INFO 6205
Program Structure and Algorithms

Prof. Robin Hillyard

About me

- B.A/M.A Engineering Science (Oxford) (1st Class Hons) 1973
- Ph.D. Computer Science (Cambridge) 1978
- Worked in:
  - Computer-aided design (“Solid Modeling”, “Surface Modeling”) (Pascal, Algol68)
  - Artificial Intelligence/Machine Learning/NLP (Lisp, Java, etc.)
  - Object-relational database design (C)
  - Document Management (C, C++, OPL, Perl, Java)
  - Financial (Java)
  - eCommerce (Java, ColdFusion, Javascript, Groovy)
  - Healthcare
    - Privacy, security, anonymization (Java)
    - Reactive programming (Java, Scala)
    - Big-data analysis with Hadoop/Spark/GraphX/ElasticSearch (Pig, Java, Scala)

Contact Info, etc.

- Email: scalaprof@gmail.com or r.hillyard@northeastern.edu
- Blogs: http://scalaprof.blogspot.com (only Scala articles at present time); http://robin-hillyard.blogspot.com (Java)
- LinkedIn: https://www.linkedin.com/in/robinhillyard
- Twitter: @Phasm1d
- Slack team*: https://info6205-18651.slack.com
- Google Plus: https://plus.google.com/u/0/collection/kqtKXB

* *best way to contact me (team name might not be exactly this)*

More about me

- 1968: wrote my first program
  - solve *sech(x)=x* (in Fortran)
  - it worked first time.
- 1969: wrote my first driver (for a plotter) as well as first use of a “personal computer”
- 1972: wrote my first debugger (for Assembly language).
- 1983: wrote my first object-relational database.
- 1984: wrote my first unit-test runner.
- 1994: wrote my first Java program.
- 2012: wrote my first Scala program.

About you…

- Backgrounds?
- Programming classes?
- Programming jobs?
- HackerRank?
- Java? Java8?
- Functional Programming?

Conduct

- Integrity: Academic integrity is expected from all students.
- Attention: Your attention is *required* at all times during lectures. Laptop computers are, of course, a boon to students. Especially if you need to look up unfamiliar words. But, if you can't resist spending your time on Facebook, Angry birds, whatever, *close the lid!*
- Quizzes: These will normally be conducted using HackerRank at the end of a lecture. Once you're finished, you're free to go if you don't disturb others. Avoid switching out of the HR page and/or copying other's answers.

About INFO6205

- Title: *Program Structure and Algorithms*
- Better Title: *Algorithms and Data Structures*
- Why?
    - Because algorithms and data structures are like Yin and Yang
    - They are dual concepts
    - They are like a marriage

Algorithm

- Al-Khwarizmi, 9th Persian mathematician
- Arithmos (αριθμός), Greek word for number
- Recipe or blueprint?

Wiki-definition

An **algorithm** is an effective method that can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.

Algorithms and…

“ *For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.* ” — Francis Sullivan

“ *An algorithm must be seen to be believed.* ” — Donald Knuth

Francis Sullivan, "The Joy of Algorithms", *Computing in Science & Engineering*, vol. 2, no. , pp. 2, January/February 2000, doi:10.1109/MCSE.2000.10000

… Data Structures

“ I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. ”

— Linus Torvalds (creator of Linux)

“ Algorithms + Data Structures = Programs. ” — Niklaus Wirth

[Aside] Computer Architecture, Languages, etc.

- Procedural language
  - Specifies *how* the computer system is to perform its task.
  - examples: Java, C++
- Declarative language
  - Specifies *what* you want to accomplish and the computer system figures out how to do it.
  - examples: SQL, Prolog

Von Neumann, Turing, etc.

- These architectures are based on the notion of memory locations, registers* and load/store instructions (very low-level stuff);
- They gave rise to the idea of *variables* and *arrays* in programming languages (these are the logical counterparts to physical memory addresses/registers).
  - Actually, you don't always need all those variables: applications written in functional languages like Scala hardly use mutable variables (if at all);
  - However, it turns out that in the development of algorithms and data structures, you can often increase efficiency by judicious use of variables and arrays.

* *Registers were special memory locations supporting load/store and part of the CPU*

Algorithms and data structures

- So, practically speaking, what are algorithms and data structures?
    - A data structure is an orderly collection of memory locations for the purpose of storing information for short or long-term—In the short-term, a data structure is typically used to support an algorithm.
    - An algorithm is a sequence of steps each of which manipulates a data structure to achieve some new state or result.

Why are A&DS important?

- Languages come and go; there are flavors of language which are best-suited to a particular type of application. Some languages are “better” than others: easier to write; easier to read; produce more efficient code; compile/interpret faster; higher/lower “level”; more or less verbose; etc. etc.

- But algorithms and data structures are **fundamental** to solving problems—for this reason, they haven’t changed very much in the last forty or fifty years.

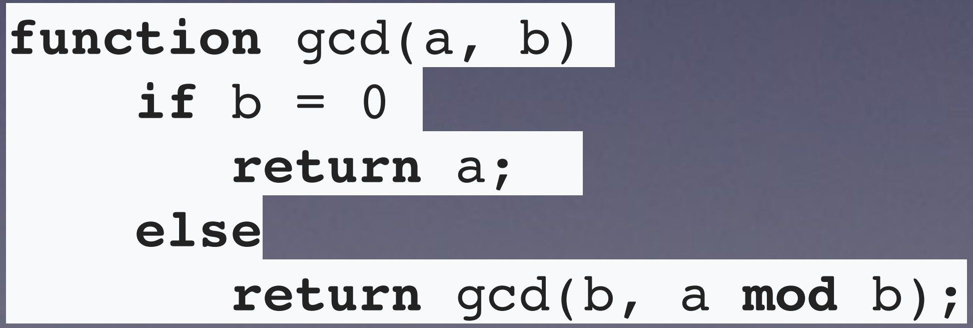
- There are two precious commodities in any computer system: cycles (i.e. time) and memory (i.e. space). Good algorithms and data structures aim to minimize the use of these commodities.

So, which language will we use?

- Java. Why?
    - As alluded to in the previous slide, languages are *orthogonal* to algorithms and data structures.
    - We could use *any* of the mainstream languages and many more besides. Donald Knuth, who wrote the first major work on computer algorithms (two heavy volumes), actually invented his own language (MIX) just so that the language issues wouldn’t get in the way.
    - But, the textbook is written in Java, and Java is the world’s #1 programming language so it makes sense to use it here.
    - *However*, I am a big fan of functional languages so I might slip in a little Scala here and there, where I think there is probative value eliminating mutable variables from an algorithm.

Historical aside

- Many ancient algorithms:
  - Sieve of Eratosthenes (200BC) for finding prime numbers;
  - Euclid’s algorithm (300BC) for greatest common divisor (see Graphical animation of Euclid's Algorithm):



```
function gcd(a, b)
    if b = 0
       return a;
    else
       return gcd(b, a mod b);
```

Why are A&DS important? (Continued)

- Their impact is broad and far-reaching:
  - Internet. Web search, packet routing, distributed file sharing, …
  - Biology. Human genome project, protein folding, ...
  - Computers. Circuit layout, file system, compilers, databases, …
  - Computer graphics. Movies, video games, virtual reality, ...
  - Security. Cell phones, e-commerce, voting machines, ...
  - Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...
  - Social networks. Recommendations, news feeds, advertisements, graph databases, …
  - Physics. N-body simulation, particle collision simulation, ...
  - Big Data, Meteorology and Engineering: Finite-element analysis, map-reduce

Why are A&DS important? (Continued)

- I used to have a colleague who was fond of saying: “There’s nothing like hardware to improve software.”
  - This was during the 80s when hardware was undergoing rapid improvements in both performance and, especially, price.
  - Unfortunately, it’s a short-sighted view because, while the cost of cores (CPUs) and memory have drastically reduced over recent decades, the clock speeds of processors has not.
- And, even if it clock speeds were still improving, improvement will never be much better than linear and, in the next slides, we’ll see that that is no match for real life!

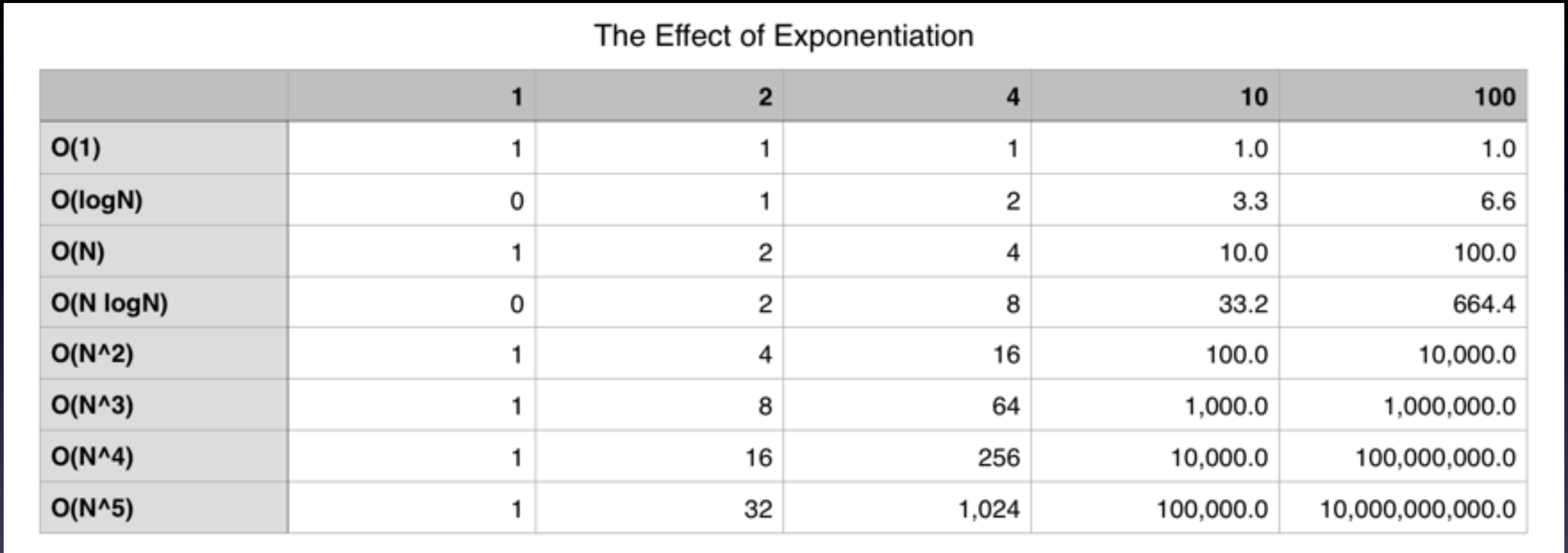
The Exponential Curse

- A nice little (but unsophisticated) database story:
    - Imagine that you create a database of your favorite songs (it doesn’t really matter how you do this). You have 1000 songs registered and when you search for one it always takes about 1 second. The space you’re using is about 10 Mbytes which is fine: you have plenty of space available.
    - Now, you get a girlfriend/boyfriend and they want to add their 1000 songs to your database. That’s fine. But now, when you search for a song, it doesn’t take 2 seconds like you’d expect. It takes 4 seconds. Hmmm! You also notice that you’re using about 40 Mbytes of space now.
    - Your entire class of 100 students all want to store their songs and use your database. But to search for a song, it now takes, 10,000 seconds (2.75 hours!!!) and you’ve used up all your memory because it takes 100 GBytes to store everything. Aargh!

$O(N^2)$

- What we just experienced (in this highly fictitious story) was an “N^2” problem, both in time and space. We write this as $O(N^2)$, pronounced “Order—N-squared.”
- Obviously, that’s completely unacceptable! Now let’s look at the power of exponentiation.

Exponentiation



The Effect of Exponentiation

	1	2	4	10	100
O(1)	1	1	1	1.0	1.0
O(logN)	0	1	2	3.3	6.6
O(N)	1	2	4	10.0	100.0
O(N logN)	0	2	8	33.2	664.4
O(N^2)	1	4	16	100.0	10,000.0
O(N^3)	1	8	64	1,000.0	1,000,000.0
O(N^4)	1	16	256	10,000.0	100,000,000.0
O(N^5)	1	32	1,024	100,000.0	10,000,000,000.0

- But who would ever create an algorithm that was O(N^5)?
- I did!
  - It was a *very* long time ago;
  - I was part of a team and possibly not the most guilty;
  - BTW, 10,000,000,000 seconds is 320 years!!

Why are A&DS important? Continued

- Execution:
  - As we just saw, the efficiency of an algorithm can make a huge difference to running time (or space)—indeed it can easily make the difference between running the application and not;
- They are interesting, elegant, and very worthy of our study *(as we saw with Francis Sullivan quote);*
- And a more prosaic reason:
  - many of the companies for which you will apply to work will ask you to implement algorithms either on the white board or using something like *HackerRank.*

Non-deterministic algorithms

- Earlier, I left out an important sentence from the Wikipedia article. In fact, many algorithms introduce some randomness into their execution:
    - Either due to timing (for concurrent algorithms);
    - Or due to the use of random values (either from measuring devices or, deliberately, employing PRNGs);
- Examples of the latter include so-called Genetic Algorithms—a favorite topic of mine.

My very first algorithm

```
public class NewtonApproximation {
    public static void main(String[] args) {
        // Newton's Approximation to solve cos(x) = x
        double x = 1.0;
        int tries = 200;
        for (; tries > 0; tries--) {
            final double y = Math.cos(x)-x;
            if (Math.abs(y)<1E-7) {
                System.out.println("The solution to cos(x)=x is: "+x);
                System.exit(0);
            }
            x = x + y/(Math.sin(x)+1);
        }
    }
}
```

- Newton-Raphson method (https://en.wikipedia.org/wiki/Newton%27s_method) to approximate the solution to cos(x) = x
- Originally written in FORTRAN in 1968 by manually punching holes (“chads”) in a partially perforated “Hollerith” card. Turnaround time: 1 week!
- In my Scala/BigData class, I use this as an exemplar of “imperative-style” programming (and then I show how to improve it with functional programming). But as a Java algorithm, it works fine.

What will be covering?

topic	Data Structures and Algorithms
intro & data abstraction	APIs, ADTs, stack, queue, bag, union-find, O(N)
sorting	quicksort, merge sort, heap sort (priority queues)
searching	symbol tables (maps), binary search trees, hash tables
graphs	undirected and directed graphs, minimum spanning trees, shortest path
advanced topics	strings, tries, substring search, regex, Btrees

Basic Programming Model

- See Chapter 1.1 of Textbook
  - Ensure that you understand about primitives in Java (their range of possible values) and their corresponding (boxed) Objects
  - I want you to start thinking about the difference between mutable and immutable variables and data structures:
    - Always initialize variables when you declare them (to do otherwise is a serious code smell);
    - If you don't plan on allowing them to mutate, mark them as *final*.
  - Arrays are, by definition, mutable and they will be used a lot in this class.

Collections

- It’s not very interesting, or useful, to deal with one thing at a time. In most code (not all, obviously), you will be dealing with *collections* of things. One of the most important use cases for collections is to serve as longer-term data structures. [This is as opposed to shorter-term data structures which are there to support algorithms, such as sorting].

- Collections are built from abstract data types and the specific requirements for creation and access are what determine the particular implementation.

Lists, arrays, symbol tables

- What are the *essential* differences between the following:
  - a list
  - an array
  - a symbol table (aka hash table or map)

List, arrays, hash tables

	Size	Access	Construction
Array	Fixed	Random, by index	Random, by index
List	Variable	Sequential, starting from the head	Sequential, starting from the head
Hash Table	Growable	By key	By key and its hash

Quiz 1

- Topic:
  - This is based on the Newton-Raphson method mentioned earlier. This kind of algorithm is very different from most of what you'll be learning in this class. But it doesn't hurt to get a feel for Numerical computing.
- Get a feel for the algorithm:
  - Watch the simulation from Wikipedia (URL: https://en.wikipedia.org/wiki/Newton%27s_method#/media/File:NewtonIteration_Ani.gif)

This week’s quiz

- Some slight improvements to the Java style here:

```
import java.util.function.DoubleFunction;

public class Newton {
    public static void main(String[] args) {
        // Newton's Approximation to solve cos(x) = x
        newton("cos(x) - x = 0", 1.0, 200, (double x ) -> Math.cos(x) - x , (double x ) -> - Math.sin(x) - 
1 );
    }

    private static void newton(final String equation, final double x0, final int maxTries, final 
DoubleFunction<Double> f, final DoubleFunction<Double> dfbydx) {
        double x = x0;
        int tries = maxTries;
        for (; tries > 0; tries--) {
            final double y = f.apply(x);
            if (Math.abs(y) < 1E-7) {
                System.out.println("The solution to "+equation+" is: " + x);
                System.exit(0);
            }
            x = x - y / dfbydx.apply(x);
        }
    }
}
```

- Quiz 1 is available on Blackboard. It shouldn't take you long to finish it now.