DS/Alg (CS3) Day 1

* What is this course about?
  + Draw yin-yang with data structures/algorithms.
  + You know that algorithm = a step by step procedure for accomplishing a task.
  + Data structure = a particular method of organizing and storing data in a computer so it can be accessed/modified quickly.
  + The ways that the access/modification of data structures work are specified as algorithms.
  + One really can’t exist without the other, in that many powerful algorithms are only fast and efficient because they with certain data structures, and if you have a data structure without algorithms for putting things into the structure and getting things out efficiently, then what’s the purpose of the structure anyway?
  + Goals – to introduce the fundamental data structures and algorithms you will see all throughout comp sci.
    - Goal – to have you understand them at the level where you can implement yourself (though you often won't have to)
    - Also to be able to change them to accomplish other, related tasks. (e.g., linked list with some modification, often times the built-in version doesn't do exactly what you want)
    - Know how to pick the right data struct/Alg for the problem!
* Pass out syllabus.  
  + Talk about textbook. Supplemental material in class, on web.
  + Collaboration:
  + Class rules
    - All about respect:
    - Be on time
    - raise hand to ask question
      * corollary – raise your hand a lot!
    - Please raise your hand if you need to leave class in the middle of class.
      * I’ll always let you go, but it’s distracting to have people coming in and out of the room with the door up here at the front, so please ask to be excused.
* Me:
  + Born and raised in Northern VA, suburbs of Washington DC.
  + I went to undergrad Univ of Maryland, grad school Univ of Massachusetts.
  + And then came here to Rhodes 8 years ago.
  + Hobbies/talents/skills – Swing dancing, reading, playing piano, running.
  + Memorable – About five years ago I was in Boston during a sabbatical when the city of Boston broke its all time record for snow. That winter I was there, the city got over nine total feet of snow.
* Get to know students
  + Name
  + Year at Rhodes (first-year, sophomore, etc.)
  + Where you're from/where you're zooming in from
  + Tell us about your pet, or the pet you had when you were growing up, or the pet you wish you had, or if you're not a pet person at all.
* Tuesday will be Java review day.
* (Data) Abstraction and Encapsulation.  
  + To get started, let’s design a class.
  + Suppose you want to design a class that represents integers.
  + We know Java already has a bunch of integer data types, but those data types can only hold a certain range of integers...there’s always a maximum.
  + Suppose we want to design a data type, in the form of a Java class, that can represent integers with any number of digits.
    - Sometimes when we design a class (142) we start with the fields/variables of the class.
    - This time, let’s start with the operations the class should be able to do.
    - Call this BigInteger. What operations should we be able to do with a BigInteger?
      * math ops, negate, get # digits
      * less than, equal to
  + Pretend now that someone gives you the code for this class, but you don’t look at it. You just use an import to bring it in your files. You would know exactly how to use this class even if you’ve never looked at the code, right?
  + Recall that **data abstraction** in computer science is the idea of making choices about what level of detail we want to allow the programmer to see when they are writing programs.
  + We want to emphasize what we think are the important details, and de-emphasize the non-important ones.
    - These can depend on context.
  + The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context.
  + *– John V. Guttag*[[1]](https://en.wikipedia.org/wiki/Abstraction_(software_engineering)" \l "cite_note-1)
  + Now let’s think about the fields of this class. how are we going to represent the number in our class?
    - Array?
    - Vector?
    - linked list?
    - What will they store?
      * individual digits. decimal representation?
      * binary representation?
      * mantissa, exponent representation?
    - How do we store the sign?
    - What order should the digits be in?
      * talk about how reverse order can be beneficial for math operations like adding. (example is 2 numbers of different lengths, like 23+531)
  + Let’s say we pick one of these representations and we write our class. If the people who are using our class can only see the operations (math, from before), WILL THEY BE ABLE TO TELL WHAT REPRESENTATION WE ARE USING?
    - Not if we do our jobs right!
  + This is the idea of **encapsulation**. = restricting access to data that the user/programmer should not be able to see. Also known as **information hiding**.
    - Example: if I release a version of my BigInteger class and everyone loves it. We use decimal representation. Suppose we switch to a binary representation and everyone downloads the new version. Nobody should know we changed, because we don’t expose these details to the user.
  + Interface vs implementation.
  + When we build abstractions, we try to keep the INTERFACE consistent as we refine them and make new versions, and especially as we work with other people who may be building similar abstractions.
    - Suppose I build my BigInteger class, but then a friend builds another version that’s better. If he and I make sure to have the same interface, then we can easily change between versions.
  + The implementation is what’s kept hidden from the user.
* In OOP, we enforce these ideas through making classes, and using public/private to keep things where we want.
* ADTs
  + A big focus of this course will be on studying, designing, and implementing abstract data types, or ADTs. An ADT is always an INTERFACE, in that it tells us what the data type is capable of.
  + It doesn’t usually tell us what the implementation is, though sometimes we can guess.
  + An ADT has 2 parts
    - A short description of what the data type represents.
    - A list of operations that the data type is capable of. The operations do not give low-level details about HOW the operation works, only WHAT it does.
  + Difference between ADT and data structure.
    - ADT = just interface.
    - data structure = interface + implementation. Tells what it does and HOW it does it.
    - Example: List is an ADT. Singly-linked-list is a data structure, doubly-linked-list too.
* So emphasize that part of what we designed for BigInt is an ADT. The parts that are marked as public, the functions, the methods, but not our choices of how the integer is represented.
* Design ADT for Time.
  + What things should a time do? Design INTERFACE.
  + print
  + get hours
  + get minutes
  + get am/pm
  + get 24-hour clock hours
  + calculate difference between times
  + Design IMPLEMENTATION
    - 2 ints + Boolean
    - 2 ints (military time)
    - 1 int (mins since midnight)
* So the title of the course is data structures and algorithms. We’ve covered a lot about data structs so far.
* Where do algorithms come in?  
  + We will focus on not only learning new algorithms, but studying how efficient they are.
  + Efficiency is measured in terms of time and space.
  + You can have any number of algorithms that are all “correct,” but have different efficiencies in terms of time and of space.
* Fibonacci – 3 algorithms
  + Naïve algorithm (v1) recursive O(2^n) time, O(n) space.
  + better algorithm. (v2) O(n) time, O(n) space.
  + even better alg (v3) constant space.
  + How to work through Fibonacci.
  + Present definition of the fib sequence. F(0) = 1. F(1) = 1. F(n) = F(n-1)+F(n-2) for all n>=2.

IN spring 2018 – Did everything but the time class. Designed an ADT for BigInt, but not Time. Did Fibonacci. lasted about the right amount of time (75min)