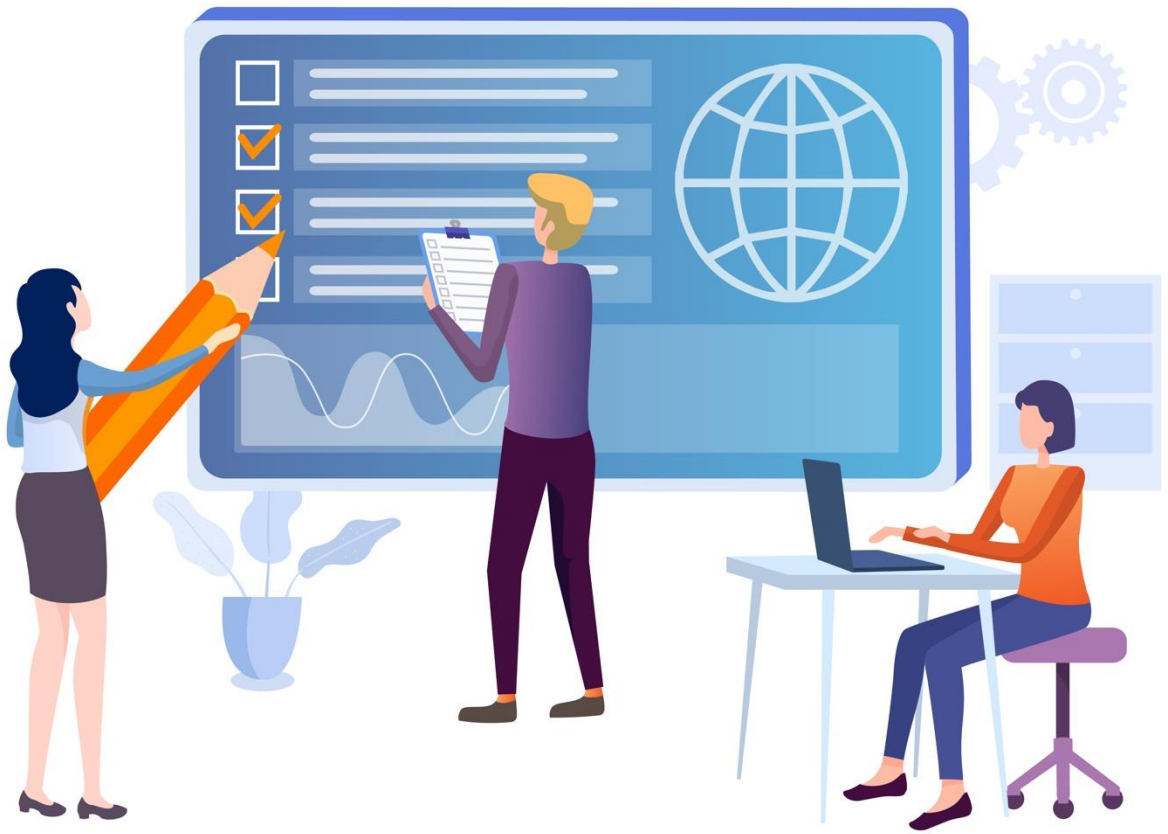
Stelios Sotiriadis

3. Sorting and Data Streams

Quiz of the day

Get ready!



Agenda

- Sorting methods
- Working with text files
- Data streams
- Python **`iterators`** vs **`generators`**
- Eager vs Lazy evaluation
- Data Analytics Project!
  - CSV Analysis with Python (Netflix Dataset)

Sorting methods

Bubble sort

8 5 3 1 4 7 9

Quick quiz on Bubble sort!

- Worst case time complexity:
  - `n · (n – 1) / 2 → O(n^2)`
    - We focus only on the dominant term → n^2
- Worst case space complexity:
  - `O(1)`
- Does Bubble Sort change the order of equal elements?
  - No! That's why we call it **stable algorithm**
- What does Bubble Sort do in each pass through the array?
  - ~~Finds the smallest element and moves it to the start~~
  - B) Finds the largest element and moves it to the end ← **Correct!**

8 5 3 1 4 7 9

What if?

- **I want Bubble Sort to find the smallest element and move it to the beginning of the array.**
  - Instead of:
    - `if arr[j] > arr[j + 1]`
  - Use:
    - `if arr[j] < arr[j + 1]`
  - Iterate from right to left in the inner loop, so small values move to the left.

Summary of worst-case time/space

- Time:
  - Repeatedly traversing the array and comparing adjacent items, swapping them if they are in the wrong order.
    - Each complete pass through an array of length `n` compares elements in pairs, leading to `n-1` comparisons (first pass), `n-2` (second), and so on, down to `1` comparison in the last pass.
    - `(n-1)+(n-2)+⋯+1 =` $\frac{n(n-1)}{2}$ `≈ O(n^2)`
- Space:
  - `O(1)` – in-place swapping, no extra space is used ☺

Insertion sort

6 5 3 1 8 7 2 4

Insertion sort

- Iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list
  - It is a stable sorting algorithm:
    - Elements with equal values maintain relative order
- Worst-case time complexity: `O(n^2)`
- Worst-case space complexity: `O(1)`

Merge sort

6 5 3 1 8 7 2 4

Merge Sort

- Divides the array into two halves, sorts each half, and then merges the sorted halves.
  - Worst-case time complexity: $O(n \cdot \log n)$
- But!
  - During the merge step, it creates temporary arrays to hold left and right halves, which uses extra space.
  - Worst-case space complexity: $O(n)$

Why is $O(n \cdot \log n)$?

- Why $\log n$?
  - Because each time you divide the array in half:
  - First: n elements
    - Then: $n/2$
    - Then: $n/4$
    - until you reach 1 element
  - This takes $\log n$
- Why n at each level?
  - At each level, you merge all n elements back together:
- Merging n elements takes linear time — $O(n)$
- Finally:

$$O(n) \times O(\log n) = O(n \cdot \log n)$$

Pen & Paper: Order bottom (best) to top (worst)

O(logn)

O(n^2)

O(n!)

O(2^n)

O(nlogn)

O(1)

O(n)

Pen & Paper: Order bottom (best) to top (worst)

O(n!) - Factorial

O(2^n) - Exponential

O(n^2) - Quadratic

O(nlogn) - Log-linear

O(n) - Linear

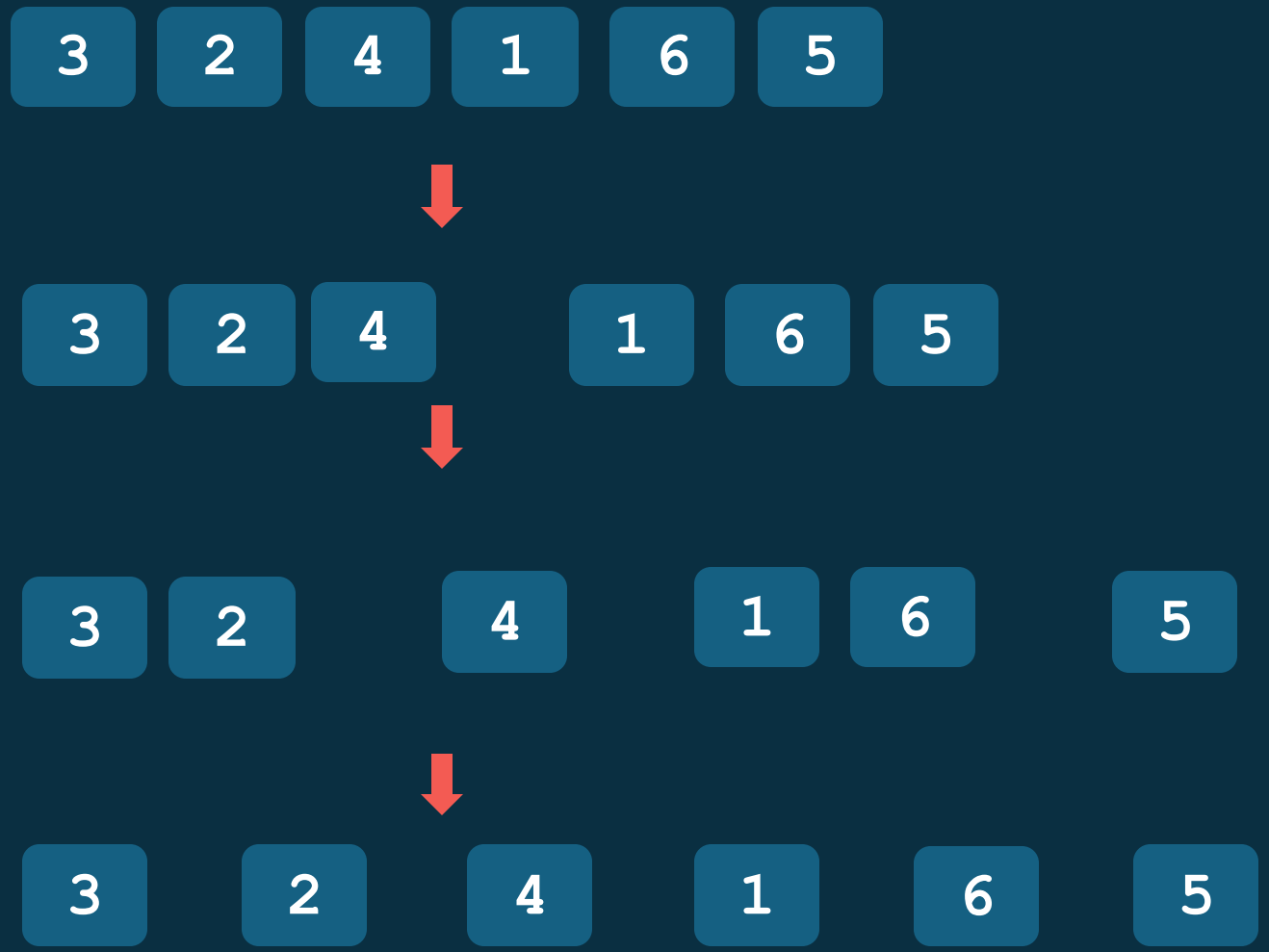
O(logn) - Logarithmic

O(1) - Constant

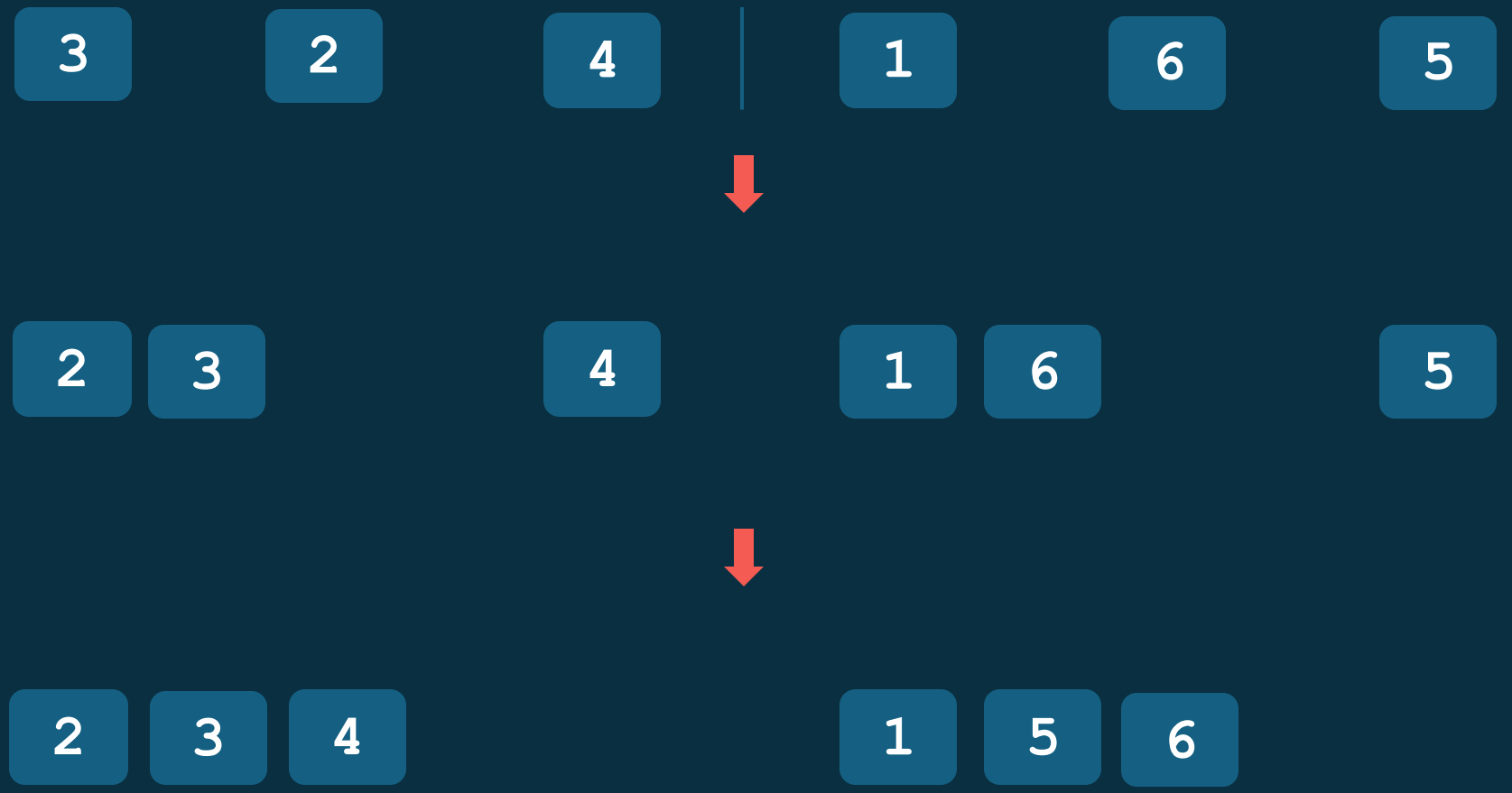
Let's do it together!

- Sort the following numbers using merge sort!

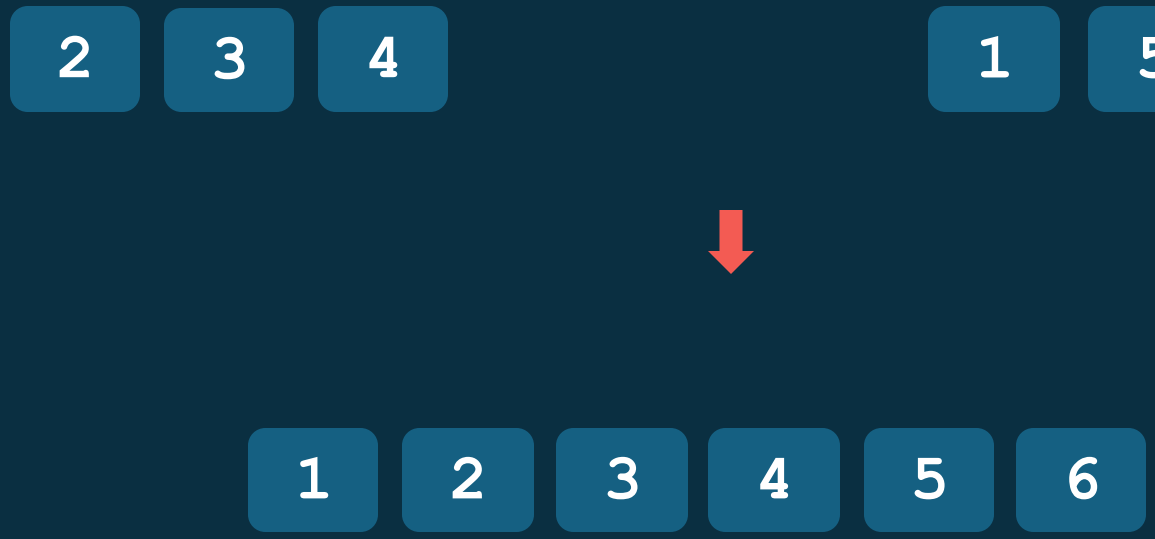
Step 1: Split the Array



Step 2: Merge and Sort



Step 3: Final merge



Insertion sort `O(n^2)`
Can you do it better?

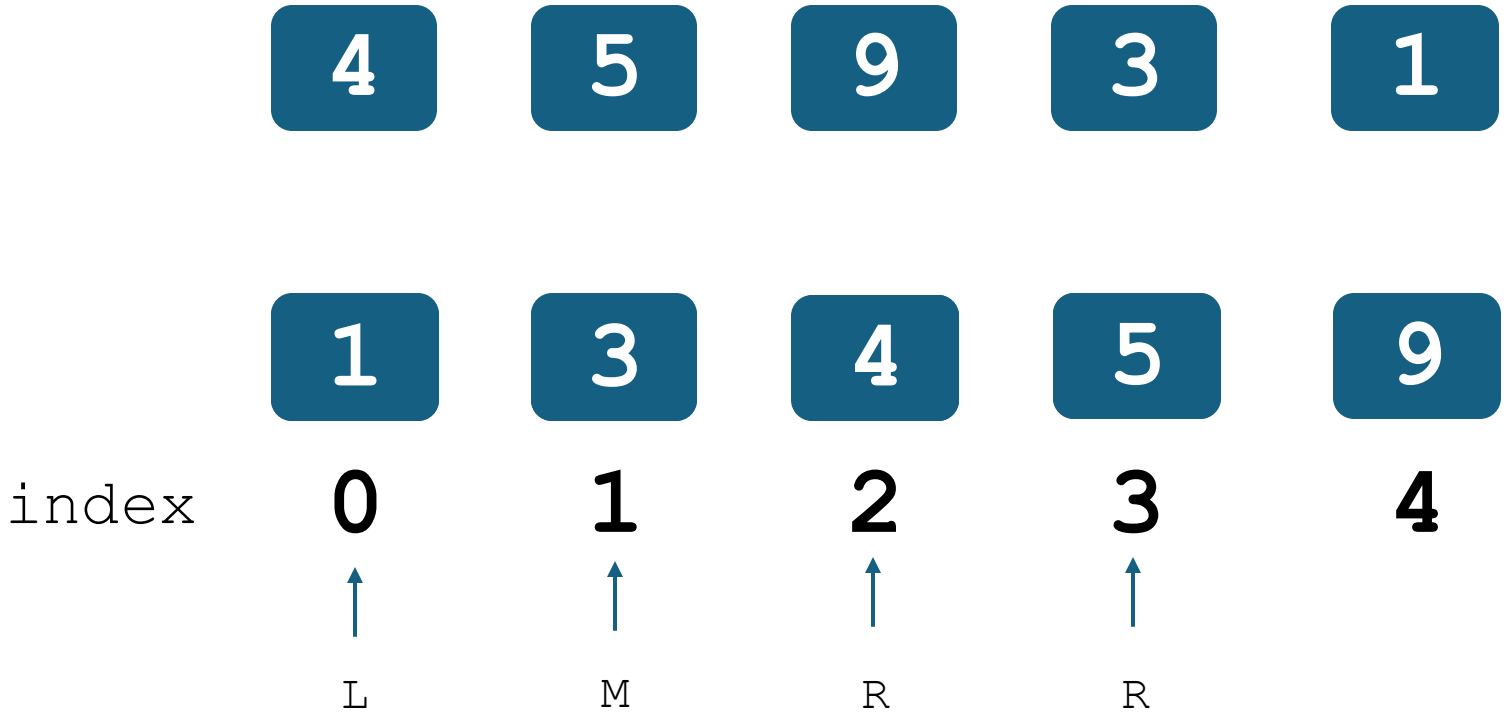
Yes! Use binary search to find the correct insertion point!

Tim sort

- It was implemented by Tim Peters in 2002 for use in Python.
- Tim Sort is the default sorting algorithm used by Python's `sorted()` and `list.sort()` functions.
- Time:
  - `O(n·log(n))`
- Space
  - `O(n)`

TimSort uses binary search insertion

- Instead of using linear search to find the location where an element should be inserted, it uses binary search.



Tim sort hacks

- TimSort scans the array to detect already sorted sequences ("runs").
- Instead of regular insertion sort, it uses binary search to find the insert position.
- TimSort is a **stable algorithm,** preserving the original order of equal elements.
- **In-Place Merge Buffers:**
  - TimSort copies only the smaller run into a temp array, reads the other in place, and merges back into the original array.
- **Galloping Mode:**
  - When merging, if one run starts to dominate (many elements copied in a row), it switches to a fast copy mode.


Tim Sort - 236 comparisons, 723 array accesses, 34 ms delay
http://panthema.net/2013/sound-of-sorting

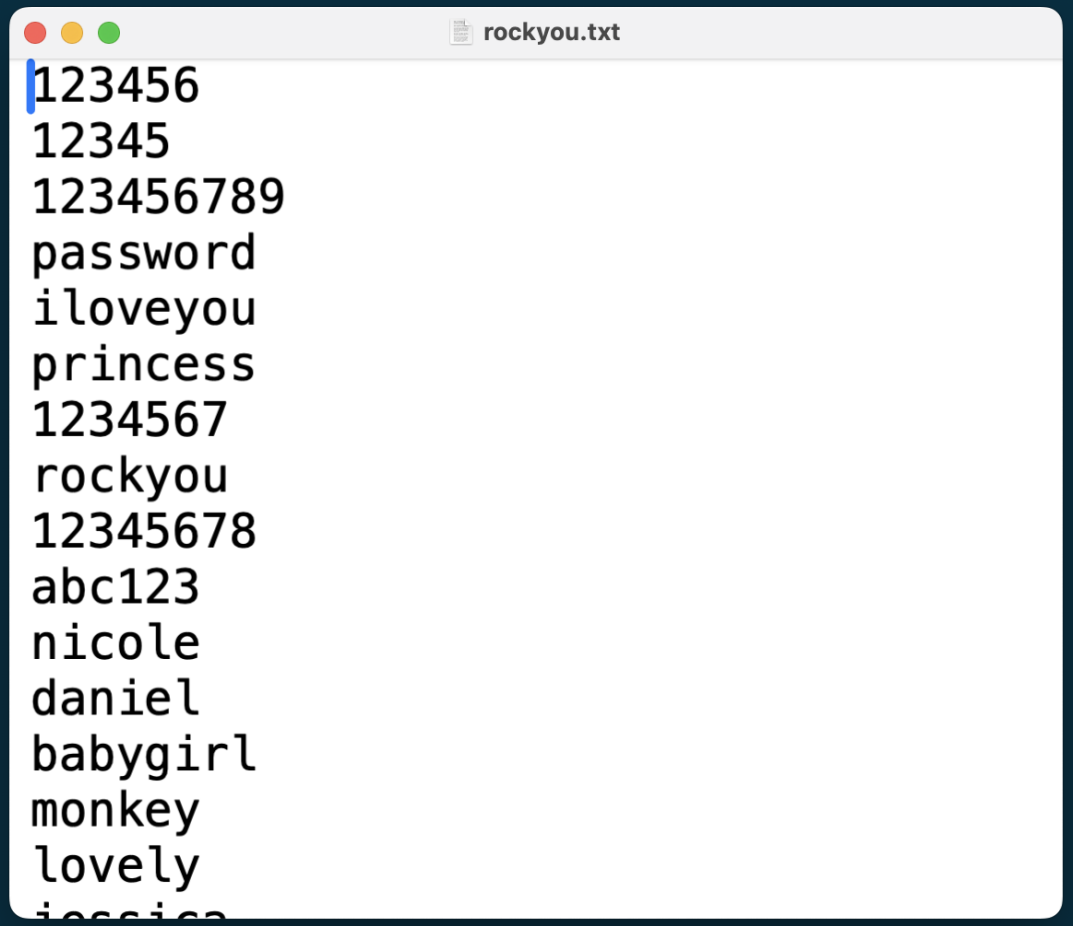
Let's compare them!

Algorithm	Time	Space
Linear search	O(n)	O(1)
Binary search	O(logn)	O(1)
Bubble sort	O(n^2)	O(1)
Insertion sort	O(n^2)	O(1)
Merge sort	O(nlogn)	O(n)
Tim sort	O(nlogn)	O(n)

Working with text files

Pen & Paper

- Write the first 3 lines as Python sees them (raw input as a long String).




```
"123456\n12345\n123456789\n"
```

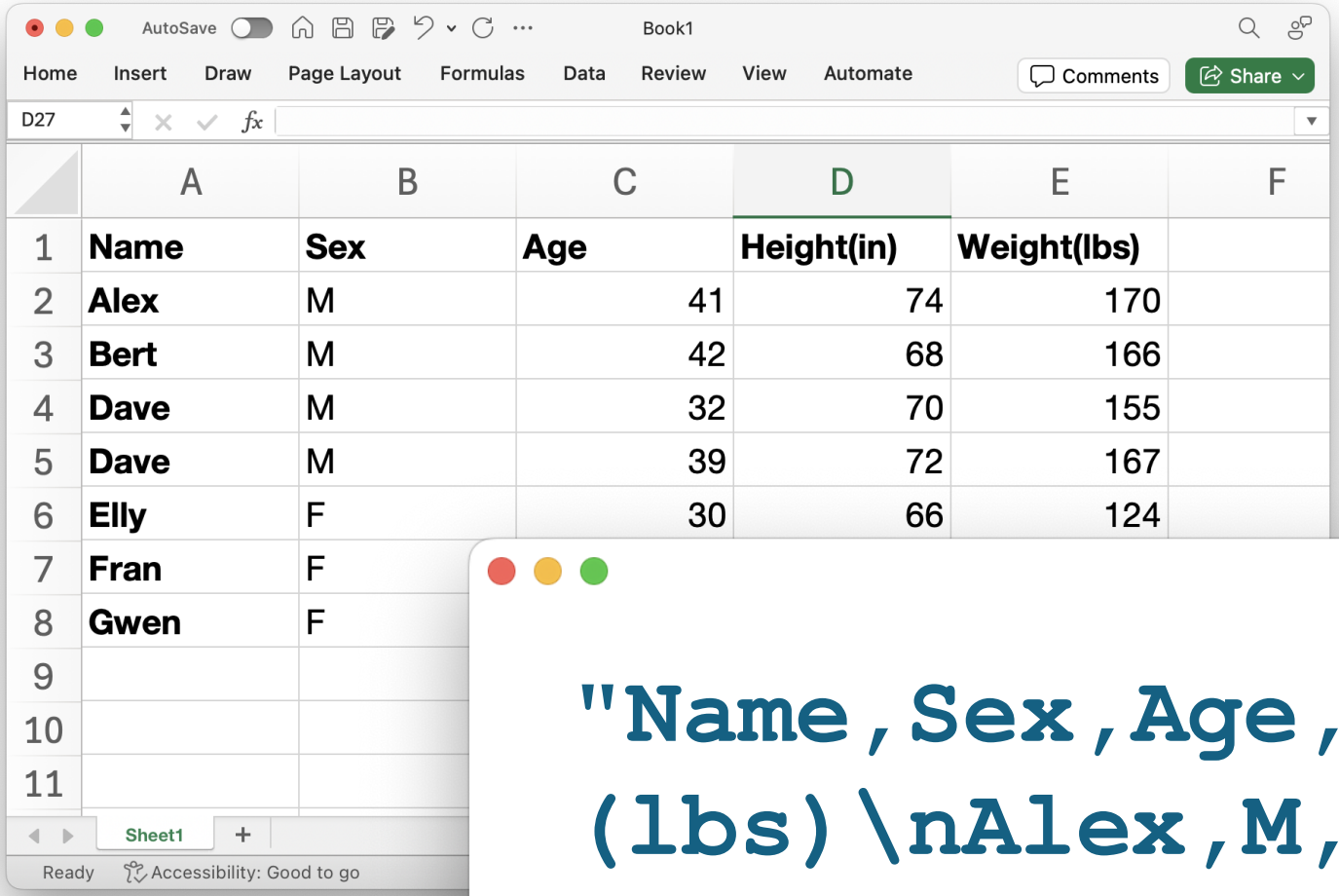
rockyou.txt

```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
nicole
daniel
babygirl
monkey
lovely
```

How Python reads txt data!

script.py.md

```
"123456\n12345\n123456789
\npassword\niloveyou\nprincess\n
1234567\nrockyou\n12345678\nabc1
23\nnicole\ndaniel\nbabygirl\nmo
```



	A	B	C	D	E	F
1	**Name**	**Sex**	**Age**	**Height(in)**	**Weight(lbs)**	
2	**Alex**	M	41	74	170	
3	**Bert**	M	42	68	166	
4	**Dave**	M	32	70	155	
5	**Dave**	M	39	72	167	
6	**Elly**	F	30	66	124	
7	**Fran**	F				
8	**Gwen**	F				
9						
10						
11						

How Python reads csv data!

CSV → Comma Separated Values

script.py.md

```
"Name,Sex,Age,Height(in),Weight(lbs)\nAlex,M,41,74,170\nBert,M,42,68,166\nDave,M,32,70,155\nDave,M,39,72,167\nElly,F,30,66,124\nFran,F,33,66,115\nGwen,F,26,64,121"
```

CSV is just one type of delimited text format

Format	Separator	Example
CSV	, (comma)	name,age,city
TSV	\t (tab)	name\tage\tcity
SSV	; (semicolon)	name;age;city
Pipe-delimited	`	` (pipe)
Colon-separated	: (colon)	name:age:city

script.py.md

```
import csv
with open('data.tsv', newline='') as file:
    reader = csv.reader(file, delimiter='\t')
    ...
```

Data streams

What are examples of data streams?

- Twitter feed (tweets coming in live) - Social Media
- Temperature readings from smart thermostats - Sensor Data (IoT)
- Real-time stock prices or crypto price feeds - Financial Markets
- Live video from a webcam or streaming audio - Video/Audio
- Real-time user activity (clicks, page views) - Web Analytics
- Location updates from delivery vehicles or phones - GPS/Tracking
- Searching in a huge file!
  - File doesn't fit to computer memory!
  - Read it line by line or in chunks

Pen & paper!

- Write a Python script that checks if a leaked password (**`ilovepizza22`**) appears in **`rockyou.txt`**.

script.py.md

```
with open("rockyou.txt", "r", encoding="latin-1") as file:
    for line in file:
        ...
```

Pen & paper!

02:00

- Write a Python script that checks if a leaked password (**`ilovepizza22`**) appears in **`rockyou.txt`**.

script.py.md

```
with open("rockyou.txt", "r", encoding="latin-1") as file:
    for line in file:
        ...
```

```
found = False

with open("rockyou.txt", "r", encoding="latin-1") as file:
    for line in file:
        if line.strip() == "ilovepizza22":
            found = True
            break

print(found)
```

What is a stream of data?

- Data that is delivered one piece at a time, instead of all at once.
- Think of it like:
  - Watching a video on YouTube — the video plays while it's still downloading
  - Processing each line of a large file — without loading the whole file into memory
- Examples
  - Iterators
  - Generators (`yield`)
  - File reading line-by-line
  - Data from sensors, APIs, or network connections

We always need better...

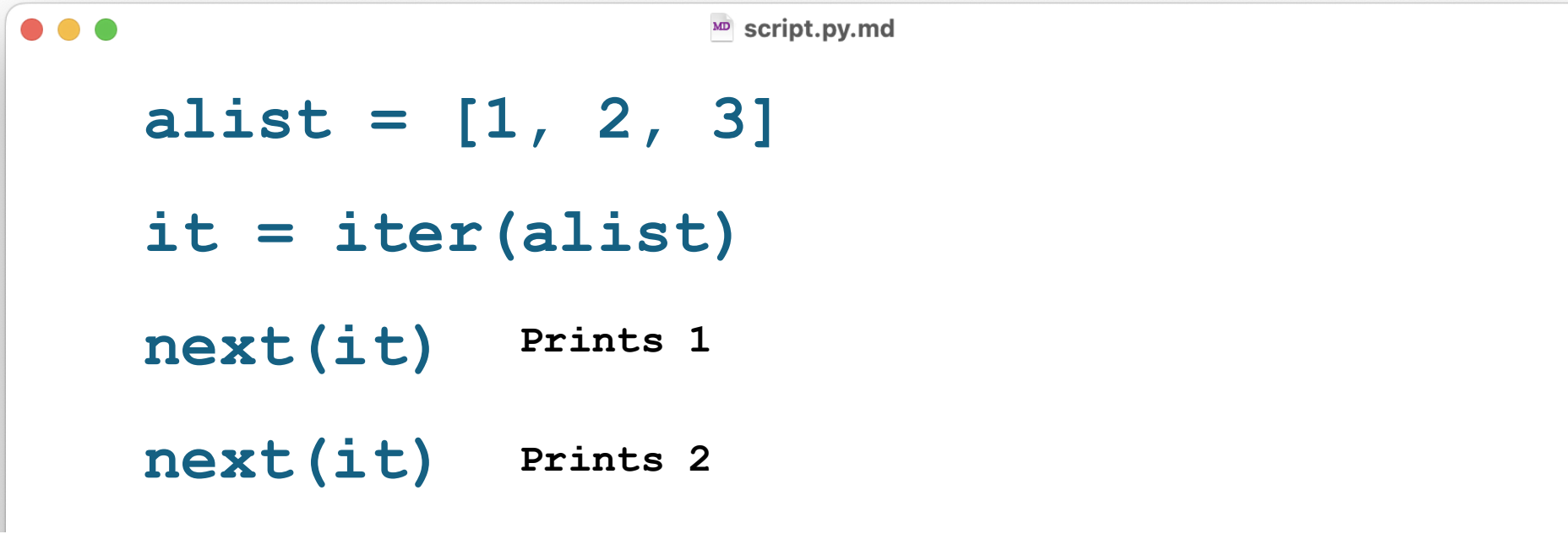
- Continuous goal to reduce the cost of:
  - Processing data → improve time complexity
  - Storing data → reduce space complexity

Working with data

- Reading large files or datasets can use a lot of computer memory
- Functions like `return` store everything in memory
- **Goal**
  - Process one element, at a time, and then ignore
  - Keep memory usage low
  - Pause and resume execution

Iterators: How does it work?

- `alist` is an *iterable* (you can loop over it).
- `iter(alist)` creates an iterator that tracks position as you go.
- `next(it)` retrieves the next item and remembers where it left off.



File example

- Let's say you're reading a large file of customer records.
- You don't load the whole file into memory, just read one record at a time.

script.py.md

```
with open('customers.txt') as f:
    it = iter(f)
    print(next(it))  # prints first customer
    print(next(it))  # prints second customer
```

Time complexity analysis

Symbol	Meaning
`n`	Total number of lines in the file
`m`	Length of a **single line** (in characters or bytes)

- You're not processing all `n` lines, just **2 lines**
- `next(it)` → `O(m)` – Reads one line from the file — time depends on the line length m)
- Total Time Complexity: `O(m)` per line

Space complexity analysis

- Only **one line** is loaded into memory at a time.
- No lists or buffers are built — it's a true **stream**
- Total Space Complexity: **`O(max_m) or O(m) ~average`**
- Where:
  - n = total number of lines (not used here)
  - m = average size of each line
  - max_m = **worst-case time per line**

Python generators

Using generators

- A generator is a special function that uses **`yield`** instead of return

script.py.md

```
def count_up_to(n):
    for i in range(n):
        yield i
```

How does it work?

- Each `yield` sends one value at a time
- Then, the function pauses, saving its state
- And resumes from the same place on the next call

What is the time/space complexity?

```
import csv

def load_all_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        rows = []
        for row in reader:
            rows.append(row)
        return rows
data = load_all_rows('netflix_titles.csv')
print(data[0])  # First row
```

Both O(n·m)

```
import csv

def load_all_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        rows = []
        for row in reader:
            rows.append(row)
        return rows
data = load_all_rows('netflix_titles.csv')
print(data[0])  # First row
```

Memory efficient — ideal for big files or filtering on the fly.

```
import csv

def stream_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            yield row  # yields one row at a time

# Usage
for i, row in enumerate(stream_rows('netflix_titles.csv')):
    print(row)       # One row at a time
    if i == 2: break  # Just show 3 rows
```

`return` VS `yield`

	return (load all rows)	*yield (stream one row at a time)*
What it does	Loads and returns the full list	Yields one row at a time (generator)
Time Complexity	O(n·m) *(read all n rows with m fields)*	O(n·m) *(same total work, row-by-row)*
Space Complexity	O(n·m) *(stores all rows in memory)*	O(m) *(just one row at a time)*
Start-up cost	Waits until all rows are loaded	Starts immediately
Scalability	Poor for large files	Excellent — memory usage stays low
Use case	Small datasets, fast access	Large datasets, streaming, filters

But in the end, is all the data held in memory?

- No — that's the beauty of yield!
  - Yields one value at a time.
  - After it's used, **it's gone from memory**.
  - Even if n = 1,000,000, only one number is ever in memory.

Use `yield` when

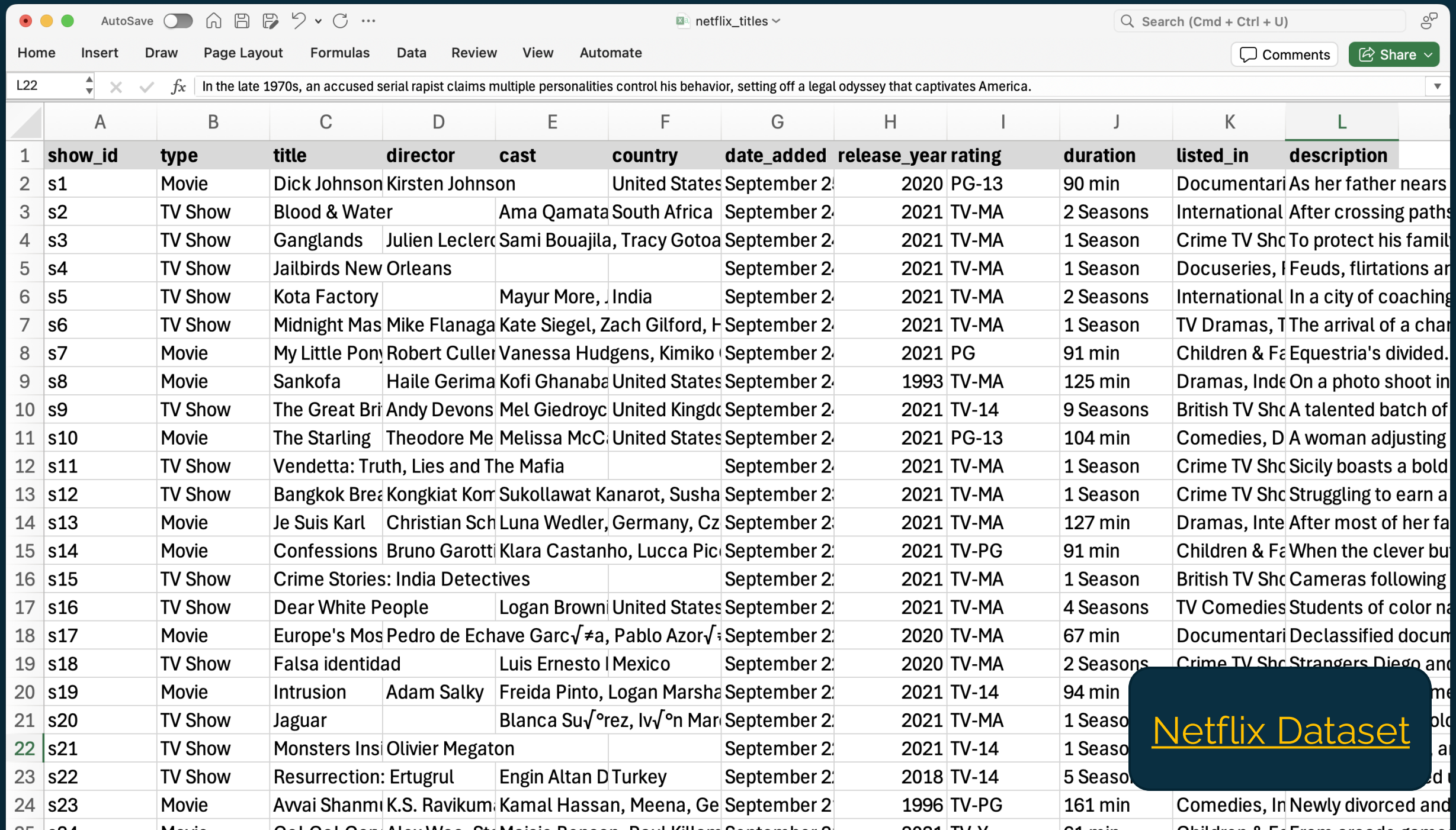
- You're working with large datasets or files
- You want to stream data one item at a time
- You don't need to access data by index
- You only need to iterate once
- You want to start processing data immediately

Avoid `yield` when

- You need to access all data at once
- You need indexing or slicing (e.g. data[0])
- You want to reuse the data multiple times
- You plan to sort or filter the whole dataset
- You'll convert it to a list anyway (list(generator)) — defeats the purpose

`Iterators` VS `Generators`

- A generator is a Pythonic way to implement an **`iterator`** that produces values one at a time, using **`yield`**, making it memory-efficient.
- **Generators are iterators**
  - Use yield to deliver values one at a time
  - Pause and resume execution
  - Are perfect for streaming data, **lazy evaluation**, and large/infinite sequences



	A	B	C	D	E	F	G	H	I	J	K	L
1	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
2	s1	Movie	Dick Johnson	Kirsten Johnson		United States	September 2	2020	PG-13	90 min	Documentari	As her father nears
3	s2	TV Show	Blood & Water		Ama Qamata	South Africa	September 2	2021	TV-MA	2 Seasons	International	After crossing paths
4	s3	TV Show	Ganglands	Julien Leclerc	Sami Bouajila, Tracy Gotoa		September 2	2021	TV-MA	1 Season	Crime TV Sho	To protect his famil
5	s4	TV Show	Jailbirds New	Orleans			September 2	2021	TV-MA	1 Season	Docuseries, I	Feuds, flirtations ar
6	s5	TV Show	Kota Factory		Mayur More, J	India	September 2	2021	TV-MA	2 Seasons	International	In a city of coaching
7	s6	TV Show	Midnight Mas	Mike Flanaga	Kate Siegel, Zach Gilford, H		September 2	2021	TV-MA	1 Season	TV Dramas, T	The arrival of a char
8	s7	Movie	My Little Pony	Robert Culler	Vanessa Hudgens, Kimiko (		September 2	2021	PG	91 min	Children & Fa	Equestria's divided.
9	s8	Movie	Sankofa	Haile Gerima	Kofi Ghanaba	United States	September 2	1993	TV-MA	125 min	Dramas, Inde	On a photo shoot in
10	s9	TV Show	The Great Bri	Andy Devons	Mel Giedroyc	United Kingdo	September 2	2021	TV-14	9 Seasons	British TV Sho	A talented batch of
11	s10	Movie	The Starling	Theodore Me	Melissa McC	United States	September 2	2021	PG-13	104 min	Comedies, D	A woman adjusting
12	s11	TV Show	Vendetta: Truth, Lies and The Mafia				September 2	2021	TV-MA	1 Season	Crime TV Sho	Sicily boasts a bold
13	s12	TV Show	Bangkok Brea	Kongkiat Kom	Sukollawat Kanarot, Susha		September 2	2021	TV-MA	1 Season	Crime TV Sho	Struggling to earn a
14	s13	Movie	Je Suis Karl	Christian Sch	Luna Wedler,	Germany, Cz	September 2	2021	TV-MA	127 min	Dramas, Inte	After most of her fa
15	s14	Movie	Confessions	Bruno Garotti	Klara Castanho, Lucca Pic		September 2	2021	TV-PG	91 min	Children & Fa	When the clever bu
16	s15	TV Show	Crime Stories: India Detectives				September 2	2021	TV-MA	1 Season	British TV Sho	Cameras following
17	s16	TV Show	Dear White People		Logan Browni	United States	September 2	2021	TV-MA	4 Seasons	TV Comedies	Students of color na
18	s17	Movie	Europe's Mos	Pedro de Ech	ave Garc√≠a, Pablo Azor√		September 2	2020	TV-MA	67 min	Documentari	Declassified docum
19	s18	TV Show	Falsa identidad		Luis Ernesto I	Mexico	September 2	2020	TV-MA	2 Seasons	Crime TV Sho	Strangers Diego and
20	s19	Movie	Intrusion	Adam Salky	Freida Pinto, Logan Marsha		September 2	2021	TV-14	94 min		
21	s20	TV Show	Jaguar		Blanca Su√°rez, Iv√°n Mar		September 2	2021	TV-MA	1 Seaso		
22	s21	TV Show	Monsters Insi	Olivier Megaton			September 2	2021	TV-14	1 Seaso		
23	s22	TV Show	Resurrection: Ertugrul		Engin Altan D	Turkey	September 2	2018	TV-14	5 Seaso		
24	s23	Movie	Avvai Shanmu	K.S. Ravikum	Kamal Hassan, Meena, Ge		September 2	1996	TV-PG	161 min	Comedies, In	Newly divorced and

`yield` or `return`?

- What is the first movie rated “PG-13”?
  - yield – streaming example
- Build a report showing average age and total movies per country.
  - return – you need all data for aggregations
- Searching for the first 5 titles with the word “love”.
  - yield – streaming example
- Sort all Netflix titles by release year
  - return - you need all rows in memory to sort)

What is a lazy evaluation?

What is a lazy evaluation?

- One of the core building blocks of big data processing.
- Don't compute a value until **you actually need it**.
  - **Eager**: Do everything now
  - **Lazy:** Wait until asked
- **DVD Download vs Netflix Streaming**
  - Eager evaluation (DVD) = Download the entire movie before watching
  - Lazy evaluation (Netflix) = Start watching right away — stream it scene by scene

Eager (loads everything at once)

- Memory is used for all 1,000,000 results.
- Even if you only need the first value!

script.py.md

```
def double_numbers():
    result = []
    for x in range(1_000_000):
        result.append(x * 2)
    return result
print(double_numbers()[0])
```

Lazy (only creates what's needed)

- Only uses memory for one value at a time
- Much faster to start
- Ideal for large datasets or early exit

script.py.md

```
def double_numbers():
    for x in range(1_000_000):
        yield x * 2
lazy_result = double_numbers()
print(next(lazy_result))
```

Lessons learned

- Lazy evaluation is useful when:
  - You're processing big files, logs, or API responses
  - You don't know how much data you need
  - You want to save memory and delay computation

Thank you!

- The lab starts soon! (404-405)

O(no!)

Lab 3

Big Data Analytics

Lab activities

- Download the data from Kaggle
- Complete lab 3 tutorial and exercises.
- Use your preferred python IDE.