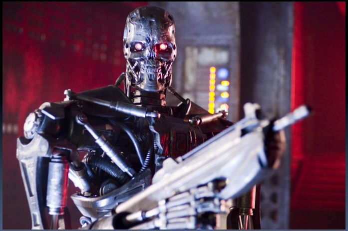
UC Berkeley EECS
Sr Lecturer SOE
Dan Garcia

The Beauty and Joy of Computing

Lecture #7
Algorithms II

Your AI isn't 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

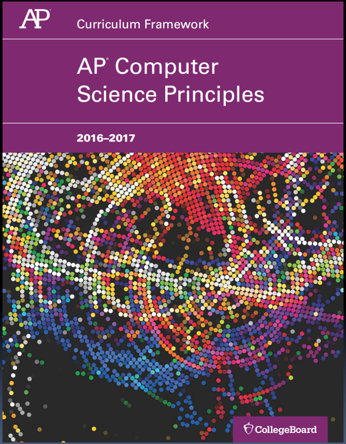
Algorithms: Specifications

(AP) Computer Science Principles

CS 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:

Garcia

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!
  - You choose Algorithm(s)!

Garcia

(Cal) What do YOU think?

Which factor below is the most important in choosing the algorithm to use?

A. Simplest?
B. Easiest to implement?
C. Takes less time?
D. Uses up less space (memory)?
E. Gives a more precise answer?

Algorithm Analysis

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?

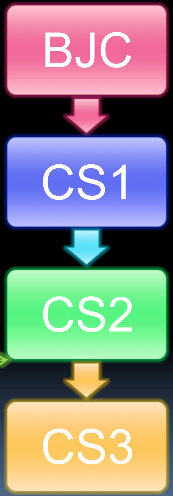
Runtime Analysis: Problem and Solution

- 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!*

Garcia

Runtime Analysis: Input Size & Efficiency

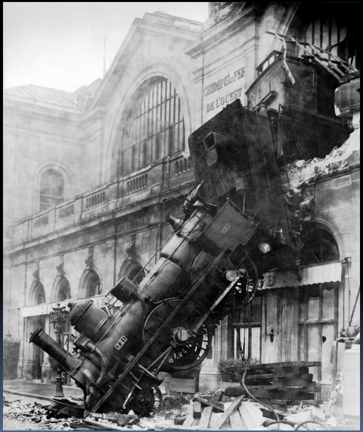
- Given # of input things
  - E.g., # of list elements
  - E.g., # of sentence characters
- We define efficiency as a function of the input size
  - Running time (# of steps)
  - Memory Usage
- We determine efficiency by reasoning formally or mathematically
- Important!
  - In BJC we won't care about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



Runtime Analysis: Worst or Avg Case?

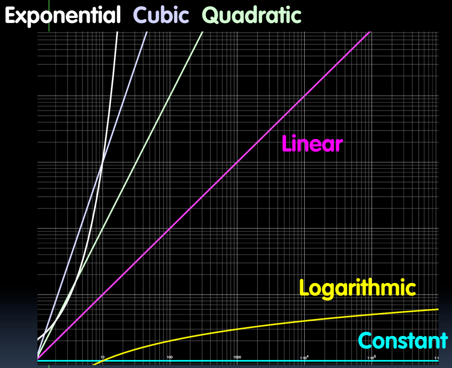
- Could use avg case
  - Average running time over a vast # of inputs
- Instead: use worst case
  - Consider running time as input grows
- Why?
  - Nice to know most time we'd ever spend
  - Worst case happens often
  - Avg is often ~ worst

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

Garcia

(Cal) Example: Finding a student (by ID)

- Input
  - Unsorted list of students L
  - Particular student S
- Output
  - True if S is in L, else False
- Pseudocode Algorithm
  - Go through one by one, checking for match.
  - If match, true
  - If exhausted L and didn't find S, false

- Worst-case running time as function of the size of L?
  1. Constant
  2. Logarithmic
  3. Linear
  4. Quadratic
  5. Exponential

(Cal) Example: Finding a student (by ID)

- Input
  - Sorted list of students L
  - Particular student S
- Output : same
- Pseudocode Algorithm
  - Start in middle
  - If match, report true
  - If exhausted, throw away half of L and check again in the middle of remaining part of L
  - If nobody left, report false

- Worst-case running time as function of the size of L?
  1. Constant
  2. Logarithmic
  3. Linear
  4. Quadratic
  5. Exponential

(Cal) Example: Finding a student (by ID)

- What if L were given to you in advance and you had infinite storage?
  - Could you do any better than logarithmic?

- Worst-case running time as function of the size of L?

1. Constant
2. Logarithmic
3. Linear
4. Quadratic
5. Exponential

(Cal) Example: Finding a Shared Birthday

- Input
  - Unsorted list L (of size n) of birthdays of team
- Output
  - True if any two people shared birthday, else False
- What's the worst-case running time?

- Worst-case running time as function of the size of L?
  1. Constant
  2. Logarithmic
  3. Linear
  4. Quadratic
  5. Exponential

(Cal) Example: Finding Subsets

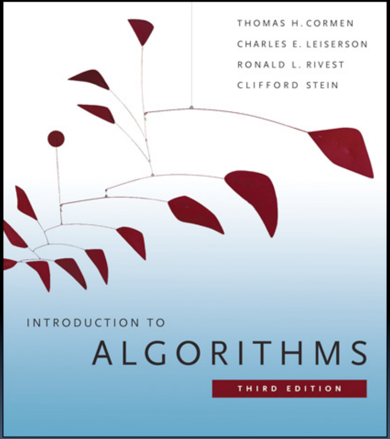
- Input:
  - Unsorted list L (of size n) of people
- Output
  - All the subsets
- Worst-case running time? (as function of n)
- E.g., for 3 people (a,b,c):
  - 1 empty: { }
  - 3 1-person: {a, b, c}
  - 3 2-person: {ab, bc, ac}
  - 1 3-person: {abc}

- Worst-case running time as function of the size of L?
  1. Constant
  2. Logarithmic
  3. Linear
  4. Quadratic
  5. Exponential

Algorithms: Correctness, Summary

Reference Text for Algorithms

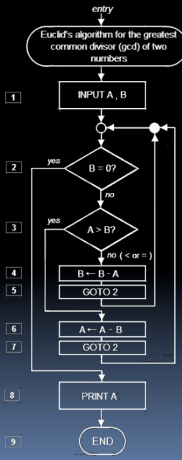
- This book launched a generation of CS students into Algorithm Analysis
  - It's on everyone's shelf
  - It might be hard to grok at this point, but if you go on in CS, remember it & own it!
  - …but get the most recent years



Algorithm Analysis: Is Algorithm Correct?

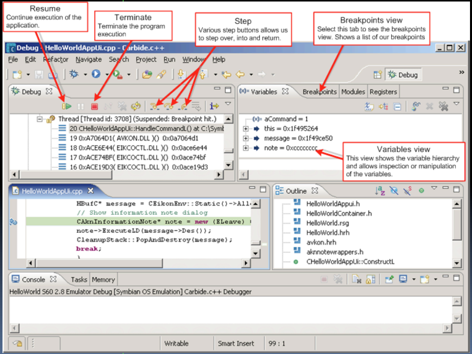
- An algorithm is correct if, for every input, it reports the correct output and doesn't run forever or cause an error.
  - Incorrect algorithms may run forever, or may crash, or may not return the correct answer.
    - They could still be useful!
    - Consider an approximation…
  - For now, we'll only consider correct 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 never be sure algorithm is correct by testing implemention



Garcia

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 cannot be solved in a reasonable time, even for small input sizes.