Day 0

This is CS50 for MBAs. Harvard Busines School. Spring 2015.

Cheng Gong

Table of Contents

Introduction

- CS50 for MBAs[1] is an introductory course to computer science, tailored for those who might be interested in becoming project managers in technology, those who want to be able to have technical conversations with engineers or others, or even those who just want to better understand the technology we have today.

Data Storage

- On the way here, the hallway next to the classroom actually has source code framed on the wall, along with floppy disks. The ones on the wall come with warnings that they have viruses, software that does bad stuff, and that they should not be put into anything.
- We got everyone a floppy disk from eBay (they're really hard to find these days), so we can physically see some technology, even if today's version of a floppy disk has evolved to be faster and smaller.

[1] https://cs50.harvard.edu/mba/

- Sliding back the metal cover, we see that the actual "floppy" disk is inside a hard plastic shell.
- Data is stored on the disk, which means that there are magnetic particles on the circular piece of material used to represent information. The polarity of each of these particles can be used to represent values. For example, a particle pointing north could be interpreted as a 0, and a particle pointing south might be a 1.
- Let's snap the plastic shell in half, and take out the actual disk. Back in the day, something like that probably stored 1.44 megabytes, or 1.44 million bytes.

  # A **byte** is 8 bits, and a **bit** is a binary digit (0 or 1).

- Some hardware today, such as hard drives, are functionally equivalent to these floppy disks, with the exception that they're physically hard and thus more durable.
- Let's take a closer look with this video[2]. Electrical signals from a computer's memory is translated into voltage fluctuations, which control the spinning disks in a hard drive, where other signals control a read/write head that actually changes the data on a disk.
- Hard drives have more capacity than a floppy disk not only because they are harder and can fit things more densely, but because they are made up of multiple disks layered together.
- The second part of the video[3] visually shows how the read/write head has a magnet, which is controlled by electrical signals, that change the polarity of magnetic particles on a disk. Lots of these particles together, each of which represent a single bit, can then be used to make up a file.
- Realize that this might all be unfamiliar, so there are no dumb questions or dumb answers in this class. In fact, we want to practice reasoning through problems to solve them, even if we have limited information, as that's what people once did to solve these problems the first time around.

Topics

- **Computational thinking** is where we'll start this first week, and that just means problem solving, much like a computer would, where we are methodical and precise with our solutions.

[2] https://www.youtube.com/watch?v=qs2mtaky1rI
[3] https://www.youtube.com/watch?v=j4aMAO0c9kA

- Next week, we'll learn about technologies that make the **Internet** possible, like routers, switches, and NAT. We'll get an overview of them, so we can use them to help us solve higher-level problems in later weeks.
- Then we'll talk about **back-end development**, which mainly deals with software running on a server, databases, and the like. This is in contrast to **front-end development**, that we'll cover later, which deals with what the user actually sees. That too, can have some sophisticated code underneath.
- Some of you might have prior experience with SQL or Microsoft Access, but we'll look more closely at **database design**, and what we can do with databases.
    # In fact, the video for Gangnam Style[4] had a huge number of views on YouTube, a number so high that the counter broke. Some years ago, YouTube set aside some amount of memory, 32 bits, for each video, to keep track of its number of views. Once the limit of what those bits could count up to was reached, the number rolled over back to 0, or possibly became negative (as we'll see later).
    # So database designers think about these cases and tradeoffs between characteristics like the amount of memory used or performance. Y2K is another example. In the 1970s, the designers of the systems used to keep track of the date decided to use only two digits to keep track of the year, thinking the software would certainly be updated by the year 2000. This turned out to not be the case, causing people to go back and fix the problem.
- **Cloud computing** is a term that just refers to the practice of having software that runs on someone else's servers, somewhere else, rather than on your own. This has been around for a long time, but recently the advancements in software and hardware have been making CPUs that are really fast, and overkill for what we usually do. So it's started to make business sense for someone to buy fancy hardware with lots of CPUs and sell virtual partitions of that computing power to many customers. In fact, servers can be oversold, since it's unlikely that all the users will be using their allotment at the same time.
- We'll look at **mobile development** too, whether in the form of apps specifically for Android or iOs, or websites adapted for mobile devices.

[4] https://www.youtube.com/watch?v=9bZkp7q19f0

- Finally, we'll look at **technology stacks**, another fancy term for what we use in our toolkit when creating a database, back-end, or front-end, in the quest to create a complete product.

Syllabus

- The **syllabus**[5] will break these topics down day-by-day, but by day 13 (**0-indexed** because we started counting at 0, as computer scientists do), you'll be empowered to understand these new topics and have a sense of what to look for, should you encounter unfamiliar ones.
- **Assignments** are meant to be done the afternoon or night of a class, to recap things, push you a little beyond, and prepare you for the next class.
- You, with 0, 1, or 2 classmates, will use some of these technologies to build something with programming outside your comfort level.

  # This week we'll start by way of a graphical programming environment called Scratch.
- No matter if you're "less comfortable" or "more comfortable," we'll try to tailor this course and its requirements to get the highest marginal return for everyone.
- In fact, people have said that CS50 at the College is about three times the work of a typical MBA class. We'll try to keep it to 1x in this class, but give you some space if you want to go beyond, too.
- We'll show you how to use CSS and HTML to make a webpage, Python and SQL to build a back-end, JavaScript for the front-end, and tentatively we'll look at Android and iOS so you have an understanding of those platforms as well.
- We have teaching staff for this course, so we'll try to find an optimal time for **office hours**.
- The website has a link to **Ask**[6], which is just a Google Form that will anonymously send a question to the staff, as we'll try to include them into the next class' discussion.
- To talk to staff and students more interactively, you can sign into **Discuss**[7], which is basically a discussion board where you can post questions and answers.
- As per the syllabus, we expect you to attend all classes, complete all assignments, and implement all projects.

[5] https://cs50.harvard.edu/mba/syllabus
[6] https://cs50.harvard.edu/mba/ask/
[7] https://cs50.harvard.edu/mba/discuss/

- **Grades** will be broken down per the syllabus, with 25% participation, 35% assignments, and 40% projects.
- For participation, simply asking questions and attempting to answer questions, even if incorrect, will be the main goal.
- Finally, Susan Wojcicki[8] of YouTube speaks of her experience with CS50 and how it "changed how [she] think[s] about everything."
- We'll bring in more examples of actual friends in industry, such as Act.md, Quora, and Dropbox, and see how they implement their technological magic.

Computation

- Let's (over)simplify **computation** to just three components: **inputs**, **algorithms**, and **outputs**. In the case of Google, for example, their inputs are all the webpages in the world, their algorithm is their method of searching through all the pages, and their output is the list of results when you search for things. But an **algorithm** is just a set of steps for solving some problem.
- Computers need some way of representing information. Back in the day, the most obvious way was to use electricity, and either turn something on, or off. Using those two states, we can actually represent all the information we want, with patterns of on and off.

Binary

- Computers still use 0s and 1s, or **binary digits** (**bits**). Humans normally use 10 digits, 0-9, to count. It turns out that decimal is quite similar to binary, or any other mode of counting.
- Let's take a look at `123`, which we would normally read as "a hundred and twenty-three." Well, each number is in a column, or a place, with the 1s place on the right, the 10s place in the middle, and the 100s place on the left. So we have `1` times 100, `2` times 10, and `3` times 1, for a hundred and twenty-three altogether.
- We can extend that to binary, where the columns will represent, from right to left, 1s, 2s, and 4s. So the places are powers of 2, where the rightmost column is 2^0, the middle is

[8] http://www.youtube.com/watch?v=y1121-De4o4

2^1, and the right is 2^2. In the previous example with decimal, the 1s place was 10^0, the 10s place was 10^1, and the 100s place was 10^2.

- So we can represent the number 1 as `001` in binary, 2 as `010` in binary, 3 as `011`, and 4 as `100`. Notice that we might need to change more than one digit sometimes, as we do when counting from, say, 99 to 100 in decimal.
- To get 8, we'd need another digit, since with three bits `111` is equivalent to 7. Adding 1 to that, we'd get `1000`, which has four bits.
- Remember that YouTube was using a 32-bit number for their counter, which meant that they had 32 places, and eventually when all those bits counted up to 1s, all of them were flipped to 0s, as we would when going from 7 to 8, but their software only had 32 bits, so when that happened it looked like the count was actually just 0. (We're simplifying here a bit, because the leftmost bit is usually used to keep track of the sign of the number, whether it's positive or negative, so there might have actually been only 31 bits. And once the 32nd bit was incremented to a 1, the number as a whole was interpreted to be negative. Even that's a simplification, as there's an entire system for representing negative numbers called Two's complement[9].
- Let's demonstrate this with a row of lightbulbs, labelling them like so:

```
 ●   ●   ●   ●   ●   ●   ●   ●
128  64  32  16  8   4   2   1
```

- Each of these circles are lightbulbs, which in turn represent bits. There are 8 total, and the total we can count up to is now 255.

  # We can't count up to 256, because one of the values we need to represent is 0. But there are still 256 total distinct values.

- To get the number 15, we'd turn on these lightbulbs:

```
 ●   ●   ●   ●   ○   ○   ○   ○
128  64  32  16  8   4   2   1
```

- To get the number 50, we'd do this:

[9] http://en.wikipedia.org/wiki/Two%27s_complement

```
● ● ○ ○ ● ● ○ ●
128 64 32 16 8 4 2 1
```

APIs

- The lightbulbs are actually called Philips Hue[10] bulbs. They can change colors and respond to an **API**, **application programming interface**, which means that software has been written to allow them to take patterns of 0s and 1s and change color and brightness.

  # We'll take a closer look at APIs soon, as we can build lots of useful applications by using all this code already implemented for us, rather than redoing everything ourselves. For example, if we want SMS in our application, to interact with customers programmatically, there are services that help us do that through an API.

ASCII

- So far we've represented numbers, but we also know that computers can represent letters of the alphabet. Some years ago the world decided on a standard mapping of numbers to letters, called **ASCII**[11].
- This website[12] shows you the mappings of all 256 values that an 8-bit piece of data can be. In particular, note that the letter `A` is the decimal number 65, `B` 66, `C` 77, and so on. And lowercase letters are another set of numbers. Numbers, too, that you type, would be represented as different numbers according to the table.

  # A sequence of bits that were `00000000` would be a special character, called `NULL`, according to ASCII. And you can see the other special characters in the table, too.

- To send the message `HI`, you'd send two numbers, `72` and then `73`. Each number would be represented by 8 bits, for a total of 16 bits to be sent.

  # And we'd know how to separate the numbers, because every 8 bits is a byte, and ASCII tells us to translate the message in units of one byte at a time.

[10] http://www2.meethue.com/en-us/
[11] http://en.wikipedia.org/wiki/ASCII
[12] http://www.asciitable.com/

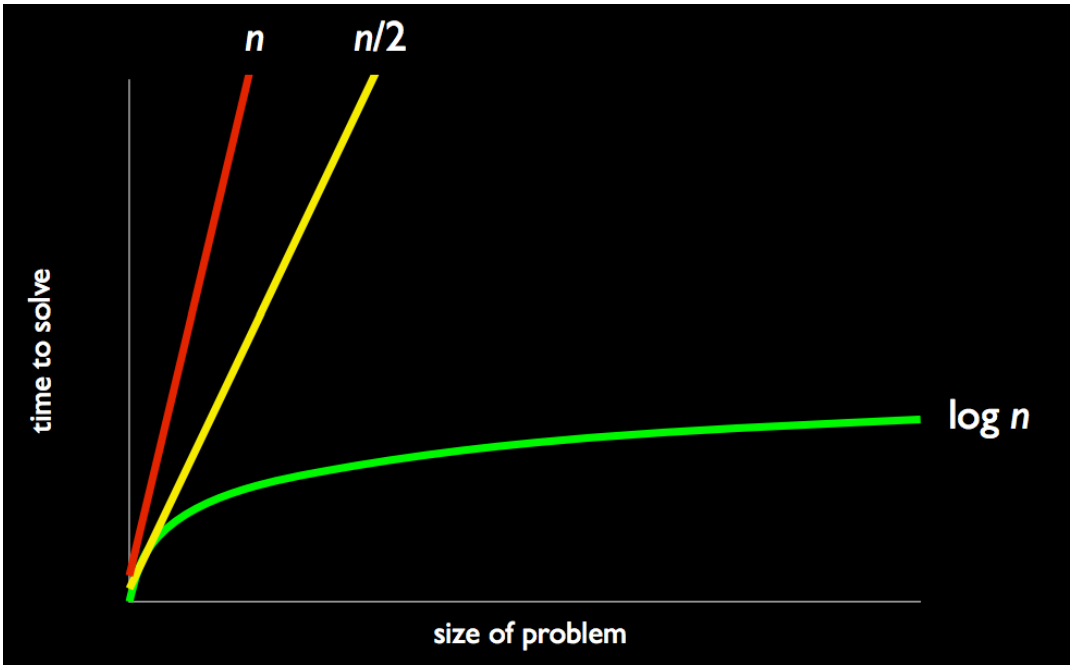
    # Even before computers, morse code was used, with dots and dashes, but each letter could have one, two, or more symbols. So people would pause between each character, but that was kind of cheating, since a pause is a third kind of symbol, in addition to dots and dashes.

- Additional symbols, like letters with accent marks, Asian languages, or even emoji, require a more sophisticated standard called Unicode[13], which uses additional bits per character to represent many more characters.
- So now we can move on to solving problems a slightly higher level, knowing that humans and computers can share the same information, just in different formats. Let us also take for granted that images, sounds, and videos, too, are represented somehow by 0s and 1s.

Algorithms

- Now that we can share data with computers, we can solve problems.
- Let's take a phone book, with, say, 1000 pages. And say that one of the pages contain the name Mike Smith.
- We could open the book and flip from the beginning, one page at a time, until we reached Mike Smith. And this algorithm would be correct, since we'd eventually reach his name. But this isn't a particular efficient solution.
- We could go twice as fast by flipping two pages at a time, but this algorithm would no longer be correct since we might skip Mike Smith if we got unlucky.
- Another way is to flip through the book one chunk at a time, skipping by the letter until we reached S.
- But let's do something simple, like opening the book to the middle. We'll see that we're in the Ms, which means that Mike Smith must be in the second half of the phone book, as S comes after M. So we can tear this problem in half, since we know we can get rid of the first half. We flip to the middle of what we have left, see that we're in the Ts, and keep just the left half since S comes before T. And we'll repeat this until we have one page, hopefully with Mike Smith's name on it.
- We can plot the relationship between the size of the problem and the time it takes to solve in this graph:

[13] http://en.wikipedia.org/wiki/Unicode



    # The red line, labeled `n`, is our first algorithm where we flipped one page at a time, meaning that a problem of size `n` would take `n` steps to solve. In our case, a phone book with 1000 pages would require 1000 page flips, and every additional page would require an additional step.

    # The yellow line, labeled `n/2`, is our next algorithm where we flipped two pages at a time. This is slightly more efficient than the red line, but still straight as every additional two pages add one more step, regardless of the size of the problem.

    # Finally, the green line, labeled `log n`, is saying that the time required to solve a problem grows logarithmically. In our case, with our halving algorithm, even a phone book twice the size would only take one additional step.

        # Let's say a database has a maximum size of 2^32 records. Searching through this database would take just 32 steps.

- But we also have to assume that the data is sorted ahead of time. Otherwise, we can't throw away half of the problem, and have to revert to our first idea of looking at one item at a time.
- The size of memory given to a piece of data in a database, too, is independent of standards like ASCII or Unicode. For example, the counter in YouTube was originally a number of size 32 bits, but has probably since been increased to a 64-bit number.

- Also, it turns out that the story[14] about the view counter on YouTube was actually fake[15], but we can pretend it was true, for learning's sake …

Pseudocode

- Let's use **pseudocode**, something English-like, to explain our algorithm:

```
pick up phone book
open to middle of phone book
look at names
if "Smith" is among names
    call Mike
else if "Smith" is earlier in book
    open to middle of left half of book
    go to line 3
else if "Smith" is later in book
    open to middle of right half of book
    go to line 3
else
    give up
```

- We can cut some corners here. In step 3, for example, we say "look at names," but how or what isn't exactly specified, since we really mean something like "look at all the last names, in both columns on the page." In fact, step 3 could be a function, or procedure, in a program, that we use to complete part of a task.
  # CPUs in your computer understand simple functions, like add or subtract, or move or load, with which we can build these larger programs.
  # And the term "hardcode" just means that something is permanently set, whether by physical wires, or software that can't be removed or changed easily.
- Step 4 has a **condition**, which mean that step 5 will only happen if step 4 is true. We indent to make this extra clear. And lines like step 5 or steps 1-3 that do something are called **statements** or **instructions**.
- Step 6 is another condition, like another fork in the road, that will cause other things to happen if it's true.

[14] http://www.economist.com/blogs/economist-explains/2014/12/economist-explains-6
[15] http://www.cnet.com/news/gangnam-style-busts-youtubes-view-counter-not-so-fast/

- Step 8 is somewhat interesting, since it creates a loop, or cycle, that brings us back to step 3. And it seems like we'll continue doing this, until we get to that last condition in step 12.
- The terminal condition in step 12, known as a **base case**, is the case where we can end our loop since we have done everything we could. In this case, it's when Mike Smith is not in the phone book, and we have no more pages to look at. (Notice that we could end our loop if we find Mike Smith before we went through the entire phone book, but this depends on the condition of finding him. In contrast, the base case will keep our loop from continuing forever for every case, as eventually we will have looked at all the pages and have nothing else to do.)
- The final `else` in step 12 is technically unnecessary, but we choose to include it for completeness and clarity.

Let's conclude with a final algorithm:

```
1. stand up and assign yourself the number 1
2. pair off with someone standing, add your numbers together, and adopt the
 sum as your new number
3. one of you should sit down; the other should go back to step 2
```

- We followed this algorithm, and eventually 2 people remained. One had the number of 18, and one had 32, making for a total of 50 (by happy coincidence). But while we did this, David counted everyone in the room one at a time, and got a total of 90. So we found our first bug, but in theory we should have counted ourselves much more quickly than one at a time. If 90 more people came in the room, it might take David twice as long, but with this algorithm, we'd only take one more step to count twice as many people.
- Don't forget to do Assignment 0[16].

[16] http://cs50.harvard.edu/mba/assignments/0/