

CS1315: Introduction to Media Computation

Speed and Complexity
Why is Photoshop so much faster?

Today's class

- Complexity of algorithms
- Compilation and interpretation
 - **Assembly language**
 - **Interpreters**
 - **Compilers**
- Hardware

More than one way to solve a problem

- There's always more than one way to solve a problem.
 - **To go from Architecture to the Student Center you can cut across the parking lot and the field, OR you can walk just on sidewalks**
- Some solutions are better (faster, shorter, less muddy) than others.
- How do you compare them?

Finding something in the dictionary

- Simple algorithm
 - **Start from the beginning.**
 - **Check each page, until you find what you want.**
- Not very efficient
 - **Best case (you find the word on the first page you check): One step**
 - **Worse case (it's on the last page in the dictionary): n steps where n = number of pages**
 - **Average case: n/2 steps**

A better search in a sorted list

- Better algorithm
 - Split the dictionary in the middle.
 - Is the word you're at before or after the page you're looking at?
 - If after, look from the middle to the end.
 - If before, look from the start to the middle.
 - Keep repeating until done or until it couldn't be there.
- More efficient:
 - Best case: It's there in the first place you look.
 - Average and worst case: $\log n$ steps

The Traveling Salesman Problem

- Imagine that you're a sales person, and you're responsible for a bunch of different clients.
 - Let's say
- To be efficient, you want to find the shortest path that will let you visit each client exactly once, and not more than once.
- Being a smart graduate of CS1315, you decide to write a program to do it.

The Traveling Salesman Problem currently can't be solved

- The best known algorithm that gives an optimal solution for the Traveling Salesman Problem depends on the *factorial of the cities*
- For 30 cities, the number of steps to be executed is 30!
 - $30! = 265,252,859,812,191,058,636,308,480,000,000$
- The Traveling Salesman Problem is real.
 - For example, several manufacturing problems are essentially the same as this problem,
 - e.g. moving a robot on a factory floor to process things in an optimal order.

Implications of algorithmic analysis

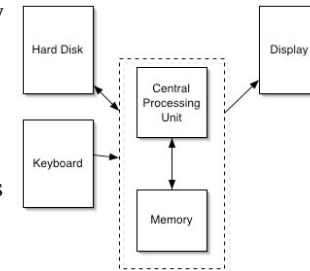
- Sometimes, the simplest approach to solving a problem isn't the best
 - For small amounts of data, this may not matter, but for large data sets, the more efficient algorithm can rapidly become hundreds, thousands, millions, times faster as the data set grows
- Some easily stated problems have no reasonable correct solution
 - We can't wait for billions of years for the answer, so those problems are intractable
 - The best we can do is find a good enough solution using rules of thumb, or heuristics

Today's class

- Complexity of algorithms
- Compilation and interpretation
 - **Assembly language**
 - **Interpreters**
 - **Compilers**
- Hardware

Each *kind* of processor has its own machine language

- Apple computers typically use CPU (*processor*) chips called G4 or G5.
- Computers running Microsoft Windows may use Pentium processors.
- There are other processors called Alpha, LSI-11, and on and on.



Each processor understands only its *own* machine language

Machine language is executed *very quickly*

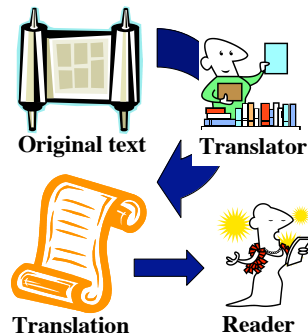
- A mid-range laptop these days has a *clock rate* of 1.5 Gigahertz.
- What that means *exactly* is hard to explain, but let's interpret it as processing 1.5 *billion* bytes per second.
- Those 12 bytes would execute inside the computer, then, in $12/1,500,000,000^{\text{th}}$ of a second

Compiled vs. Interpreted Programs

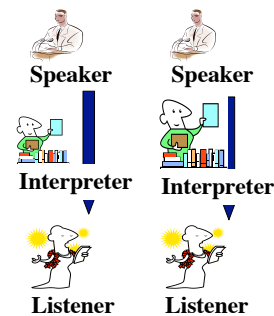
- Applications like Adobe Photoshop and Microsoft Word are *compiled*.
 - **This means that they execute in the computer as pure machine language.**
 - **They execute at *that* level speed.**
- However, Python, Java, Scheme, and many other languages are (in many cases) *interpreted*.
 - **They execute at a slower speed.**
 - **Why? It's the difference between *translating* instructions and directly *executing* instructions.**

Two ways to translate

Translation a.k.a. compiling



Interpreting



Applications are *compiled*

- Applications like Photoshop and Word are written in languages like C or C++
 - These languages are then *compiled* down to machine language.
 - That stuff that executes at a rate of 1.5 billion bytes per second.
- Python programs are interpreted.
 - Actually, they're interpreted *twice!*

Why interpret?

- For us, to have a command area.
 - **Compiled languages don't typically have a command area where you can print things and try out functions.**
 - **Interpreted languages help the learner figure out what's going on.**
- For others, to maintain portability.
 - **Java can be compiled to machine language.**
 - In fact, some VMs will actually compile the virtual machine language for you while running—no special compilation needed.
 - **But once you do that, the result can only run on one kind of computer.**
 - **Programs for Java (.jar files typically) can be moved from any kind of computer to any other kind of computer and just work.**

Java tries to get the best of both worlds

- Java code is *compiled* into a machine language (called byte code) for a fictitious machine, the Java Virtual Machine.
- Each computer that can execute Java has an *interpreter* for the Java machine language.
- Interpreting Java machine is pretty easy
 - **Takes only a small program**
 - **Devices as small as wristwatches can run Java VM interpreters.**

Today's class

- Complexity of algorithms
- Compilation and interpretation
 - **Assembly language**
 - **Interpreters**
 - **Compilers**
- Hardware

What makes a *computer* fast?

- If you ran the exact same program on two different computers, it will run faster on one than the other.

MOBILE INTEL® CELERON® PROCESSOR 2.8GHZ NOTEBOOK WITH DVD /CDRW COMBO DRIVE AND CANON I80 PRINTER

- 256MB DDR SDRAM memory
- 40 gigabyte hard drive (5121) 4388526
- Minimum 5 Toshiba 5121 notebooks per store. No rainchecks.

- *Why?*
- When you want to buy a *fast* computer, what should you look for?

- Intel® Celeron® Processor 2.7GHz
- CD-RW Drive
- 400MHz Front Side Bus
- 128KB L2 Cache
- 256MB DDR SDRAM
- 40.0GB Hard Drive

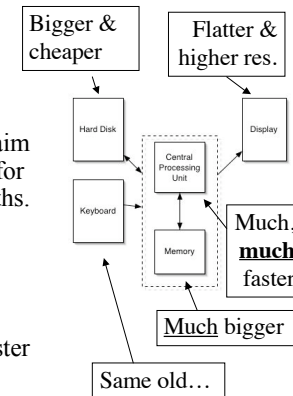
- AMD Athlon™ XP Processor 3000+ with QuantiSpeed™ Architecture
- 400MHz Front Side Bus
- 512KB L2 Cache
- DVD-ROM Drive
- CD-RW Drive
- 512MB DDR SDRAM
- 120.0GB Hard Drive

The Clock drives the cpu

- The CPU does *something* every time that the clock ticks
- A clock rate of 1.5 gigahertz means that the CPU is told to do *something* once every 1.5 *billionth* of a second.
- For the *same kind of processor*, then, a faster clock rate means faster processing.
- For *different* processors, it depends on how much the processor does for each instruction

Moore's "Law"

- Gordon Moore, one of the founders of Intel, made the claim that computer power doubles for the same dollar every 18 months.
- This has held true for over 30 years.
- And in three years time, the same priced computer will probably do this four times faster than today.



The biggest factor in speed: *Storage!*

- Your CPU can only work on data that it has.
- Where does it get the data?
 - **Registers are your CPU's hands—that's where it's working on data**
 - **Cache memory is super-fast memory that's almost as fast as registers**
 - **RAM storage is slower than cache memory but thousands of times faster than...**
 - **Disk storage which is thousands of times *larger than* RAM storage (because it's cheaper)**
 - **And the network is slowest of all to access data.**
- How slow?

Storage Latency: How Far Away is the Data?



Jim Gray