

This 4-hour course is updated at

<https://github.com/JeremyCoxBMI/PythonDataStructures>

CEAS introduction to python seminar: watch here <http://webcentral.uc.edu/hslclass/home.aspx>

Data Structures for "Non-programmers"

We all write programs, but few of us have experience as a Software Developer and some of us don't have a CS Bachelor's. We're students learning this stuff.

A key piece of feedback from BMI research professors is that many students do not have knowledge of data structures, which is critical to their work. Data Structures is the third class of three in the Undergraduate Computer Science curriculum, which makes it inaccessible to any graduate student who has not already completed it.

Learn everything YOU NEED to know as a "non-programmer" about data structures in two 2-hour sessions!

Come and learn

- a. Understand the fundamental principles of data structures
- b. Apply fundamental principles to your work
- c. Know when to ask for help (and what to say)!

Teacher's Notes Day 1

Agenda. Students ask questions throughout.

- 1) Top 10 list
- 2) Interactive lecture
- 3) Break
- 4) Interactive lecture

Lecture Outline

- A. For next week
 - a. <https://www.python.org/>
 - b. Put it on laptop: Downloads → Get Python 2.7 NOT Python 3.5
- B. Goals for class
 - a. Understand fundamental principles of data structures
 - b. Apply fundamental principles to your work
 - c. Know when to ask for help (and what to say)!
 - d. NOT a goal for you to be algorithms or data structures expert

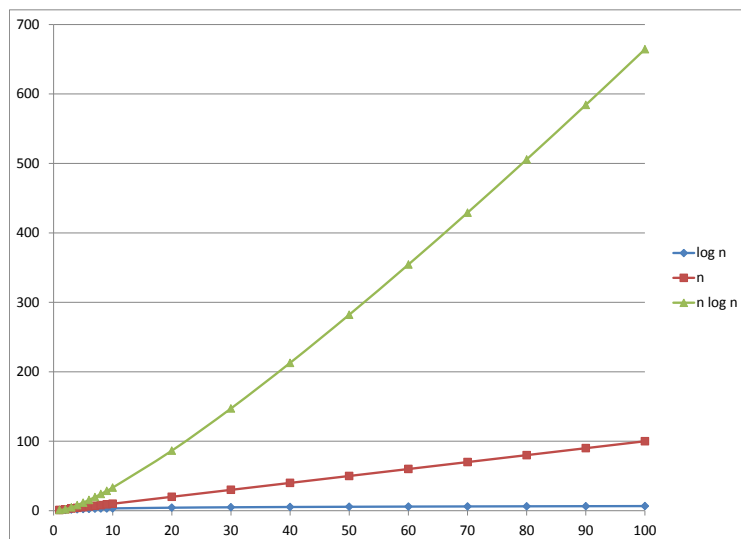
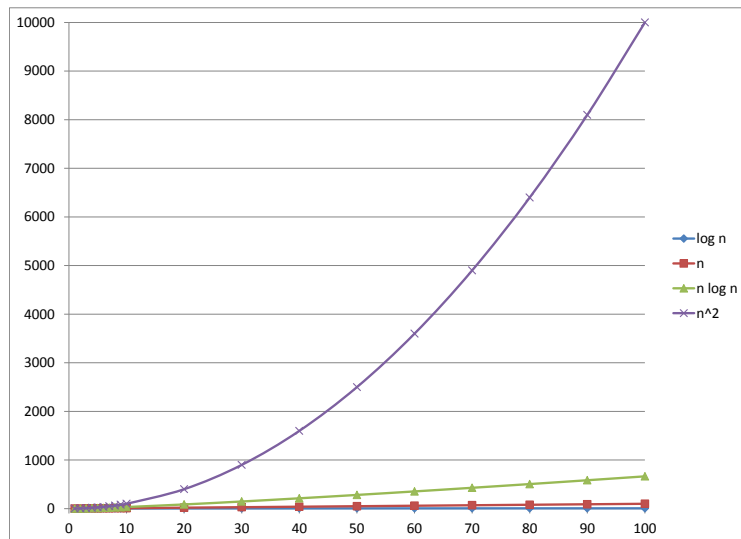
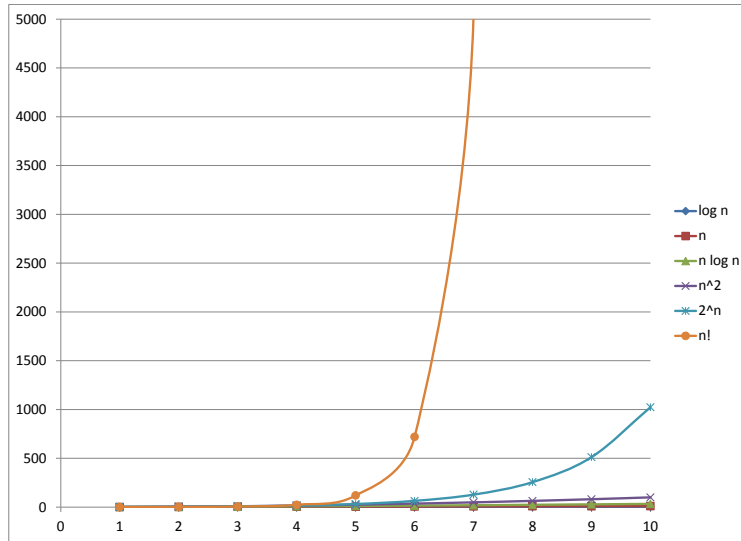
C. Vocabulary

- a. Data Structure is how we store information + associated functions
 - i. Data structures have advantages and disadvantages.
- b. Implementation is a specific approach to represent the data structure.
 - i. Implementations introduce new advantages and disadvantages.
- c. Programming Language specific details can impact implementation
 - i. Languages also have advantages and disadvantages. You may pick a language because it is better to implement something.
- d. Container – simplest data structure
 - i. Can I put zebra cake in swiss cake roll box? What if you are allowed in your kitchen, but not another kitchen? You could cause mom to SEG FAULT.
- e. Model of computing: I am the computer, and the white board is my memory
 - i. ACTIVITY: demonstrate Linked List

D. Fundamentals of Data Structures

- a. Understand the basics, so you know when you are in trouble before you are in big trouble.
 - i. How to know the program works correctly and how long it will take to run.
- b. Four resources
 - i. CPU \$
 - ii. Memory \$\$
 - iii. Disk space \$\$\$
 - iv. Programmer Time. \$
 - 1. (Most valuable: enter the naïve algorithm). \$\$\$
- c. Algorithm
 - i. Strategy to solve a problem and get a correct answer.
 - ii. Heuristic – strategy to solve a problem and probably get the right answer.
 - iii. We use heuristics (i.e. computation cheats) in Bioinformatics, because we don't have a choice based on using big data, algorithms are just too expensive.
 - 1. Example: we want to find a prime number
 - 2. Algorithm: does not exist. (Cryptography depends upon this.)
 - 3. Heuristic: guess approximate value of a prime
 - iv. We know how to solve most problems, but almost all are “incalculable.”
 - 1. Example: chess and Big Blue 1989 (start), 1995 (prototype), 1996 (loss), 1997 (win vs Kasparov)
 - v. Some solutions are better than others. However, usually different solutions are better suited in different situations. Now we have choices.
- d. How do you choose? PRIMARY REASON to study data structures is to know strengths and weaknesses and to pick the right tool for the job.

- i. Your goal for this class is (1) to be able to know what data structure is being used (2) ability to look it up in a “book” and know what it’s weaknesses are (3) when to ask for help from a CS professional.
 - ii. ACTIVITY: Demonstration of sorted list, guess sorted list, list, heap
- e. Introduce $O(n)$ notation
 - i. Relative number of computations. (This is a rough, not precise measure)
 - 1. Simplify: HOW MUCH TIME, keep time spent to a minimum
 - ii. WORST-CASE: what does worst-case mean?
 - 1. example: bubble sort $\rightarrow O(n^2)$ or is it $(n^2)/2$?
 - 2. do you always want to design for the worst case scenario?
 - 3. for bubble sort: worst case is a reverse list -- but why?
 - a. (this insight lead to quick sort)
 - iii. $1 < \log(n) < n < \log(n^2) < \dots < \log(n^c) < n^c < n! < C^n < P(n,r)$
 - 1. C is a constant
 - 2. Going over n^2 is undesirable
 - 3. Geometric or combinatorial – unsustainable
 - iv. Best case and average case exist, although I am not certain there is a universal notation. Sometimes Θ (best) and Ω (average) are used (Wikipedia).
- f. Case study: chunking
 - i. Problem: need to load 90 billion 8 byte objects and sort them
 - ii. $90E9 \log 90E9 = 3.3$ trillion calculation = oh hell no!
 - iii. $90E9 * 8 = 720$ billion byte; cluster max is 256 billion
 - iv. Chunk the data into y chunks of size x | $x*y > n$
 - v. Choose y to be small, perhaps 32. We read a chunk and sort it, then dump to disk. We do this in time: $y * x * \log(x) = n * \log(x) \ll n \log(n)$
 - vi. Then, the data is sorted on disk. So I read one value at a time from the disk, then write the lowest one, then read the file from where that lowest one came from. This is all done in n/y time (n elements, written to priority queue in y time). Because y is small, $ny \cong n \log x$, and the computation time is not greatly increased.



Teacher's Note: End Day 1 assign 0-index

Assign Count What? Worksheet as homework (no time for it in class)

Teacher's Note: Day2 Outline

- 1) Review using PowerPoint
- 2) Interactive Lecture
- 3) Use the programs to apply principles

E. Compare four containers

- a. Do board exercises, where we add things to list, heap, and sorted list, linked list. (Do we want a max-heap or a min-heap?) Teacher does heap.

	Write operation Insert()	Read operation contains()	Space required in memory $O(n)$
List	$O(1)$	$O(n)$	n
Heap	$O(\log n)$	$O(\log n)$	n (footnote ¹)
Sorted List	$O(n)$	$O(\log n)$	n
Linked List	$O(1)$	$O(n)$	n (footnote ²)

- b. Which applications belong to which?³
 - i. Grocery List
 - ii. Priority Queue (Wife's Honey-do list, Tasks at Work)
 - iii. Patients in a data table
- c. Heap is "partially sorted" so you can ONLY READ first value, you don't know where the 15th value is.
- d. Quick Sort – parable of green M&M's one side, red on the other
 - i. Keep mixing will it ever return to all green one side, all red the other
 - ii. NO – entropy
 - iii. Would you sort a reverse sorted list? That's stupid.
 - iv. Because unsorted lists have entropy, they lack certain patterns, thus they are somewhat predictable, and quick sort leverage that to write a better algorithm in average case

F. When to ask for help

- a. How long will my program run?
- b. I need a faster algorithm
- c. I am using too much memory
- d. I can't figure out the algorithm
- e. I can't get it to work right

¹ Has fixed maximum size. Resizing requires a new sort.

² Linked list contains extra data (pointers); thus, more memory is used per element. As a theoretical upper bound, this does not matter. As a practical impact on the program's memory usage, it's a big deal for big data.

³ Heap has limited sort property, always know what maximum is. You cannot access the nth element.

- f. I got a weird compiler/interpreter error
 - i. Computers sometimes make mistakes (Shh! It's a secret.)
- G. How to get help: bring the details
 - a. Don't ask me "will my program finish before Christmas?" without details.
 - i. Get help long before this point
 - b. If I had a nickel for someone asking me this simple question, "Why doesn't my program work?"
 - c. Can you clearly define what are you trying to compute?
 - i. As simple as "converting A -> B"
 - ii. It is very common that this is not clear. Getting this clear is essential.
 - iii. This question drives me crazy. "Can you go try this program on our data?"
 - d. What data structure and algorithm are you using? Why?
 - e. What is your theoretical and observed $O()$ function?
 - f. What is going wrong?
 - i. Bad computation
 - ii. Speed (too slow)
 - iii. Memory (too much)
 - iv. Does it happen only in some circumstances? (does it ever work?)
- H. Fun With Arrays and Pointers
 - a. Fixed (or Static) vs Dynamic memory
 - i. Array vs Linked List
 - ii. Static Memory -- very stable, fixed size
 - iii. Dynamic memory -- slightly unstable, prone to misbehavior
 - 1. Segmentation Fault, SIG SEV, Error 129, Memory Leak
 - iv. Compare Array and Linked List
 - 1. Both implement the idea of a list of numbers.
 - 2. Besides size difference, how do they compare on
 - a. insert()
 - b. get()
 - v. Talk about implementing a Heap as an Array or Tree
 - vi. Garbage Collection
 - 1. C and C++ language allow direct dynamic memory control.
 - a. Good to micromanage if you need to. But if you don't it's a big hassle. Programmer time: \$\$\$\$
 - 2. Other commonly used languages do this automatically.
 - b. Arrays are pointers
 - i. 0-index math and memory
 - ii. Let's resize an array to
 - c. Implement my list from before as
 - i. Array is fixed length

- ii. Linked list takes more memory, $O(n)$ time to access n th member.
 - d. Example: Sorted List can be implemented as an Array, Dynamic Array, or Linked List (choices)
 - e. Example: Heap can be implemented as List. List can be implemented as an array or dynamic array. Dynamic array is implemented as vector/ArrayList (C++/Java, respectively). Or other data structures could be used like Linked List, or ...
 - i. This illustrates layers of choices.
- I. Data structure: Map
- a. Maps can be built in many ways
 - b. Math: Map stores $(x, f(x))$, and lets you lookup $f(x)$ given an x .
 - c. Programming: a Data Structure
 - i. Usually a hash map
- J. Case Study in building the KLUE map
- a. How do you store the map in memory?
 - i. My map is huge, so I don't want to waste memory
 - ii. Problem: one to many relationship
 - iii. Can we save by not writing down keys?
 - iv. Could make a list of pairs
 - 1. key is static
 - 2. value is dynamic, unknown length
 - b. Building a Map in Java
 - i. Map Long to List<Long> (DYNAMIC)
 - 1. HashMap stores separately
 - a. a list of keys
 - b. a hash table to pointers to values
 - 2. Problem is that values change size.
 - a. costs time
 - 3. Problem: memory fragmentation. 50% I lose all benefit.
 - a. It's actually worse, the garbage collector goes apeshit
 - ii. Solution:
 - 1. Store heap as an array of all (key, value) pairs (STATIC)
 - 2. When heap is full,
 - a. Output in order as dump to disk
 - b. Destructive In-place Heapsort
 - 3. Later, combine the databases