A. Go over the syllabus and what this course is about

Data Structures
Algorithms (searching and sorting and many others)
Recursion
Object Oriented Programming
Analysis
Python language

B. What is Computer Science and how it differs from Computer Programming

analysis of algorithms – involves numerical and mathematical judgments
design – create structures for future use, kind of like "best practices in bridge building"
planning and communication – how to improve the field by inventing new ideas

C. Some general ideas

- Moving beyond mere programming to more sophisticated problem solving
- comparison between approaches is what is meant by analysis
- more mathematically oriented (but still easier than Calculus!)
- hacking out solutions by pure coding is not what this is about

D. Abstraction

functions
classes
ADTs
the dashboard of your car (or airplane)

E. Algorithms

The heart of CS, the main idea!
Definition:

*A step-by-step description of a method to solve an instance of a problem in a finite amount of time using well-defined basic operations.*

Alan Turing 1936

3 levels:
1. unsolvable – provably impossible, undecidable, uncomputable
2. intractable – computable, decidable, but would take too long for practicality — 1974
3. tractable – computable in a reasonable amount of time

What is reasonable? The lifetime of the universe or longer is unreasonable.
Google years $10^{100}$ years, all black holes will evaporate. If your program is still running without an answer then, then it is impractical!

Decidable vrs. undecidable
Yes / No answer -- decidable will definitely give you an answer of one of these two, but could take a long time.

Undecidable – You’ll get a YES answer if that is the case, but never a NO answer. How long do you wait?

The Halting problem – classic undecidable problem

HP(P,X) = "Give me a program P and input X, and I will tell you if P will ever halt on X."

One way to do this is to simulate P running X, i.e. actually run P on X. But if it would not halt on this input, then your decision program would not halt and you’ll never get a NO answer!!!!!!

There is provably no way to write HP. Alan Turing proved it!

History: Alan Turing

E. Comparison of algorithms

What is a problem? – description of a task to be done and the form of its input and output
What is a problem instance? – specific values of input

Addition is a problem: "Add 2 integers together to get a sum"

Instance: "Add 57 and 94"

Network shortest path traversal is a problem:
"Given a network of nodes and paths between them, and a desired starting point A an a desired ending point Z, find a path (sequence of nodes and links between nodes) that go from A to Z in the shortest amount of time."

Instance: Given network N1, start at Buffalo and go to Denver

Algorithm – the general abstract process of getting a solution to a problem instance
Program – an algorithm that has been encoded in a programming language to run on a specific computer

Algorithms are abstract, but programs are definite and real. An algo. can be written in general terms, in English (or other NL). Freedom of thought in USA means you cannot copyright an algorithm, but you can copyright a program! A program is a definite expression, a sequence of symbols, letters, etc. that cannot be plagiarized.

The algorithm is mightier than the chip.
Two algorithms that solve the same problem but they differ:

1. One takes a lot longer than the other
2. One uses a lot more memory during the problem solution than the other
3. One requires more cores (processing units) than the other
4. One is much harder for people to understand the other

These are shortened:

1. TIME
2. SPACE
3. IRON
4. COMPLEXITY (human complexity)

Sometimes easy-to-understand programs run abysmally slow.

There seems to be in general a trade-off between TIME and SPACE. If an algorithm is very fast it often requires vast amounts of extra working memory. Or if a program is very parsimonious in terms of memory, it often runs slow.

However, there are violations of this! There are some algos that are both very very fast and parsimonious (=cheap, stingy, tight). So we don't know.

Current debate in CS centers around TIME v. SPACE, called the "P=NP? Problem."

The trouble is that there might be an as-yet undiscovered algorithm that is both fast and memory-cheap. No one has proved that there isn't one!

F. Data Structures

Scalars – simple unitary data items:

- booleans, bits, chars, ints, real numbers, complex numbers, fractions, grid coordinates

Vectors – collections

| array (1-dim, 2-dim, 3-dim, etc) | list |
| stack | queue |
| dequeue | dictionary (map) |
| tuple | tree |
| graph | |
| set | bag (multiset) |
| poset (partially ordered set, aka "dag") | ordered set |
Implementation issues:

a list is a generic collection, but an array is a low-level way of making a list, at least in machine language

we can think of an array as a numbered list, because list items have an order but are not necessarily ordered with integers. Could be ordered with real numbers, fractions, etc!

How are these things mapped into the von-Neumann architecture?

VN machine says that memory consists of a 1-dimensional array of memory words, each of N bits, and each numbered from 0 on up. It is a simple numbered list of memory words. In most modern computers, a memory word is a byte (8-bit chunk, the smallest addressable unit of memory.)

So how would you map a graph or a tree into a 1-dim array? Or even a 2-dim array? Or a set? This course tells you how! And it tells you good and (not-so-good) ways of doing this, ways that would cost you either extra time or extra memory.