CS 111 Green: Program Design I
Lecture 27: Speed (cont.);
parting thoughts

Robert H. Sloan (CS) & Rachel Poretsky (Bio)
University of Illinois, Chicago
December 5, 2017
Please complete the post-class survey (now)

https://uic.qualtrics.com/jfe/form/SV_9LYcB6ZTePIVOIL

Link also available from front page of course Blackboard site
Questions on Phylogenetic Tree Assignment?
How are you doing?

- It's the end of the semester, a stressful time, for all of us, and end of the semester for our first year in college for many of us, a particularly stressful time.
  - A. Not a care in the world
  - B. Awfully busy but I'm feeling really fine
  - C. Somewhat stressed
  - D. Quite stressed
  - E. Very stressed
SPEED AND LANGUAGE TRANSLATION: COMPILED VS. INTERPRETED
Some final course mechanics: Profs do read Piazza

- Next class: Entirely review for final

```python
if final_exam(student) >= 80:
    replace_any_lower_hour_exams(student)
```
LOAD R1,#65536 ; Get a character from keyboard
TEST R1,#13 ; Is it an ASCII 13 (Enter)?
JUMPTURE #32768 ; If true, go to another part of the program
CALL #16384 ; If false, call func. to process the new line

Machine Language (13 bytes):
05 01 255 255
10 01 13
20 127 255
122 63 255
Machine language is executed very quickly

- A mid-range laptop these days has a clock rate of 1.3–2.6 Gigahertz.
- What that means exactly is hard to explain, but let’s interpret it as processing 1.5 billion bytes per second.
  - Probably correct to within multiplicative factor of 8
  - Certainly correct to within multiplicative factor of 100
- Those 13 bytes would execute inside the computer, then, in $\frac{13}{1,500,000,000,000}$th of a second!
Applications are typically compiled

- 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*, JavaScript, PHP, R, 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.
  - *Java technically intermediate case (complied to bytecode)
Why do we write programs?

- One reason we write programs is to be able to do the same thing over-and-over again, without having to rehash the same steps at the Python command line in the interpreter window each time.
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.
High-level languages must be translated to machine language

- “High-level language” means any computer language except assembly language and machine language
- *All* higher-level languages must be translated to machine language before they can be executed.
- Two kinds of translators from source to target:
  - Compiler: C, C++, FORTRAN
  - Interpreter: Python, JavaScript, PHP, Flash, PDF, Java*
Interpreter idea (Using Python to illustrate)

Python Code

```
print("Hello, World!")
```

Probably in file hello.py somewhere

Interpreter

Translates Python code to machine language 1 line at a time and executes it

For us, lower-right console window of Spyder; could also use terminal window
Compiler idea (Using C++ to illustrate)

```cpp
#include <iostream>

int main()
{
    cout << "Hello, World!";
}
```

This is an app you can run from, e.g., a terminal window.
Which is faster for same program & inputs?

A. Running Python program
B. Starting with C++ source code, translating it using a C++ compiler, and then running that program
C. Not enough information; it might be close between them
D. No Clue
Which is faster for same program & inputs?

A. Running Python program
B. Starting with the result of compiling C++ source code, and then running that compiled executable
C. Not enough information; it might be close between them
D. No Clue
Java programs typically don’t compile to machine language.

- Recall that every processor has its own machine language.
  - How, then, can you create a program that runs on any computer?
- The people who invented Java also invented a make-believe processor—a virtual machine.
  - It doesn’t exist anywhere.
  - Java compiles to run on the virtual machine
    - The Java Virtual Machine (JVM)
    - The language it compiles to is called Java Bytecode
What good is it to run only on a computer that doesn’t exist?!?

- Machine language is a very simple language.
- A program that *interprets* the machine language of some computer is not hard to write.
  ```python
def VMInterpret(program):
    for instruction in program:
      if instruction == 1:  # It's a load
        ...
      if instruction == 2:  # It's an add
        ...
  ```
Java runs on everything...

- Everything that has a JVM on it!
- Each computer that can execute Java has an **interpreter** for the Java machine language, Java bytecode
- Interpreting Java bytecode is pretty easy
  - Takes only a small fast program
- Devices as small as wristwatches can run Java VM interpreters.
- Intermediate between compiled and interpreted language, with some of the benefits of both
Is it any wonder that Python programs can be slower?

- Photoshop and Word simply execute
  - At 1.5 Ghz and faster!
- Python programs interpreted
  - So for flat-out speed, people don't use Python
- But:
  1. Some of the Java intermediate representation trick ideas have been adopted by other languages, including Python, so 2017 Python *lots faster* than 2005 Python
  2. Not so many things need flat out speed today
Which is least likely to be developed in Python?

A. Google putting up results page for a Google search
B. Large data science application, like studying Chicago crime rates by location, income, education
C. Graphics-intense multi-player computer game
D. Large computational biology applications
E. Automating the testing of large pieces of software for errors
Why interpret?

- For us, to have interpreter area on RHS of Spyder
  - 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.
- Same advantage for scientist data user and data scientist
  - Answer developers first; software developers second ("run once")
- 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*. 
Why else interpret?

- Interpreted languages are usually more flexible, *optimizing programmer time*, not running time.
- With great speedups in computers (Moore’s) law, interpreted languages tend to be fast enough for most things today.
- “Hot” languages 1997–2007 mainly interpreted: Java*, Python, Perl, Flash
- *And of those, Python and Java still hot today and (interpreted) JavaScript and R more important today than then*
THERE ARE WORSE THINGS THAN BEING A LITTLE SLOWER THAN THE BEST
Interpreted vs. compiled isn't so bad

- Is at worst a constant multiplicative factor
- And so many good optimization tricks for Python by 2017 that it's not a big multiplication factor
Imagine that you’re a sales person, and you’re responsible for a bunch of different clients. Let’s say 30. 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 CS 111, 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 takes \( n! \) steps (That’s *factorial*)
  - There are algorithms that are better that give close-to but *not* guaranteed-best paths
- For 30 cities, the number of steps to be executed is 265,252,859,812,191,058,636,308,480,000,000 (30!)
- The Traveling Salesman Problem is real.
  - For example, several manufacturing problems actually become this problem, e.g., moving a robot on a factory floor to process things in an optimal order.
Class P, Intractable, and Class NP

- Many problems (like sorting n elements) can be solved with a polynomial # of steps, like $n^2$
  - We call that Class P problems.
- Other problems, like optimization, have known solutions but are so hard and big that we know that we just can’t solve them a reasonable amount of time for even reasonable amounts of data.
  - We call these *intractable*
- Still other problems, like Traveling Salesman Problem *seem* intractable, but maybe there’s a solution in Class P that we just haven’t found yet.
  - We call these *class NP*
- Does P=NP? *BIG QUESTION!* (Also $1 million prize)
Then there are impossible problems

- There are some problems that are provably impossible.
  - We know that no algorithm can ever be written to solve this problem.
- The most famous of these is the *Halting Problem*
  - Which is, essentially, to write a program to completely understand and debug another program.
The Halting Problem

- We could have written programs that could read another program (open a file!).
  - Spyder is in the end just a program.
- Can we write a program that will input another program (say, from a file) and tell us if the program will ever stop or not?
  - Think about **while** loops with some complex expression—will the expression ever be false?
  - Now think about *nested while* loops, all complex…
- It’s been proven that such a program can never be written.
Alan Turing

- Brilliant mathematician and computer scientist.
- Came up with mathematical definition of what a computer could do...before one was even built!
  - The Turing machine was invented in answer to the question of what the limits of mathematics were: What is *computable* by a function?
- Proved that halting problem had no solution in 1936—almost 10 years before first computers were built.
Imagine that there IS a program to check if an input program halts or runs forever

```python
def doesItHalt(prog, input):
```

- We don’t have its code; we’re just assuming it exists, takes two inputs, returns True or False
- Will derive contradiction/paradox
def trouble(function):
    if doesItHalt(function, function):
        while (True):
            #I.e., ALWAYS!
            x = 1
    else:
        return

#What does trouble(text-of-trouble) do??????
Can we write a program that thinks?

- Are human beings doing (mere?) computation?
- Can human intelligence be captured in an algorithm?
  - Yes, we can create and understand programs, but some of our programs write programs too.
- Is it Class P? Class NP? Intractable?
- Are humans just computers in flesh?
  - These are questions that artificial intelligence researchers and philosophers study today.