UC Berkeley EECS
Sr Lecturer SOE
Dan Garcia

The Beauty and Joy of Computing

Lecture #7
Algorithms II

Your AI <u>isn't</u> going to rise up & kill you!

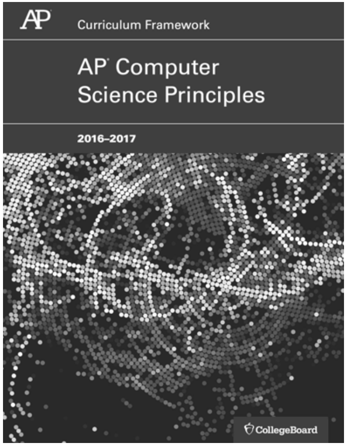
IBM AI researcher says the fear of conscious AI rising up and harming us is overblown, citing: (1) intelligence – say beating us in chess – is not consciousness and we're many many years from that, (2) "our AI are likely to derive its root volition from us", – your self-driving car, when not giving suggestions of where to charge it and get coffee, will sit in the garage, quietly charging.

http://www.washingtonpost.com/opinions/the-robots-are-not-going-to-kill-you/2015/02/06/550e0c6e-ae10-11e4-9c91-e9d2f9fde644_story.html

(AP) Computer Science Principles

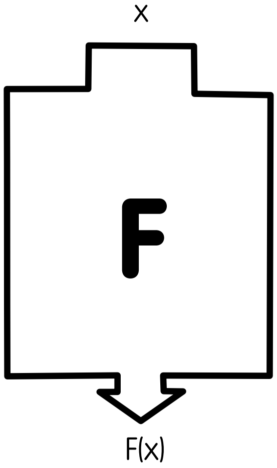
7 Big Ideas

- Creativity
- Abstraction
- Data and Information
- Algorithms
- Programming
- The Internet
- Global Impact



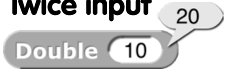
Functional Abstraction (review)

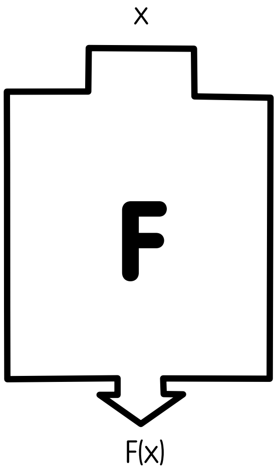
- A block, or function has inputs & outputs
  - Possibly no inputs
  - Possibly no outputs (if block is a command)
    - In this case, it would have a "side effect", i.e., what it does (e.g., move a robot)
- The contract describing what block does is called a specification or spec



What is IN a spec? (review)

- Typically they all have
  - NAME
  - INPUT(s)
    - (and types, if appropriate)
    - Requirements
  - OUTPUT
    - Can write "none"
  - (SIDE-EFFECTS)
  - EXAMPLE CALLS
- Example
  - NAME : **Double**
  - INPUT : **n** (a number)
  - OUTPUT: **Twice input**
  - SAMPLE:





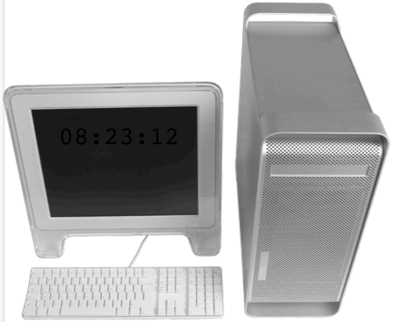
What is NOT in a spec?

- How!
  - That's the beauty of a functional abstraction; it doesn't say how it will do its job.
- Example: **`Double(n)`**
  - Could be n * 2
  - Could be n + n
  - Could be n+1 (n times)
    - if n is a positive integer
- This gives great freedom to author!
  - <u>You</u> choose Algorithm(s)!

☑

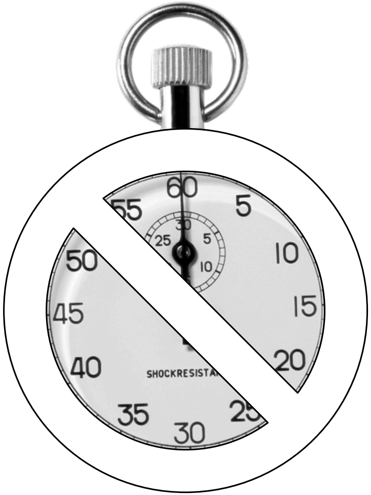
Algorithm Analysis: Running Time

- One commonly used criterion in algorithm analysis is **running time**
  - how long does the algorithm take to run and finish its task?
- How do we measure it?



Runtime Analysis: Problem and Solution

- Time w/stopwatch, but…
  - Different computers may have different runtimes. ☹
  - Same computer may have different runtimes on same input. ☹
  - Need to implement the algorithm first to run it. ☹
- That's called empirical analysis
- *Solution*: Count the #of "steps" involved, not time!
  - Each operation = 1 step
  - *If we say "running time", we'll mean # of steps, not time!*



Runtime Analysis: Input Size & Efficiency

- Given # of input things
  - E.g., # of list elements
  - E.g., # of sentence characters
- We define efficiency as a function of the input size
  - Running time (# of steps)
  - Memory Usage
- We determine efficiency by reasoning formally or mathematically
- Important!
  - In BJC <u>we won't care</u> about the efficiency of your solutions!
  - Usually they care in CS2

BJC

CS1

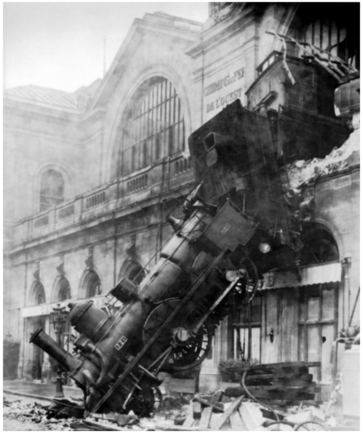
CS2

CS3

Runtime Analysis: Worst or Avg Case?

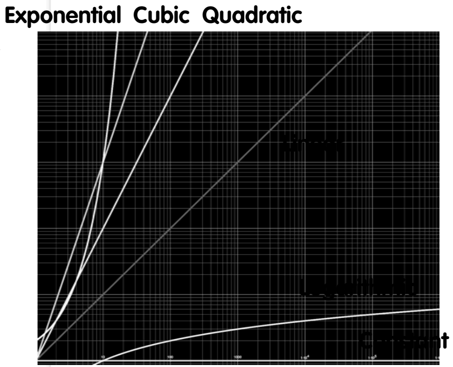
- Could use avg case
  - Average running time over a vast # of inputs
- Instead: use worst case
  - Consider running time as input grows
- Why?
  - Nice to know most time we'd <u>ever</u> spend
  - Worst case happens often
  - Avg is often ~ worst

Montparnasse Derailment
(*Wikipedia,* Public Domain)



Runtime Analysis: Order of Growth

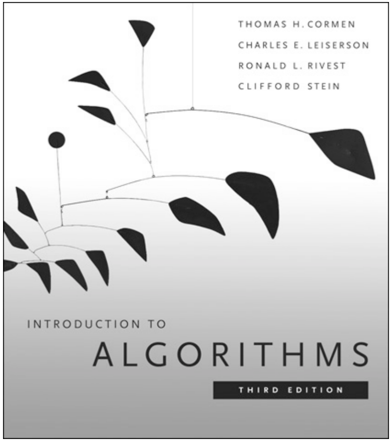
- Instead of an exact number of operations we'll use abstraction!
  - Want order of growth, or dominant term
- In BJC we'll consider
  - Constant
  - Logarithmic
  - Linear
  - Quadratic
  - Cubic
  - Exponential
- E.g. $10\ n^2 + 4 \log n + n$
  - …is quadratic
- More efficient algorithm
  - Could be more complex…
  - But helps handle larger input!



Graph of order of growth curves on log-log plot

Reference Text for Algorithms

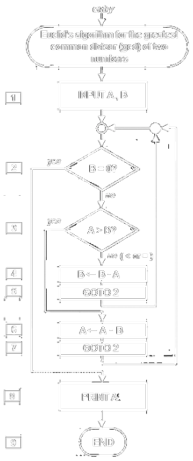
- This book launched a generation of CS students into Algorithm Analysis
  - It's on everyone's shelf
  - It might be hard to grok at this point, but if you go on in CS, remember it & own it!
  - …but get the most recent years



Algorithm Analysis: Is Algorithm Correct?

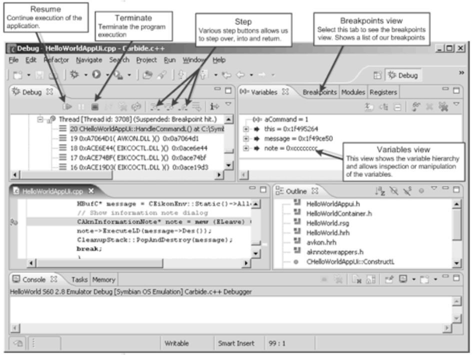
- An algorithm is correct if, for every input, it reports the correct output <u>and</u> doesn't run forever or cause an error.
  - Incorrect algorithms may run forever, or may crash, or may not return the correct answer.
    - They could still be useful!
    - Consider an approximation…
  - For now, we'll only consider <u>correct</u> algorithms

Euclid's GCD Algorithm
(*Wikimedia*)



How do you know if it is correct?

- For algorithms?
  - Reasoning formally or mathematically
- For programs? Empirically via testing:
  - Unit Testing
  - Debugging
  - Can <u>never</u> be sure algorithm is correct by testing implemention


Resume
Continue execution of the application.
Terminate
Terminate the program execution
Step
Various step buttons allows us to step over, into and return.
Breakpoints view
Select this tab to see the breakpoints view. Shows a list of our breakpoints
Variables view
This view shows the variable hierarchy and allows inspection or manipulation of the variables.
Debug - HelloWorldAppUi.cpp - Carbide.c++
File Edit Refactor Navigate Search Project Run Window Help
Debug
Variables
Breakpoints
Modules
Registers
HelloWorldAppUi.cpp
Outline
Console
Tasks
Memory
Writable
Smart Insert
99 : 1

```
Thread [Thread id: 3708] (Suspended: Breakpoint hit.)
    20 CHelloWorldAppUi::HandleCommandL() at C:\Symb
    19 0xA7064D1( AVKON.DLL )() 0x0a7064d1
    18 0xACE6E44( EIKCOCTL.DLL )() 0x0ace6e44
    17 0xACE74BF( EIKCOCTL.DLL )() 0x0ace74bf
    16 0xACE19D3( EIKCOCTL.DLL )() 0x0ace19d3
```

```
(x)= aCommand = 1
this = 0x1f495264
message = 0x1f49ce50
note = 0xcccccccc
```

```
HBufC* message = CEikonEnv::Static()->All
// Show information note dialog
CAknInformationNote* note = new (ELeave)
note->ExecuteLD(message->Des());
CleanupStack::PopAndDestroy(message);
break;
```

```
HelloWorldAppui.h
HelloWorldContainer.h
HelloWorld.rsg
HelloWorld.hrh
avkon.hrh
aknnotewrappers.h
CHelloWorldAppUi::ConstructL
```

```
HelloWorld S60 2.8 Emulator Debug [Symbian OS Emulation] Carbide.c++ Debugger
```

Summary

- When developing an algorithm, could optimize for
  - Simplest
  - Easiest to implement?
  - Efficient? (running time, memory)
  - Uses up least resources
  - Gives most precision
  - …
- In BJC we'll consider
  - Constant
  - Logarithmic
  - Linear
  - Quadratic
  - Cubic
  - Exponential
- Many problems can be solved in a reasonable time
  - Reasonable time means that as the input size grows, the number of steps the algorithm takes is proportional to the square (or cube, fourth power, fifth power, etc.) of the size of the input. (not exponential)
- Different correct algorithms for the same problem can have different efficiencies.
  - E.g., Linear search can be used when searching for an item in any list; binary search can be used only when the list is sorted.
- Some problems <u>cannot</u> be solved in a reasonable time, even for small input sizes.