Lab 02 – Complexity & friends

Instructor: Lorenzo De Carli, University of Calgary (lorenzo.decarli@ucalgary.ca)
Slides by Lorenzo De Carli

In this lab

- Make sure you have a working Python installation
- Refresh your knowledge of Python programming and syntax
- Make sure you are familiar with the tools you'll need in this class
- **We'll discuss submission format at the end of this presentation**

Exercise 1 - Divide and conquer

- **Memoization** is an approach to algorithmic optimization where intermediate results are saved in a cache for future reuse
- A ”cache” in this case is anywhere data can be accessed quickly, such as for example a Python array or a dictionary
- We are going to see how memoization can improve performance

Exercise 1 /2

The exercise is going to be based on this code

```python
def func(n):
  if n == 0 or n == 1:
    return n
  else:
    return func(n-1) + func(n-2)
```

Exercise 1 /3

- First, answer these questions:
  1. What does this code do? [0.1 pts]
  2. Is this an example of a divide-and-conquer algorithm (think carefully about this one)? [0.1 pts]
  3. Give an expression for the time complexity of the algorithm [0.2 pts]

Exercise 1 /4

4. Implement a version of the code which uses memoization to improve performance [0.2 pts]

5. Give an expression for the time complexity of the optimized algorithm [0.2 pts]

6. Time the original code and your improved version, for all integers between 0 and 35, and plot the results (output plots must be called ex1.6.1.jpg and ex1.6.2.jpg) [0.2 pts]

Exercise 1 – what to deliver

- Submit a Python file called ex1.py
- The file must contain:
  - The answer to question 1, 2 and 3 as code comments
  - The code implementing the answer to question 4 as a Python function
  - The answer to question 5 as code comments
  - Code implementing the timing requested by question 6
  - **Do not submit the output plots**

Exercise 2 – interpolation search

For this exercise, we are going to use the following code

```python
def interpolation_search(arr, x):
  low = 0
  high = len(arr) - 1
  while low <= high and x >= arr[low] and x <= arr[high]:
    pos = low + int(((float(high - low) / (arr[high] - arr[low])) * (x - arr[low]))
    if arr[pos] == x:
      return pos
    if arr[pos] < x:
      low = pos + 1
    else:
      high = pos - 1
  return -1
```

Exercise 2 /1

1. Mention at least two aspects that make interpolation search better than binary search [0.1 pts]
2. Interpolation search assumes that data is uniformly distributed. What happens this data follows a different distribution? Will the performance be affected? Why? [0.2 pts]
3. If we wanted to modify interpolation search to follow a different distribution, which part of the code would be affected? [0.1 pts]

Exercise 2 /2

- When comparing linear, binary and interpolation search:

4. When is linear search your only option for searching data as binary and interpolation search may fail? [0.2 pts]
5. In which case will linear search outperform both binary and interpolation search, and why? [0.2 pts]
6. Is there a way to improve binary and interpolation search to solve this issue? [0.2 pts]

Exercise 2 – what to deliver

- Provide the answer to questions 1-6 as a markdown (.md) file called ex2.md
- Visual Studio Code already supports editing .md files

Exercise 3 - profiling

- In class, we have discussed **benchmarking** as a way to measure code performance
- Now, let’s learn about **profiling**

Exercise 3 /2

We'll use this code for the exercise

```
import timeit

def sub_function(n):
  #sub function that calculates the factorial of n
  if n == 0:
    return 1
  else:
    return n * sub_function(n-1)

def test_function():
  data = []
  for i in range(10):
    data.append(sub_function(i))
  return data

def third_function():
# third function that calculates the square of the numbers from 0 to 999
  return [i**2 for i in range(100000000)]

test_function()
third_function()
```

Exercise 3 /3

- First learn about the cProfile module in Python
  https://docs.python.org/3/library/profile.html

- Then, answer the following questions:

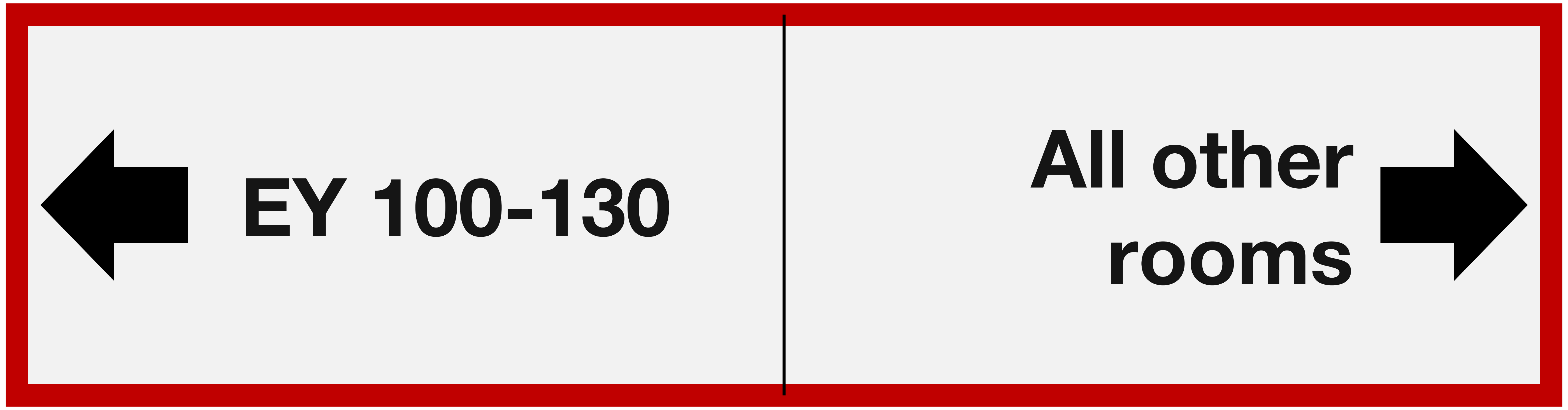
1. What is a profiler, and what does it do? [0.25 pts]
2. How does **profiling** differs from **benchmarking**? [0.25 pts]
3. Use a profiler to measure execution time of the program (skip function definitions) [0.25 pt]
4. Discuss a sample output. Where does execution time go? [0.25 pts]

Exercise 3 – what to deliver

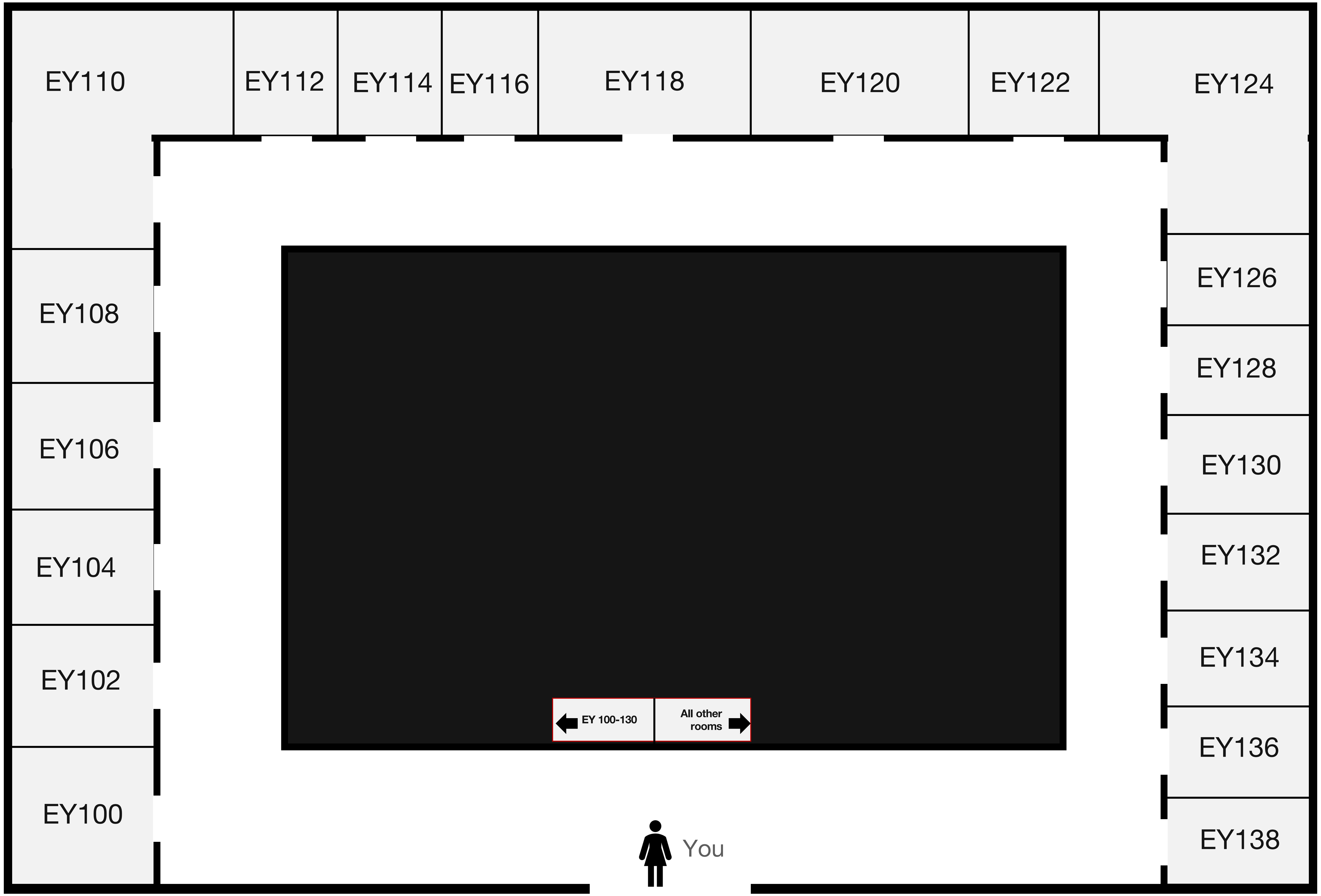
- Deliver a Python file named ex3.py
- The code should implement answers to question 3
- Answers to questions 1-4 should be provided in the comments to the same code

Exercise 4 – todo

- Imagine there is a new engineering building on Ucalgary campus ("Engineering Y"). You enter the building on the ground floor and you need to attend a lab in EY128.
- As you enter, you find the sign below:



Exercise 4 /2 The floorplan of the first floor is described below



Exercise 4 /3

1. Describe the algorithm you will use to find the room. **Assume all the information you have is the one given by the sign; you have no knowledge of the floor plan** [0.5 pts]

2. How many ”steps” it will take to find room EY128? And what is a “step” in this case? [0.25 pts]

3. Is this a best-case scenario, worst-case scenario, or neither? [0.25 pts]

4. With this particular sign and floor layout, explain what a worst-case or best-case scenario would look like [0.5 pts]

5. Suppose after a few weeks in the term you memorize the layout of the floor. How would you improve the algorithm to make it more efficient? [0.5 pts]

Exercise 4 – what to deliver

- Provide the answer to questions 1-5 as a markdown (.md) file called ex4.md

Exercise 5 – visualizing complexity

- In this exercise, you will implement basic search algorithms and attempts to confirm theoretical complexity findings with empirical measures

1. Implement linear search and binary search [0.5 pts]
2. Measure the performance of each on sorted vectors of 1000, 2000, 4000, 8000, 16000, 32000 elements . In each case, you must do the following for 1000 times, and compute the average [0.5 pts]:
   a. Pick a random element in the vector
   b. Measure the time it takes to find the element using timeit, using 100 iterations (number=100)

Exercise 5 /2

3. Each plot should also interpolate the data points with an appropriate function. For example, linear complexity with a linear function, quadratic complexity with a quadratic function, etc. [0.5 pts]

   - You already know how to fit a linear function (from class). Fitting more complex functions can be accomplished using `scipy.optimize.curve_fit()`

4. Discuss the results. For each interpolating function, describe (1) the type of function, and (2) the parameters of the function. Are the results what you expected? Why? [0.5 pts]

Exercise 5 – what to deliver

- Provide the answer to questions 1-3 as a Python file named ex5.py. The code file must include implementations of the algorithms, timing code, and plotting/interpolation code.
- Provide the answer to question 4 as comments within the same code.

What to deliver

- Upload a zip file to the “Lab 2” dropbox on D2L, containing the required content for every exercise:
  - ex1.py
  - ex2.md
  - ex3.py
  - ex4.md
  - ex5.py

Grading rubric

- You get **3 pts** for uploading a **partial solution** by **end of lab**
  - **Must not be an empty file or irrelevant material**
- Then, you'll have until **11:59PM of the day before the next lab** to upload the **complete solution**. That will be graded as follows:
  - Exercise 1: 1 pts
  - Exercise 2: 1 pts
  - Exercise 3: 1 pts
  - Exercise 4: 2 pts
  - Exercise 5: 2 pts
  - **Can upload the complete solution to the same dropbox**

That’s all folks!