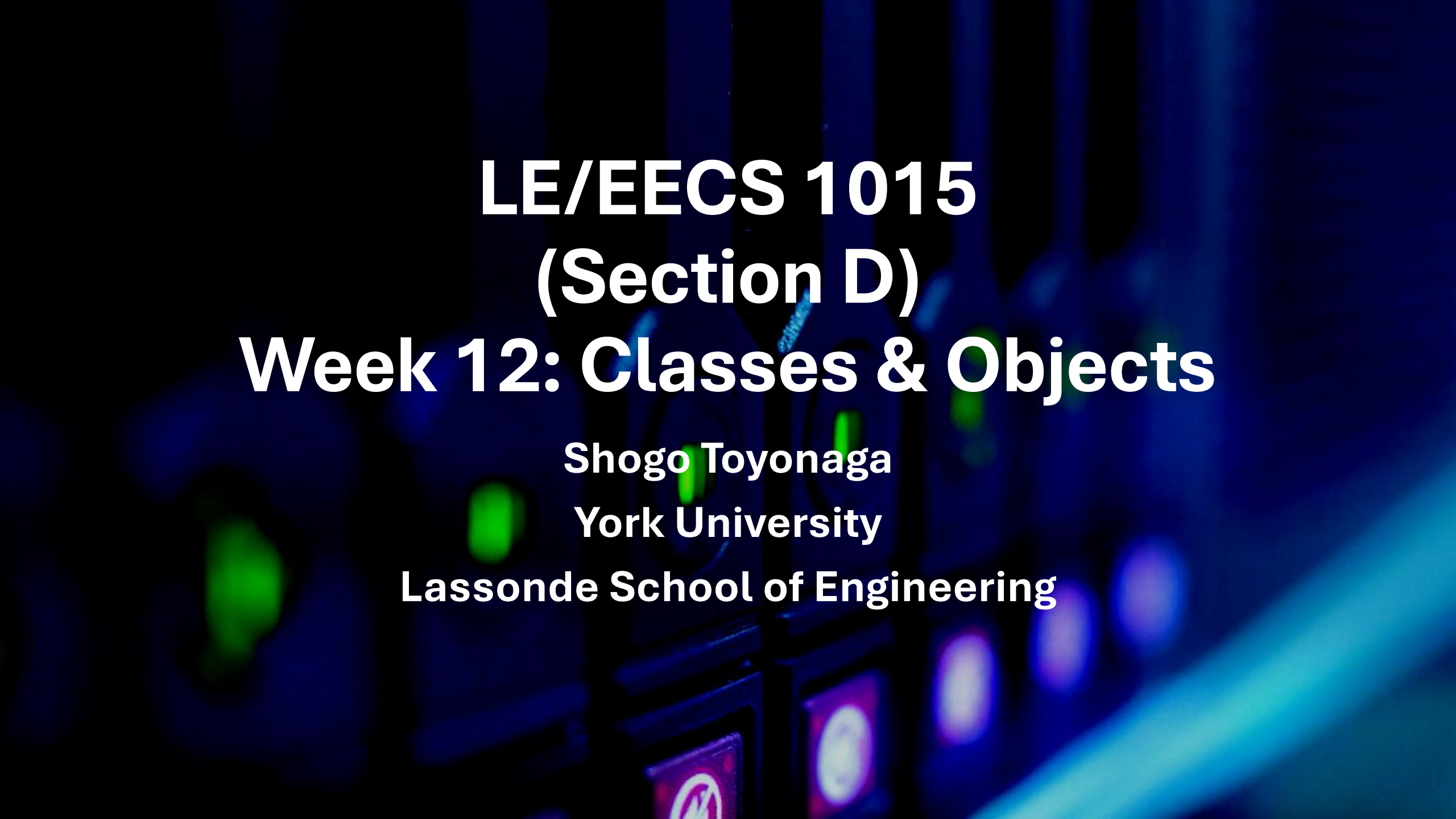


LE/EECS 1015 (Section D) Week 12: Classes & Objects

Shogo Toyonaga

York University

Lassonde School of Engineering

This Week...

1. An Introduction to Classes & Objects
   - Instance Variables
   - Function Parameters
   - Collections
   - “self” / “this”
2. Some Forewords on Time & Space Complexity
   - Data Structures
   - Algorithms (Sorting)

Goals of Lab 10

1. Writing and debugging scripts which define classes and objects.
2. Learning how to apply object-oriented programming principles to real-world problems.

Lab 10 – What You Do....

Task	Points
Follow the Steps (Rectangle Class)	50
Debugging (Inventory Management System)	30

Lab 10 – Useful Resources

- Python Object Oriented Programming in 10 minutes 🐍 (BroCode)
- Python read a file 🔍 (Bro Code)
- Write files using Python! ✍️ (Bro Code)

Optional (But Highly Encouraged)

- Python inheritance 👨‍👨‍👦 (BroCode)
- Learn Python polymorphism in 8 minutes! 🎭 (BroCode)
- Learn Python generators in 8 minutes! 🚰 (BroCode)
- Learn Python generator expressions in 9 minutes! ♻️ (BroCode)
- Big-Oh, Big-Omega, Big-Theta Lecture Slides
- Time Complexity Proofs (Some Examples)

Classes & Objects

- **Classes** allow us to create our own data type with custom *attributes* and *methods* (“actions / behaviours”).
  - Attributes are created and accessed using **dot notation** and can represent any kind of data type

    (e.g., we’ve seen int, float, list, set, dict, and tuple so far….)
  - The first parameter for methods in a class should always be **self** **(unless it is a static method)** by convention. It shares a similar role to, “this” in Java. *It is used to access instance-specific attributes or behaviours of an object.*
- **Objects** represent the *instantiation* or *realization* of your class. As such, each object has its own memory address!

Classes & Objects

- We use Classes & Objects because they allow us to….
    1. Modularize code such that each block focuses on a small set of priorities. (This makes debugging easier!)
    2. Allows you to re-use code through more advanced techniques which you will learn in LE/EECS 2030 such as…
        a) Inheritance
        b) Polymorphism

Classes & Objects

- You can implement certain methods with special names in your classes which allow your objects to interact appropriately with basic Python syntax (e.g., operators).
- Examples
  1. __str__(self, /) will return str(self)
  2. __add__(self, value, /) will return self + value
  3. __lt__(self, value, /) will return self < value
  4. __invert__(self, /) will return ~self
- __init__(self, *args, **kwargs) will initialize the attributes of a newly defined object. It serves as the class's constructor.

Putting It All Together

1. Declare a Class (Naming Convention Uses CamelCase)
2. Define the Constructor (__init__(....))
   - Define private attributes using the underscore convention.
3. Define the Methods (Naming Convention Uses snake_case)
   - Getter: Returns the value of an objects attribute.
   - Setter: Binds a new value to the objects' attribute.
   - Helper: Assists in implementing new behaviour of the class.

File I (Input) / O (Output)

- Python has built-in functions $(such\ as\ open())$ which allows us to create, read, or write data to/from memory!
  - **Relative Path:** Contingent based on where you are $(ls \vee dir)$.
  - **Absolute Path:** Hard-coded for your specific memory management hierarchy (pwd).
- Once you are done working with a file, you should close it as soon as possible to (1) reduce memory overhead and (2) avoid the risk of a race condition.

Some Notes on Open()

- The mode parameter can be set to create, read, or write to/from a file based on the following table:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

Demo Use-Case of Open()

```python
def stream_lyrics(file_name: str):
    with open(file_name, mode = 'r') as file:
        for line in file:
            for word in line.split():
                yield word + " "
                time.sleep(0.02)
            yield '\n'
```

✓ 0.0s Python

```python
for word in stream_lyrics('lyrics.txt'):
    print(word, end = '', flush=True)
```

Python

Computational Complexity

- Time
  - How, ‘fast’ is your program?
  - You must consider the pros and cons of using certain data structures and/or algorithms in developing the optimal program.
- Space Complexity
  - How much memory does a program need?
  - How much auxiliary space is required?
  - How deep is the call stack? (e.g. ,Recursion)

Time Complexity

- Time complexity involves the analysis of time required to solve a problem given the size of an input.
- We are interested in assessing how much longer it takes to solve a problem as the input grows given an algorithm.
- We are mostly interested in the worst-case scenario when using Big-O notation, however, you can also assess the average-case and best-case.

Big-Oh Notation (Upper Bound)

- Big-O notation is used to estimate the number of operations that an algorithm makes as the input grows.

- "Let f and g be functions from the set of integers or the set of real numbers. We say that $f(x)$ is $O(g(x))$ if $\exists C, k > 0$ such that $|f(x)| \leq C \times |g(x)|$ whenever $x \geq k$."

- This basically says that $if\ f(x) is\ O\big(g(x)\big)$, then $f(x)$ grows slower than some fixed $c \times g(x)$ as x grows within a bound.

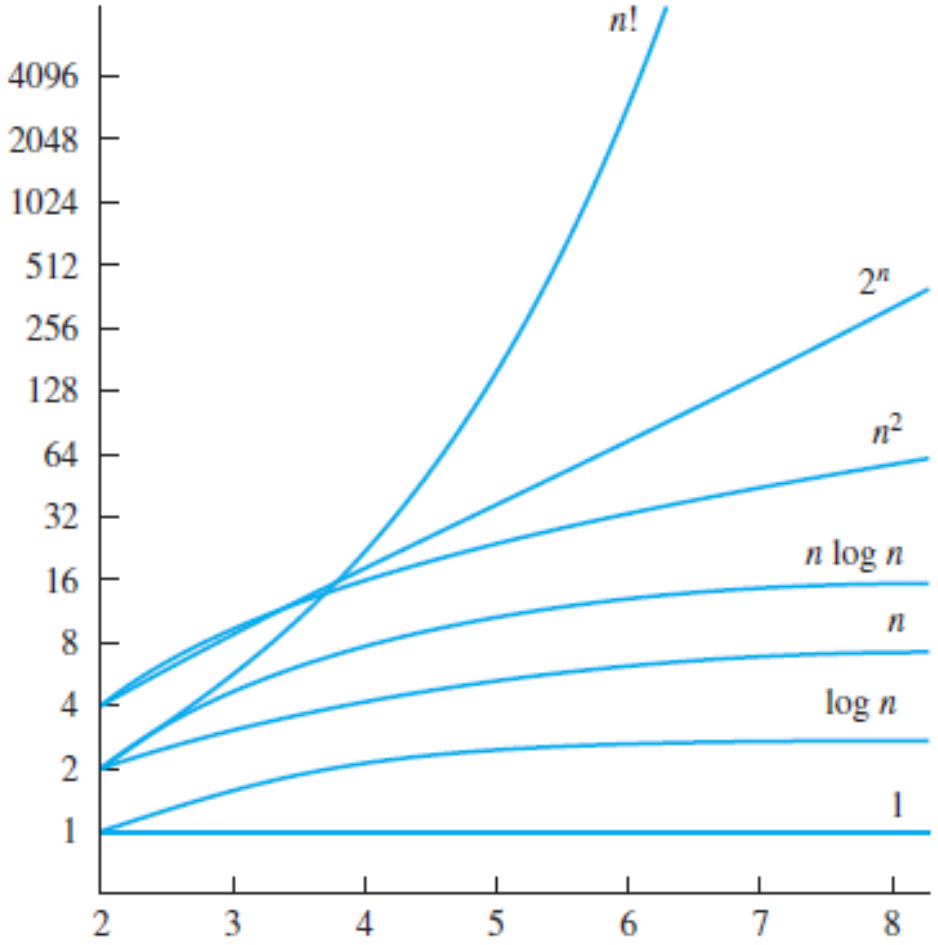


FIGURE 3 **A Display of the Growth of Functions Commonly Used in Big-*O* Estimates.**

Big Omega Notation (Lower Bound)

- "Let f and g be functions from the set of integers or the set of real numbers. We say that $f(x)$ is $\Omega\big(g(x)\big)$ if $\exists C, k > 0$ such that $|f(x)| \geq C \times |g(x)|$ whenever $x \geq k$"

- $f(x)$ is $\Omega\big(\mathrm{g(x)}\big)$ if and only if $g(x)$ is $O\big(f(x)\big)$.

Big-Theta Notation (Tight Bound)

- Useful when we want to give an upper and a lower bound on the size of a function $f(x)$ relative to some reference function $g(x)$.

- "Let f and g be functions from the set of integers or the set of real numbers. We say that $f(x)$ is $\Theta\left(g(x)\right)$ if $f(x)$ is $O\left(g(x)\right) \wedge f(x)$ is $\Omega\left(g(x)\right)$."

- If $f(x)$ is $\Theta\left(g(x)\right)$, we say that f is big-Theta of $g(x)$, $f(x)$ is of order $g(x)$, and that $f(x)$ and $g(x)$ are of the same order.

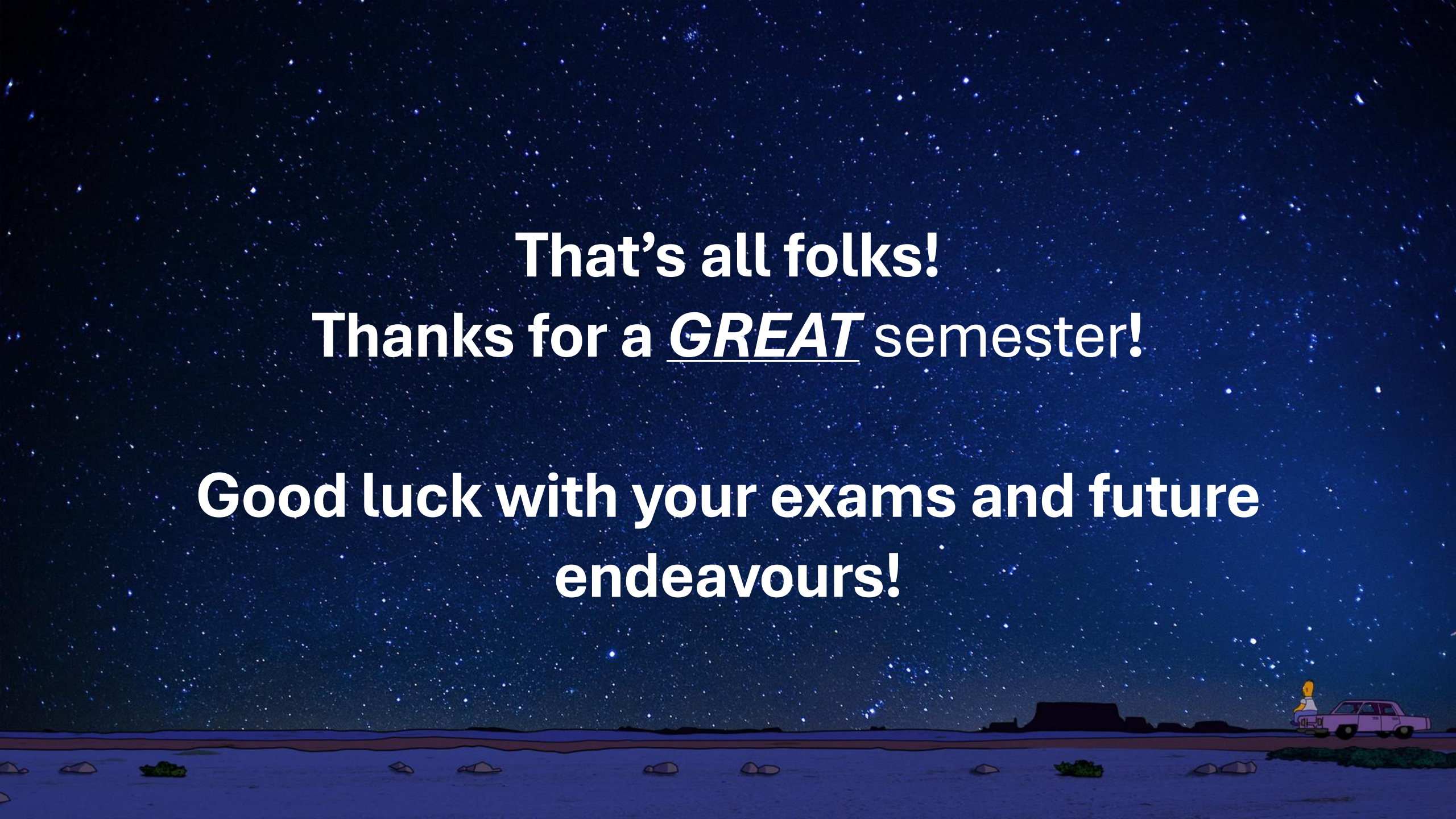
Putting It All Together

- Assume that we have a sorted list of size n. Our task is to determine whether an element exists within the list and to return its index.

    1. How can this be done in $O(n)$ time complexity?

    2. How can this be done in $O(log_2(n))$ time complexity?

- In LE/EECS 2101, you will have similar discussions around sorting lists. Some sorting algorithms run in $O\left(nlog_2(n)\right)$, however, they require certain preconditions to be met (e.g., list must be unsorted).


That’s all folks!
Thanks for a *GREAT* semester!
Good luck with your exams and future endeavours!

Thank You!

Shogo Toyonaga
Lassonde School of Engineering