Computers Are Not That Great!

Lawrence Snyder
University of Washington, Seattle
Computers are amazingly fast ... but that’s because we usually ask them to do really easy stuff; they can do billions of instructions per second (gips?) ... 

So, what’s a “really easy” computation? 

- Looking up in a dictionary or address book how the letters you’ve typed might be completed 
- Recovering a losslessly compressed file 
- Looking in a file for a specific letter string 
- ...
How long would it take the Census Bureau to alphabetize the US population by first name?

- Recall Exchange Sort & Bubble Sort – both are \( n^2 \) algorithms, meaning they take \( cn^2 \) seconds for some amount of time \( c \) – usually the overhead to process one item; lets estimate \( c \approx 0.5 \ \mu s \)
- If the US population is \( n = 310,000,000 = 3.1 \times 10^8 \)
  - \( n^2 = 3.1 \times 10^8 \times 3.1 \times 10^8 = 9.6 \times 10^{16} \)
  - \( 0.5 \ \mu s = 0.000 \ 000 \ 5s = 5.0 \times 10^{-7}s \)
  - \( cn^2 = 5.0 \times 10^{-7}s \times 9.6 \times 10^{16} = 48 \times 10^9s = 1521 \ \text{years} \)
Recall there was also the Merge Sort

Merge Sort requires $cn \log_2 n$

$$cn \log_2 n = 5.0 \times 10^{-7} \times 3.1 \times 10^8 \times 29$$

$$= 15.5 \times 10^5 \times 29 = 4495s = 1.24 \text{ hr}$$
The input size $n = 310,000,000 = 3.1 \times 10^8$

- Exchange sort & Bubble sort require $cn^2$ time
- For $c=1/2$ microsecond, $cn^2 = 1521$ years
- Merge sort requires $cn \log_2 n$ time
- Because $\log_2 n = 29$, $cn \log_2 n = 1.24$ hours

- Algorithms matter ... and smarter algorithms are better
More data means more work, but sometimes it means a lot more work.

Traveling salesman problem:
Minimize the cost of the plane tickets

- Finding a tour is reasonably easy
- Finding the cheapest tour is NP-hard
Minimize the cost of the plane tickets

- Finding a tour is reasonably easy
- Finding the cheapest tour is NP-hard
NP, which stands for “nondeterministic polynomial time” (don’t learn that), is a class of problems with these features:

- They are easy (like $cn^2$, perhaps) ways to solve if the computer can guess and is always right
- They have no known easy (like $cn^5$, say) solutions, it seems, if the computer can’t guess, which it can’t
- All known solutions effectively check all possible alternatives and pick the best
- These are “normal” computations, like TSP
- “Complete” means solve one and you’ve solved all
... Are Data

For Example: Processing is a program that accepts YOUR program as data and runs it ... so it “computes on” (processes) your program

Except for really trivial languages (e.g. HTML) all programming languages are universal – CS people can write a program in that one language, say Processing, which can run programs in any other language – all programs

This is the “Universality Principle”
String code1, code2;
void setup() {
  size(500, 400);
  background(255);
  noLoop();
  fill(0);
}

void draw() {
  code1 = "\"String code1, code2; \" +
    "void setup() { \" +
    "size(500,400); \" +
    "background(255); \" +
    "noLoop(); fill(0); \} \" +
    "void draw() { \" +
    "code1 = ;\"\";\n  code2 = "code1 = code1 + code1 + \"code2 = \" + code2 + code2; \" +
    "text(code1, 50, 50, 200, 400); \}" \";
  code1 = code1 + code1 + "code2 = " + code2 + code2;
  text(code1, 50, 50, 200, 400);
}
Divide the program into two halves --

- code1 = [red bar]
- code2 = [blue bar]

display: [bar chart with red and blue segments]
Running the Program ...

- Output

```
"String code1, code2; void setup( ) { size(500,400);
background(255); noLoop( );
fill(0); } void draw( ) { code1 = 
"String code1, code2; void setup( ) { size(500,400);
background(255); noLoop( );
fill(0); } void draw( ) { code1 = 
"code2 = code1 + code1 + 
"code2 = " + code2 + 
"code2 = " + code2 + code2;
text(code1, 50, 50, 200, 400); }
" code1 = code1 + code1 + 
"code2 = " + code2 + code2;
text(code1, 50, 50, 200, 400); }
```

- Helpfully Formatted Output

```
"String code1, code2; void setup( ) { size(500,400);
background(255); noLoop( );
fill(0); } void draw( ) { code1 = 
""code2 = code1 + code1 + 
"code2 = " + code2 + code2;
text(code1, 50, 50, 200, 400); }
" code1 = code1 + code1 + 
"code2 = " + code2 + code2;
text(code1, 50, 50, 200, 400); }
```

```
code1 = code1 + code1 + "code2
= " + code2 + code2;
text(code1, 50, 50, 200, 400); }
```
Adding Additional Code

- Notice that new code can be added, and the program can still print itself out

```java
void draw() {
  String code1, code2; 
  "void setup() { " +
  "size(500,400); " +
  "background(255); " +
  "noLoop(); fill(0); } " +
  "void draw() { " +
  "code1 = ;" +
  "code2 = \"";
  code2 = "code1 = code1 + code1 + \"code2 = \" + code2 + code2; " +
  "text(code1, 50, 50, 200, 400); }\" ";
  code1 = code1 + code1 + "code2 = " + code2 + code2;
  text(code1, 50, 50, 200, 400);
}
```

Put the new code here and here
A self-printing program shows that programs can manipulate program text ...

Examples of programs manipulating programs
- The highlighter that “colors” your programs
- The translator that converts Processing code into machine code so a computer can run it
- The code that figures out what you did wrong when you forget a semicolon
- A debugger can help you find errors in your pgm
Suppose we want to determine if a Processing program draws a red circle or not

It seems possible, perhaps ...

- Analyze the code to see if it displays any circles
- Check if any of the circles it draws are red
- Etc.

Suppose Boolean check-pde(String code) is a Processing function that determines if a Processing program draws a red circle (return true) or does not draw a red circle (return false)
String code = "void trick( )... " ;
void setup( ) {
    size(200,200); background(255); noLoop( ) ;
}
void draw( ) {
    trick( ); // Guaranteed to get it wrong!
}
void trick( ) {
    if (check-pde(code)) { // does code draw red circle?
        fill(0,0,255); // check-pde says yes
    } else {
        fill(255,0,0); // check-pde says no
    }
    ellipse(100,100,10,10);
}
The Impact

- There are simple problems that computers cannot solve, b/c probs are not algorithmic ... no deterministic sequence of operations can find the answer; debugging is an example
- Alan Turing’s insight in 1936
We considered how “hard” computations can be, where “hard” is measured as running time.
- Linear time – thinking about how long the code runs
- Quadratic; NlogN – thinking about sorting
- NP Hard and the TSP
- Universal machine – yeah Turing!
- Undecidibility