Stelios Sotiriadis

1. Introduction to Big Data Analytics

About me!

- Lecturer at CS, PhD in distributed systems
- I worked to solve modern industry-based problems to improve systems' performance
  - Projects at the University of Toronto with Huawei, Autodesk, IBM and other companies and various startups worldwide

Agenda

- Introduction to BDA
  - Module admin
- Big data analytics and algorithms
- Computational complexity
- Searching and sorting (part 1)
- Python lab → Live coding intro + Race Lab

Module administration

BDA is designed to be in person

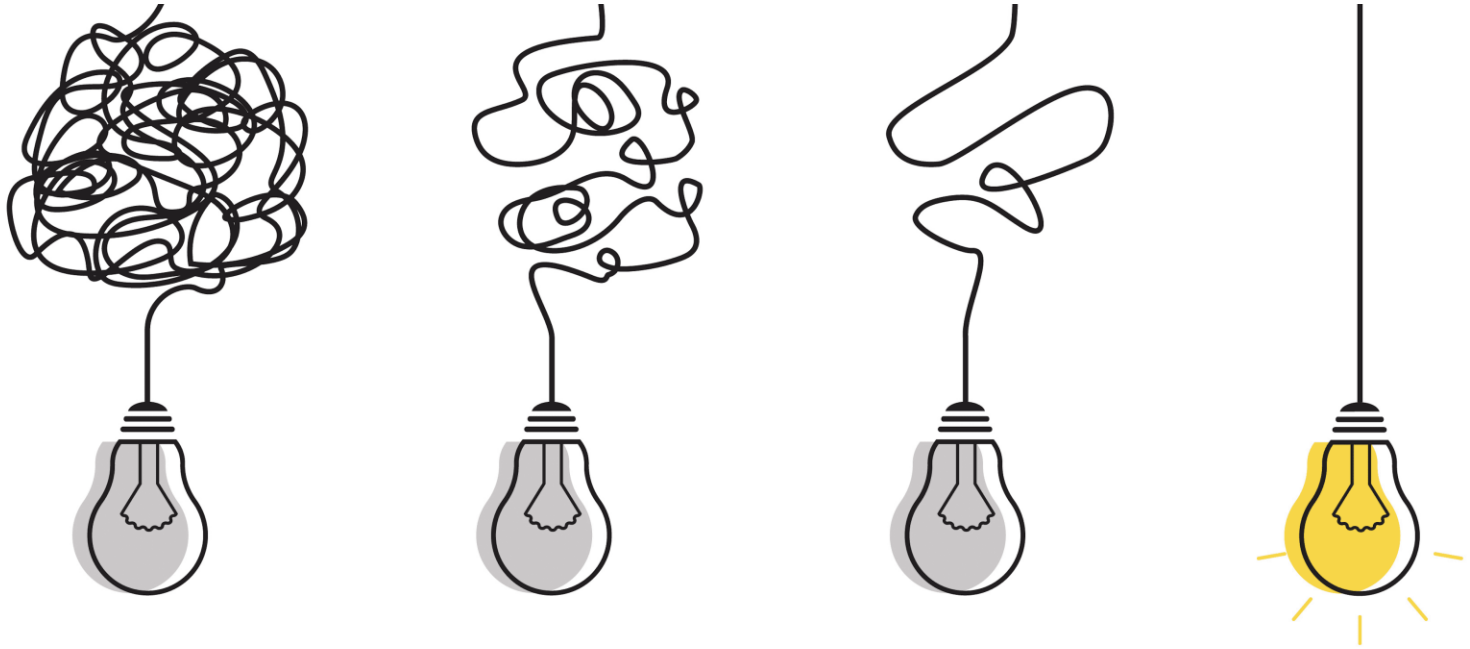
- Lecture sessions are in Malet St MAL B20
- Lab sessions are in Malet St MAL—404/405
- Classes are streamed and recorded – but the fun is on the face-to-face
- The class starts at 2 pm
  - Please join 5 minutes earlier
  - ~3.30 pm we move to the lab room
- Bring laptops/tablets to lectures/pen and paper

Let's have an interactive class!

- Interrupt me any time you like
- Make mistakes…
- If you don't get it, ask!
- Be gentle with MS Teams if you join online…

What is this module about?

Small solutions for big problems!



Big data analytics

- Theory:
  - Fundamental algorithms
  - Data processing methods
  - Parallel and concurrent processing
  - Working with unstructured data
  - Data processing pipelines
  - Working with processing frameworks
- Practice:
  - Working with unstructured data sources with Python and its libraries
  - Advanced databases like Cassandra and MongoDB
  - Use of data processing frameworks like Hadoop MapReduce and Spark

Who should take this module

- This is a module for those who want to become programmers:
  - Programmers vs. Coders?
  - Software developers who want to learn solutions for developing applications using data!

How to contact?

- **MS Teams (preferred way)**
  - Feel free to send at any time
- My office:
  - Room 151A (Office hours: Wednesday 11-2pm)
- Virtual Office hours:
  - Any day and anytime, including weekends ☺
- Me email:
  - s.sotiriadis@bbk.ac.uk

Prerequisites

- **Excellent knowledge of programming with Python**
  - Or willing to work hard to learn more!
- Experience with advanced programming methods
  - Object-oriented programming, Functional programming, exception handling, etc.

This is a heavily involved programming class!

The module is not for you if…

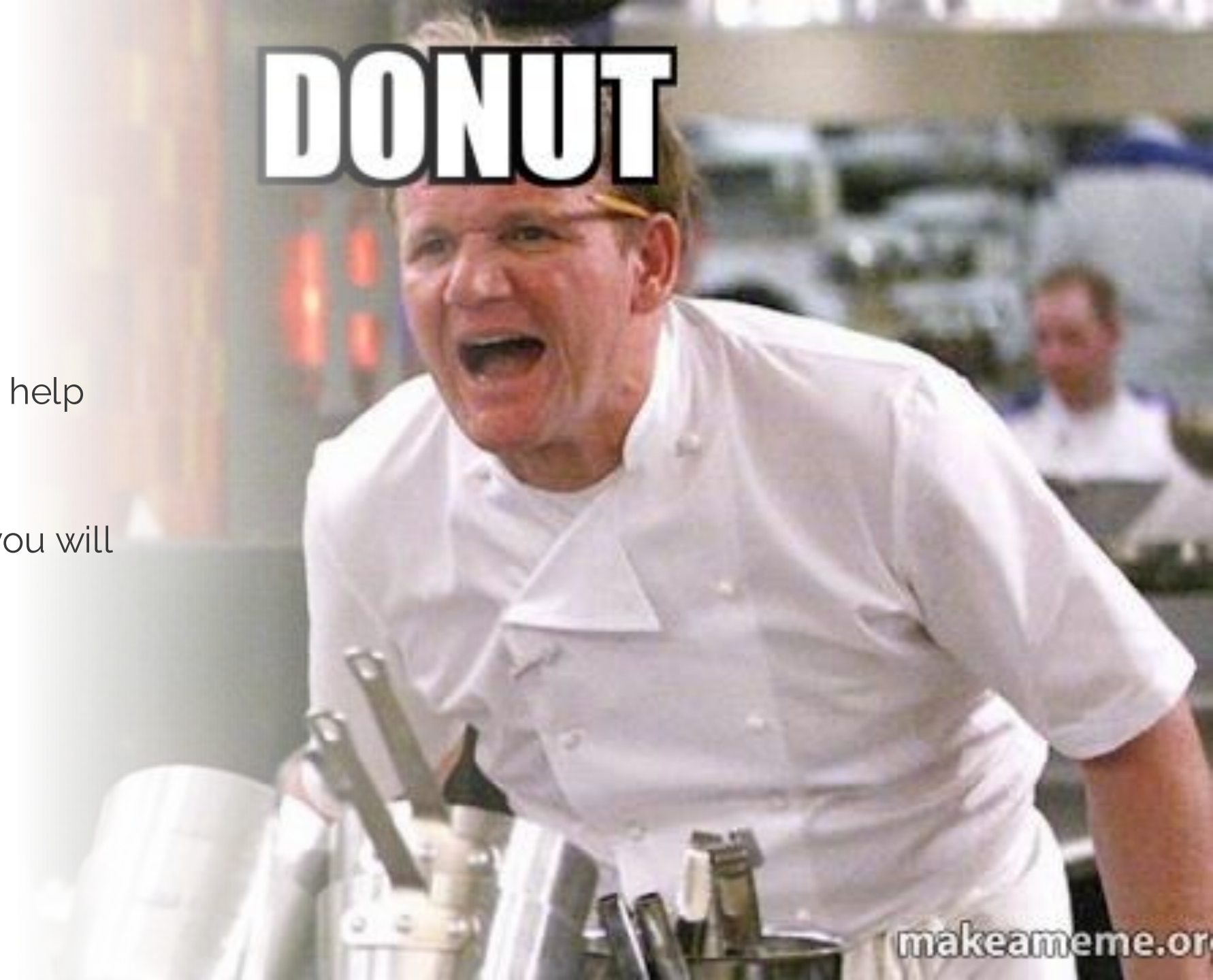
- You never used Python before
  - Basics and desire to learn more Python is enough
- You are not genuinely interested in the topic of **PROGRAMMING** for big data
- You cannot put in the time: **1 hour in class requires 2-5 hours of study** outside of the class

Assessment

- There is no exam ☺
- 100% coursework (split into assignments)
  - **All tasks are programming-based with Python**
- There are two assessments:
  - Week 3 (released): Programming-based assignment (Python) – 40%
    - Web-crawler using parallel programming.
  - Week 7 (released): Programming-based assignment (Python) – 60%
    - Analytics of unstructured data.
  - **Four weeks for each assignment to complete.**

Use of ChatGPT

- Yes! use it as a tool to help you learn.
- Don’t copy answers, you will miss all the fun!



Questions?

- Ready to start?

Are you ready to start?

Patience you must have...

*Yoda

Let’s start with a short quiz!

Introduction to Big Data

What is big data about?

What is big data about?

- Amount of data
- Processing of data
- Real-time data
- Moving or transforming data

Real estate data example

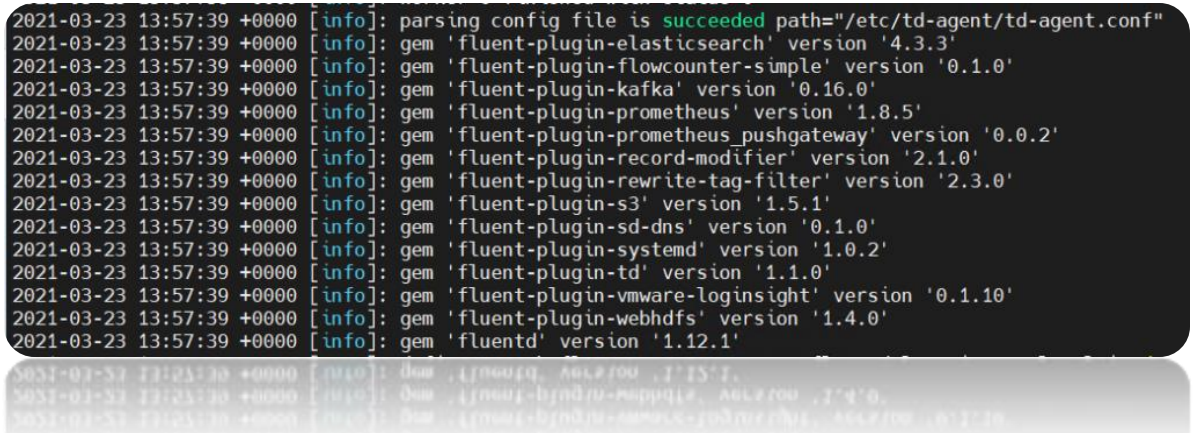
- Real estate company in London
  - 120GB of real estate data with US addresses
    - 160 million addresses (low dimensional data)
    - The provider gave us a CSV file (all the data)
  - Decide how to store the data
  - Decide how the app will use the data
  - Data needs to be updated once per month
    - New address added – fixes required

Real-time data analysis example

- Use case:
  - A seizure is a sudden surge of electrical activity in the brain that affects how a person appears or acts for a short time.
- Healthcare startup in Canada.
  - Requirement: 10GB of high-modality brain signal data
    - There were 24 channels to collect data - data streamed every 7 minutes.
    - Analyse data and decide within 7 minutes.
  - Decide how to collect, store and process data.
  - Run a complex ML algorithm and identify if there is a seizure.

Log data analytics example

- Analyse log data from a database and determine if there are issues or anomalies.
- Requirement:
  - Apache Cassandra (NoSQL database)
  - Our setup generated 20TB ≈ 3.33 trillion words of data per month!
- Problem: We don't want to read the log files physically! → Impossible…
- Solution: Somehow, use a mathematical model to read our logs.



Big data analytics

- A set of methods and techniques to solve data-related problems.
- Related mainly to:
  - Data input/output operations
  - Computational operations

Velocity – Variety – Volume

Velocity

- Batch:
  - Batch processing is about periodically processing high-volume, repetitive data jobs.
  - When you can wait for days (or longer) for processing (Payroll is a good example.).
- Near-time:
  - When speed is essential, but you don't need it immediately.
  - Speed is essential but not instant.
- Real-time:
  - When you need information processed immediately (such as at a bank ATM).

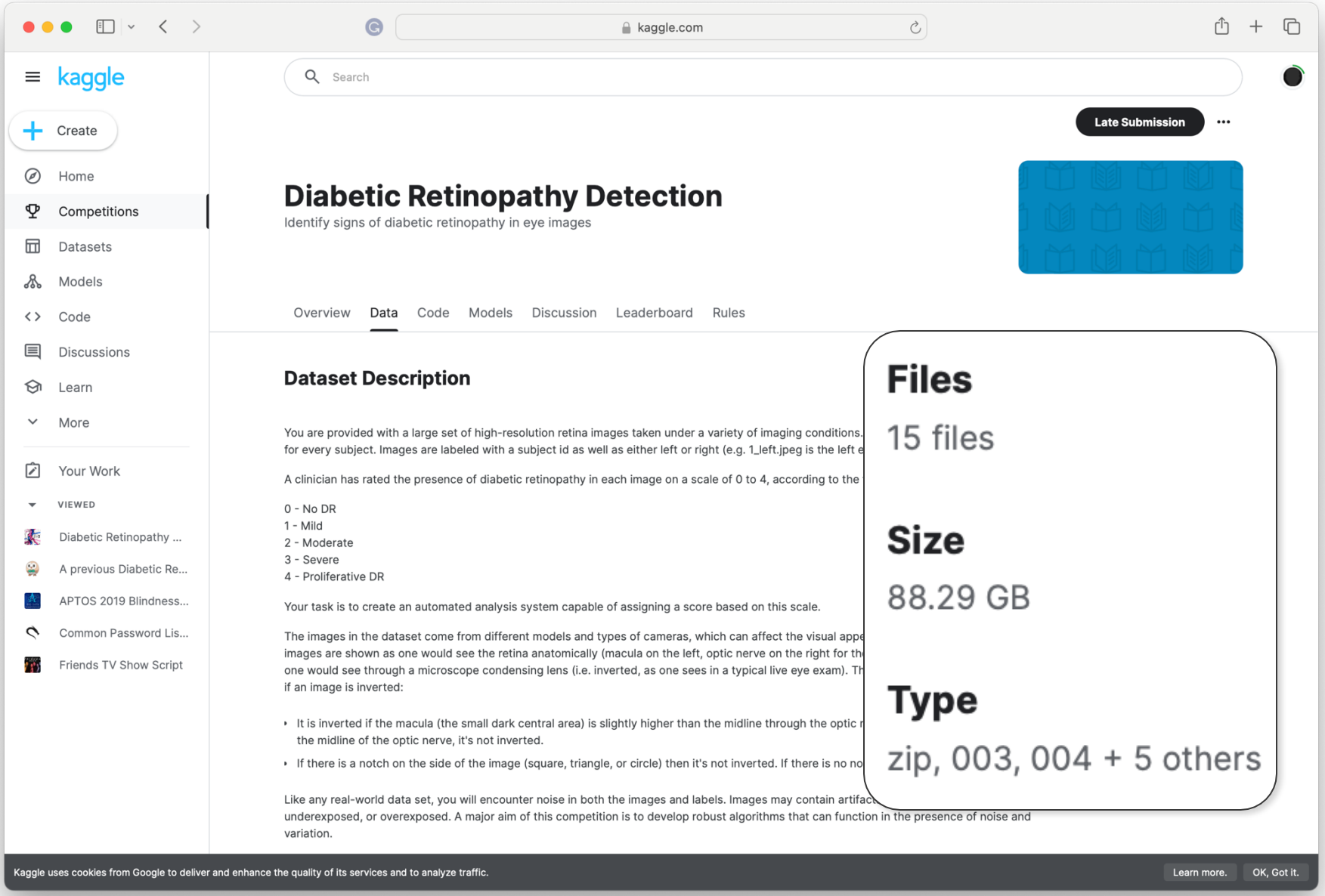
Variety

- Structured
  - A customer database in a relational database management system (RDBMS) like MySQL, PostgreSQL, or Oracle.
- Unstructured
  - Customer feedback is collected through emails.
- Semi-structured
  - A JSON file containing user information.

[Link]

Volume

- Size of data.



Byte to Brontobyte

Unit	Equals	# Words
1 Bit	Binary digit	0/1
8 Bits	1 Byte	1 word (ASCII)
1024 Bytes	1 Kilobyte	≈ 166 words
1024 Kilobytes	1 Megabyte	≈ 166,667 words
1024 Megabytes	1 Gigabyte	≈166.67 million
1024 Gigabyte	1 Terabyte	≈166.67 billion
1024 Terabytes	1 Petabyte	≈166.67 166.67 trillion
1024 Petabytes	1 Exabyte	≈166.67 quadrillion
1024 Exabytes	1 Zettabyte	≈166.67 quintillion
1024 Zettabytes	1 Yottabyte	…
1024 Yottabytes	1 Brontobyte	…

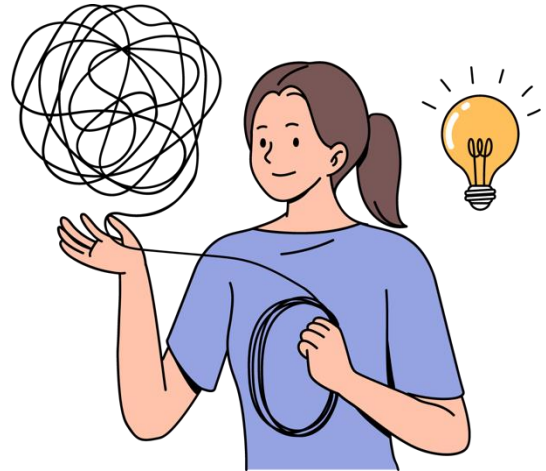
Big data

Why 1024B=1KB and not 1000?

- The **binary number system is base 2**, while the decimal number system is base 10.
- In the binary number system, every digit has a value of 1, 2, 4, 8, 16, etc. (powers of two).

Having a problem to solve…

- How to start?
  - Take a decision!
    - Is your problem CPU-bounded?
    - Is your problem Input/Output-bounded?
    - Or is it both?

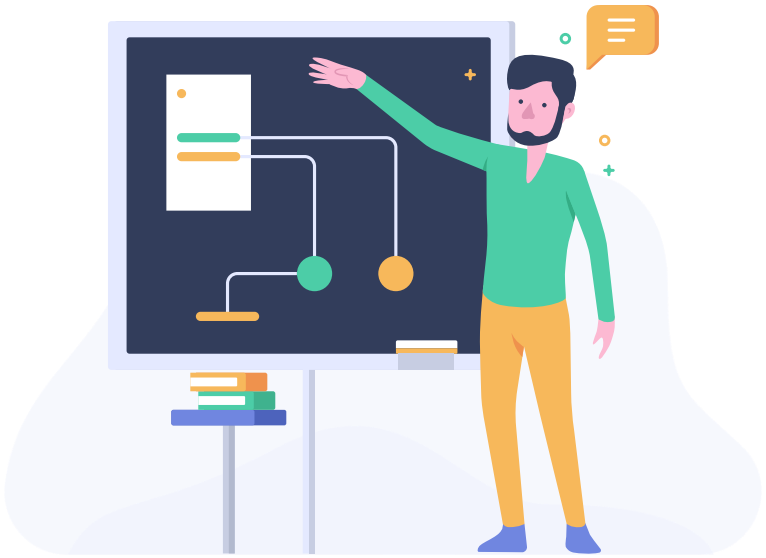


Is it **CPU** or **data**-intensive?

- Rotating thousands of images — CPU-intensive (Matrix multiplication)
- Storing user input to text files — Data-intensive
- Blurring thousands of images — Both!

Having a problem to solve, how do we start?

- Algorithms!
  - A set of steps to solve a problem.
- But is there only one way?
  - We must always answer the following question:
    - Is there a better way?
  - We need to find a way to quantify the performance of our algorithms (programs)



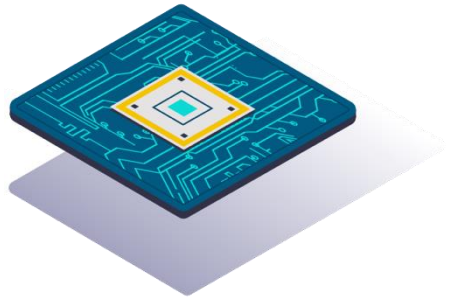
Algorithms can be complex

- Example:
  - Download stock market data from the web
  - Clean it
  - Transform it to preferable format
  - Store it in a database
  - Apply ML to analyze it and find patterns to invest in
  - Get rich!

- This program includes sub-programs (sub-algorithms).
- Our programs can be seen as a set of tasks!
  - Computationally or data-intensive tasks

Computationally intensive tasks

- Use your mind to calculate 25*25
- Operations requiring significant computational power → CPU
  - Multiplying two matrices, running a machine-learning model
- These tasks demand CPU/GPU and memory.



Data intensive tasks

- Find and count how many times the word "London" exists in the British Library resources
- Using a large volumes of data
  - Require significant data bandwidth and storage capabilities
- It is more about the amount of data rather than computations.

How do we create GOOD programs?

Does it make any difference (1)?

Script 1

script

```
numbers = [10,20,30,40]

print(numbers[-1])
```

That's better!

```
40
```

Script 2

script

```
numbers = [10,20,30,40]

for i in numbers:

  pass

print(i)
```

```
40
```

Does it make any difference (2)?

Script 1

script

```
arr = [1, 5, 2, 8]

results = []

for number in arr:

    if number >= 5:

        results.append(number)

print(results)
```

```
[5,8]
```

Script 2

script

```
arr = [1, 5, 2, 8]

for i in range(len(arr)-1, -1, -1):

    if arr[i] <5:

        arr.pop(i)

print(arr)
```

That's better (if we don't want to use an extra list)!

```
[5,8]
```

Operations & Space matters!

- Operations:
  - Number of iterations to solve a problem
  - How many times a for loop runs
- Space:
  - Amount of space needed to solve a problem
  - How many data do you store, temporarily or not

How do I quantify the quality of my solution?

Computational complexity

Time complexity examples

- Whenever you visit your local grocery store for some chocolate…
  - You know precisely which aisle/shelf has your favourite chocolate – you walk in and grab it.

    `Constant time (just one operation) – O(1)`
  - You need to find out its location, so you check each shelf in search of chocolate, one by one.

    `Linear time (one by one) – O(n)`
  - You are sorting all chocolates according to cocoa volume.

    `Quadratic time (compare all with all) – O(n^2)`

Space complexity examples

- You now grab a basket!
  - You find and pick your chocolate. Then you put your chocolate in your basket.

    Constant space (just one space used) – O(1)
  - You are unsure, you put all the chocolates in your basket, then you look again and select the one you prefer.

    Linear space (n space) – O(n)

Running time of an algorithm

- Algorithms:
  - Transform input into output
- Paradox:
  - Run time is not about the time to run a program!
  - It is about the amount of operations.

How many operations?

script

```
arr = [10,20,30,40]
  for i in arr:
    print(i)
```

4

script

```
arr = [1,2,3,...,n]
  for i in arr:
    print(i)
```

n

script

```
arr = [-10,-5,0,10,33]
  for i in arr:
    if i>0:
      print(i)
```

5

- We don't care about 4 or 5 - it is always **n**!
- Operations = **len(arr)**

Time complexity

Time complexity is about number of operations!

- Describes the amount of time it takes to run an algorithm as a function of the input length.
  - Expressed as Big O notations
- Best Case: The minimum time of program execution. **`Big-Omega(Ω)`**
- Average Case: The expected time for program execution. **`Big-Theta(Θ)`**
- Worst Case: The maximum time required for program execution. **`Big-Oh(O)`**

Big O notations

- Searching an element of a list
- Best Case

  Ω(1) - Element is first ☺

- Average Case

  Θ(n/2) - Element in the middle ☺

- Worst Case

  O(n) - Element is last ☹

- Which case should we focus on the most?

  Big - Oh - Worst case!

Space complexity

- The total amount of the working storage that an algorithm needs.
- This memory usage might include not only the variables but also the space required for the input and output data:
  - Total Space: Includes input data space, auxiliary space (extra or temporary space), and output data.

Big O

- **`O(1) [Constant time]`**
  - Accessing an element in an array by index.
- **`O(n) [Linear time]`**
  - Linear search in an unsorted list or array.
- **`O(n^2) [Quadratic time]`**
  - Nested loops where each loop iterates over the entire input size.

Group quiz

- What is the computational complexity?
  - Picking a random card from a deck of cards **O(1)**
  - Picking the last card **O(1)**
  - Searching for the first King of Spades **O(n)**
  - Counting the Queens **O(n)**
  - Sorting the cards **O(n^2)**
  - Picking the fifth card at once **O(1)**

Linear Search algorithm

script.py.md

```
def linearSearch(arr,key)
  for i in arr:
    if i==key:
      return True
  return False
```

- An array of elements and a key.
- Does the key exist in the array?

Let's see an example

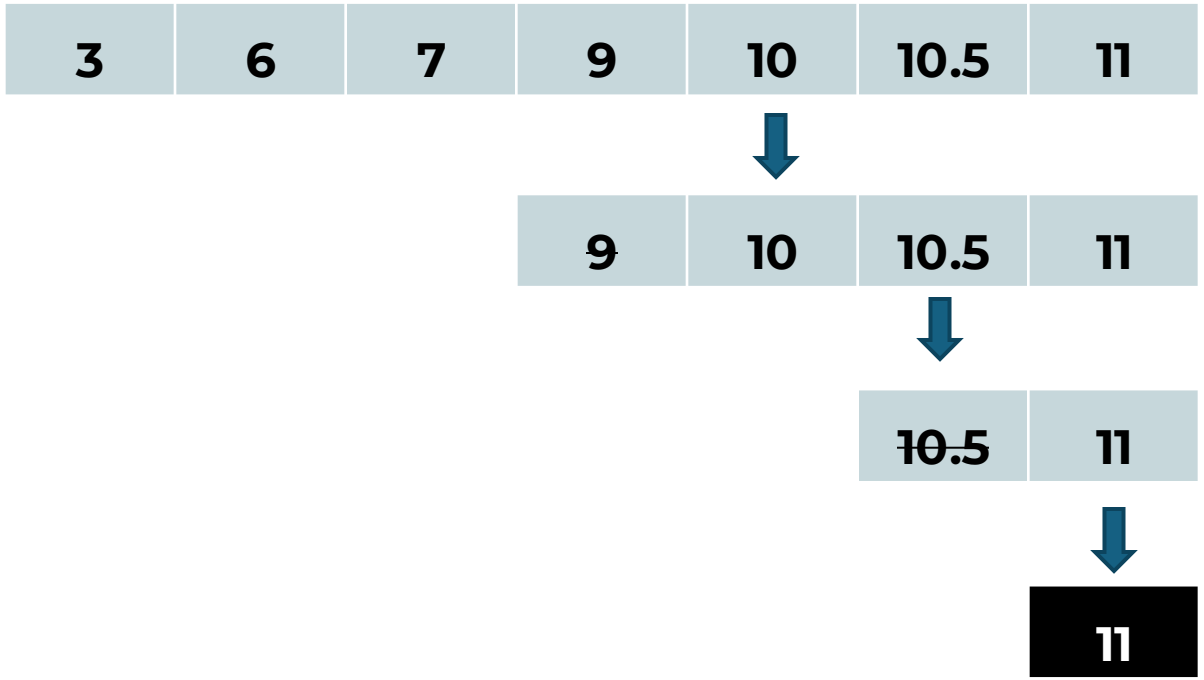
- **`A = [4, 9, 12, 16, 18], key=6`**
- In principle, there are ***`n`*** comparisons before we can say "no".
  - We will spend five operations to search for a key.
  - We care about the worst-case scenario: **`5`** operations = **`O(5)`** = **`O(n)`**.
- What about this case?
  - **`B = [4, 9, 12, 16, 18], key=4`**
  - Still, we care about the worst-case scenario… **`O(n)`**

Group quiz

- You just opened a new deck of cards!
  - You removed the two jokers
  - You have 52 cards in order!
  - What is the time complexity to search for a King of hearts? **Position 39!**
- What if you don't know? **Linear search (n)**
- Can you do it better? **Yes**

Can we do better?

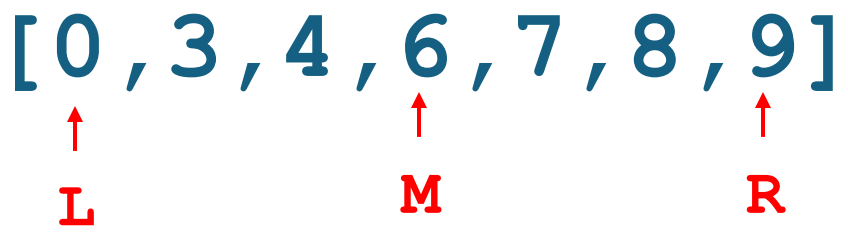
- `A = [3, 6, 7, 9, 10, 10.5, 11], key=11`



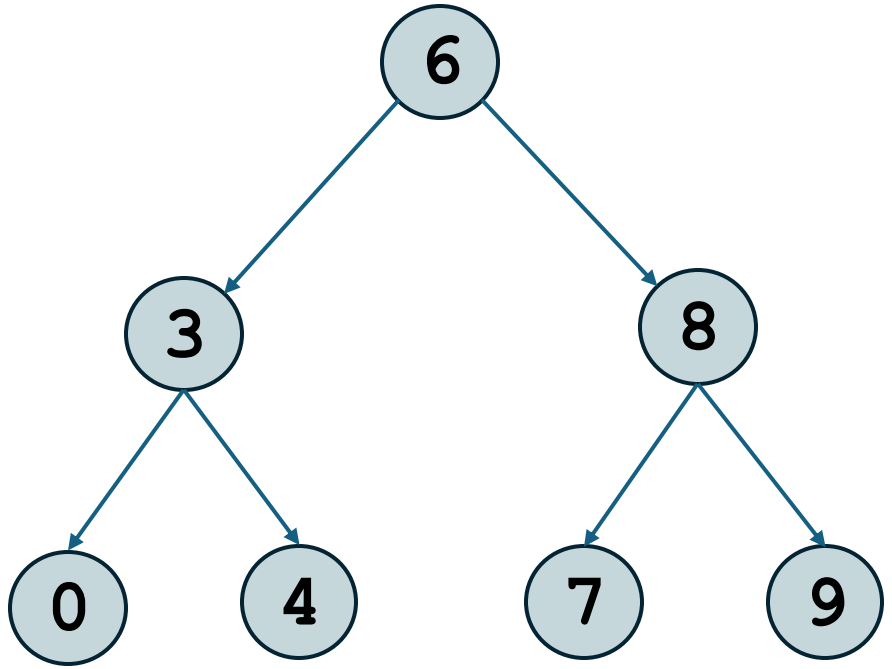
The search space is repeatedly divided in half during each iteration…

`O(logn)`

Binary search example



Binary Search



Binary Search Tree

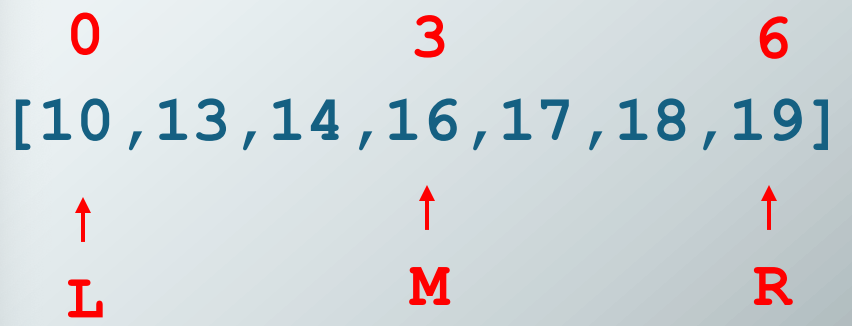
Binary search

- Search in a **sorted array** by repeatedly dividing the search interval in half.
  - Begin with an interval covering the whole array.
  - If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
  - Otherwise, narrow it to the upper half.

script

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

target=13

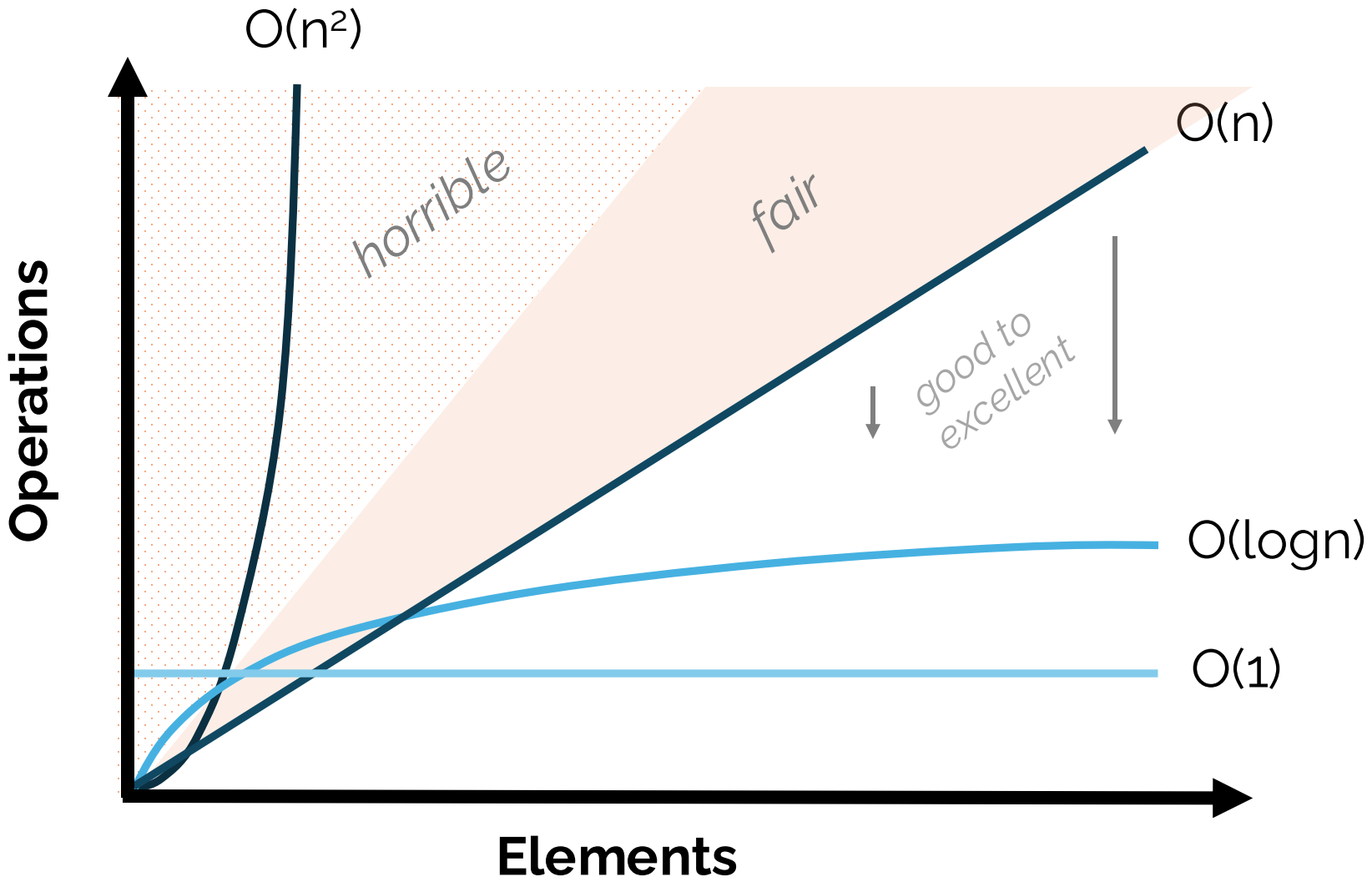


💡 Quiz

- How many iterations to find 23? **3**

```
[2,5,8,12,16,23,38,56,72]
```

Big-O Complexity Chart



Group quiz

- Write down all the possible permutations of ABC?

- **ABC**
- **ACB**
- **BAC**
- **BCA**
- **CAB**
- **CBA**

Can you think of a mathematical formula to generalise this example?

```
3! = 1*2*3 = 6
```

O(N!) [Factorial time]

- Generating all possible permutations of a set of elements is inherently factorial.
  - Consider a set of N elements
  - There are N! permutations to be generated

Group quiz

- You have been tasked with guessing a password.
  - The password includes three digits, each of which can be either `1` or `0`. A digit may appear multiple times or not at all in the combination. Write down all the possible combinations!
  - Can you think of a mathematical formula to generalise this example?

```
2^3 = 8 = 2^n
```

- `000`
- `001`
- `010`
- `011`
- `100`
- `101`
- `110`
- `111`

O(2^N) [Exponential time]

- All possible combinations!
  - Scenario: In security testing or hacking, generating all possible passwords up to a certain length, given a specific character set (brute-force password cracking).

Thank you!

- The lab starts soon (404-405)!

O(no!)

Lab 1

Big Data Analytics

Prepare your workspace

- Download and install the latest Python 3
  - https://www.python.org/downloads/
- Download and install Visual Studio Code
  - https://code.visualstudio.com
- Install Git and have a GitHub account ready

Setting up your working environment

1. Let's explore VSC together and the terminal
2. What is a virtual environment, and how do you set it up?

Race lab

Home activities

- Complete Lab1 activities and exercises
- Run the lab in your preferred Python editor
  - I recommend using Visual Studio Code
- Keep your notes as you prefer
- https://github.com/warestack/bda

Take home

- What are CPU and I/O bounded tasks?
- What is computational complexity?
- Discuss in practice how complexity affects code performance.