Introduction to Data Science Programming
Live Session

Week 6

Taylor Martin

Section 8

Remind me to start recording! ☺

Berkeley
SCHOOL OF INFORMATION
MIDS • DATA SCIENCE PROGRAM

Today

- Midterm Course Evaluations
- Big Ideas Presentation
- Complexity Activity & Discussion
- Modules & Packages Activity & Discussion

Midterm Course Evaluations

- Log in to course-evaluations.berkeley.edu

Tips for when you know that you know the answer right off and the rest of your group doesn’t

- Be Patient and Allow Others to Participate
- Ask Guiding Questions
  - Qs that can help others figure out the problem

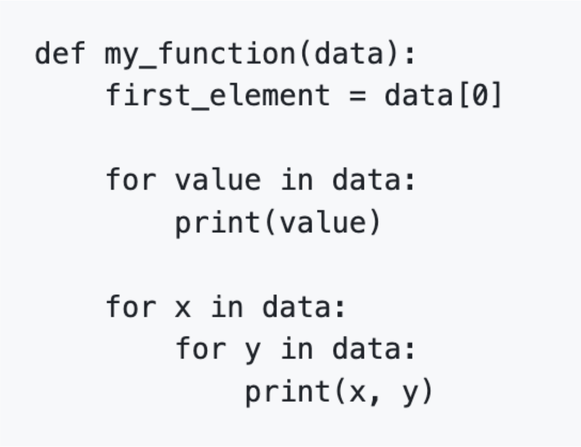
- Respect everyone's contributions, regardless of their level of knowledge.
- Lead the group in breaking down the problem

Complexity

- See the file complexity_activity.ipynb
- Classify code snippets’ complexity
- Practice analyzing the possibility of making code more efficient and rewriting it to reduce complexity if possible.

Complexity Follow up

- The time complexity of an algorithm with several operations is based on the largest complexity among all operations
- The operations in ‘my_function’ don’t make sense
- But it has multiple time complexities: $O(1) + O(n) + O(n^2)$.
- When increasing the size of the input data, the bottleneck of this algorithm will be the operation that takes $O(n^2)$.
- Based on this, we can describe the time complexity of this algorithm as $O(n^2)$.
- Why is the growth rate of an algorithm's execution time more important than a single execution time measurement?



```
def my_function(data):
    first_element = data[0]

    for value in data:
        print(value)

    for x in data:
        for y in data:
            print(x, y)
```

Why on earth do I care? :-)

- Someday, you'll likely need to implement an algorithm for a data operation.
- Understanding time complexity helps you grasp efficiency, identify bottlenecks, and improve your code, especially with large data sets.
  - i.e., helps you:
    - understand why code might run inefficiently
    - write code that runs more efficiently.

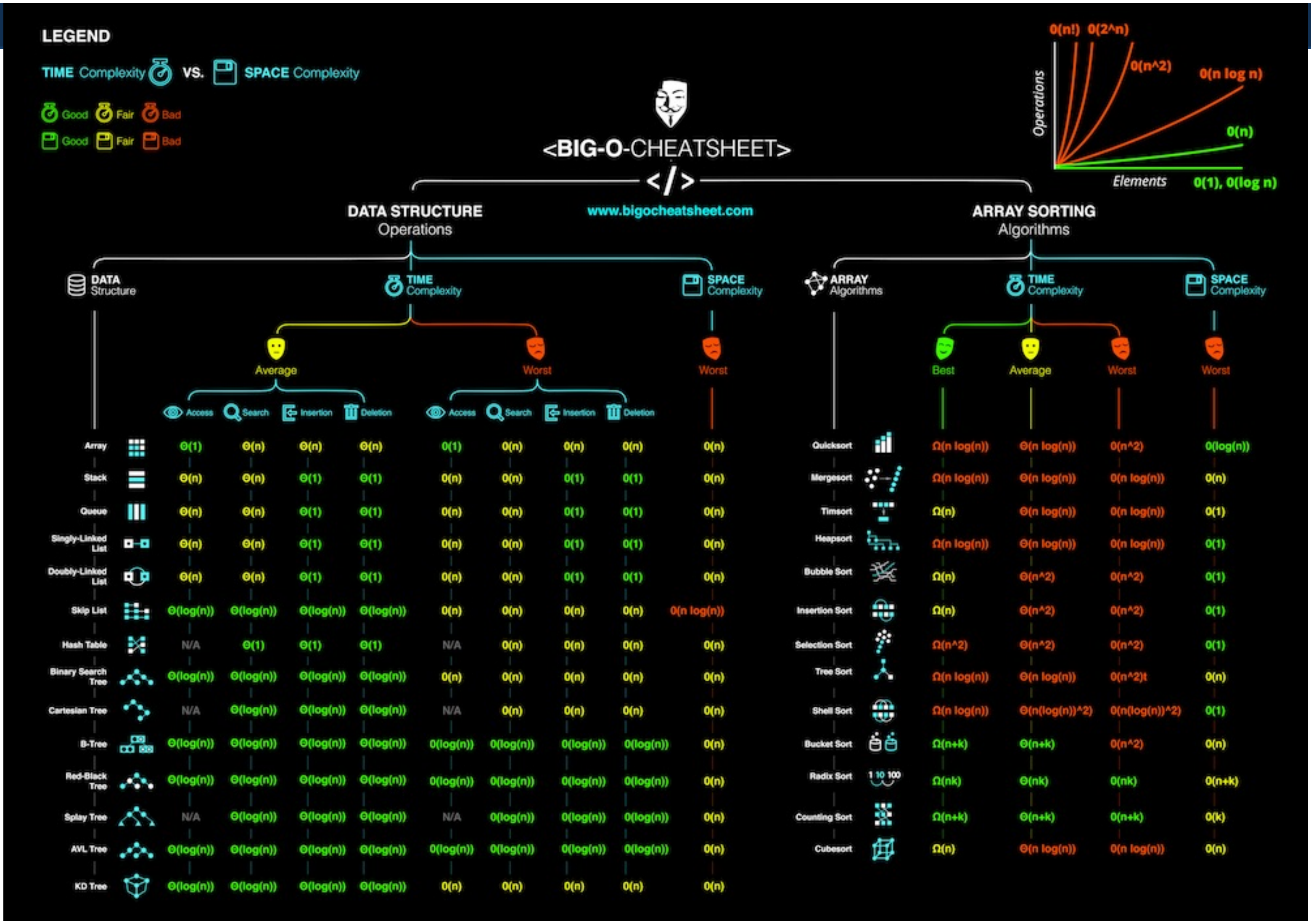
Modules and Packages Breakout

- See the file modules-packages-activity.ipynb
- You’re going to take some existing code, make modules and packages, and make function calls with the appropriate imports.

Modules and Packages Follow up

- A module is just a single .py file file containing Python code.
- A package is just a collection of Python modules organized in directories with a special __init__.py file.
- Packages are just .py files that live in the same folder with a __init__.py file in it.
- This is super helpful to your life.
  - Modularity
  - Reusability
  - The standard python library and third party packages
  - Can create Custom Packages

Have a great week!



Operations
Elements
0(n!)
0(2^n)
0(n^2)
0(n log n)
0(n)
0(1), 0(log n)

LEGEND

TIME Complexity VS. SPACE Complexity

Good Fair Bad

Good Fair Bad

<BIG-O-CHEATSHEET>

</>

www.bigocheatsheet.com

DATA STRUCTURE Operations

DATA Structure	TIME Complexity Average Access	Average Search	Average Insertion	Average Deletion	Worst Access	Worst Search	Worst Insertion	Worst Deletion	SPACE Complexity Worst
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	0(1)	0(n)	0(n)	0(n)	0(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(n)	0(n)	0(n)	0(n)	0(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	0(n)	0(n)	0(n)	0(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(n)	0(n)	0(n)	0(n)	0(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	0(n)	0(n)	0(n)	0(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(log(n))	0(log(n))	0(log(n))	0(log(n))	0(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(log(n))	0(log(n))	0(log(n))	0(log(n))	0(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	0(log(n))	0(log(n))	0(log(n))	0(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(log(n))	0(log(n))	0(log(n))	0(log(n))	0(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	0(n)	0(n)	0(n)	0(n)	0(n)

ARRAY SORTING Algorithms

ARRAY Algorithms	TIME Complexity Best	Average	Worst	SPACE Complexity Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	0(n^2)	0(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	0(n log(n))	0(n)
Timsort	Ω(n)	Θ(n log(n))	0(n log(n))	0(1)
Heapsort	Ω(n log(n))	Θ(n log(n))	0(n log(n))	0(1)
Bubble Sort	Ω(n)	Θ(n^2)	0(n^2)	0(1)
Insertion Sort	Ω(n)	Θ(n^2)	0(n^2)	0(1)
Selection Sort	Ω(n^2)	Θ(n^2)	0(n^2)	0(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	0(n^2)t	0(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	0(n(log(n))^2)	0(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	0(n^2)	0(n)
Radix Sort	Ω(nk)	Θ(nk)	0(nk)	0(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	0(n+k)	0(k)
Cubesort	Ω(n)	Θ(n log(n))	0(n log(n))	0(n)

Berkeley
SCHOOL OF INFORMATION
MIDS • DATA SCIENCE PROGRAM

Project 1 Breakout

- What is your task?
- What is a Second-Price Auction?
- What do you know?
- What do you need to find out?

Project 1 Follow up

- What do we know?
- What do we need to find out?