Advanced Programming and Algorithms

Dr. Baldassano

Yu's Elite Education

About Me

- Went to college at Princeton for Electrical Engineering

- PhD at Stanford in Computer Science

- Now a research fellow at the Princeton Neuroscience Institute

Goals of the class

- Learn about the most useful algorithms that power modern computing
- Learn how to apply algorithms to solve new problems
- Learn how to develop and analyze new algorithms

Course website

- Go to www.chrisbaldassano.com, click on Teaching, then “Advanced Programming and Algorithms”

- Has my email address <chrisb@princeton.edu>, schedule and assignments

Programming languages

- Most of this class will just use “pseudocode” and talk about algorithms at a conceptual level
- For homework assignments you can use any programming language you’re familiar with
- I will be using Python for solutions and demos, but the code should be understandable even if you don’t know Python

What are algorithms and data structures?

- Algorithm: A procedure for solving a problem
  - We are especially interested in *efficient* algorithms, that solve problems using as little time and/or resources as possible
- Data structure: A way of organizing information
  - If we know the types of operations we want to do on the data, we can organize it in a way that makes these operations fast and/or easy

Simple examples

- How to find where a particular term is defined in a book?
- Algorithm 1: Open to random pages to find the definition
  - Valid algorithm, will work eventually!
  - But very inefficient – could revisit pages multiple times
- Algorithm 2: Start from beginning, go one page at a time to find definition
  - Close to the best we can do for finding a single word
- Algorithm 3: Go through entire book, create an index of the page number where every word occurs, then use this index to find term
  - Silly to do for one lookup, but this index makes all future lookups very fast

Data structures

- All algorithms make use of data structures to keep track of computations in an efficient way
- Different data structures make different types of operations fast
- Example: lists of numbers
  - Array – fast to read out number at any position, slow to insert new number
  - “Linked list” – fast to insert a new number, slow to read out number at any position
- Often there is a tradeoff between memory and time
  - If we never throw any information away then we won’t have to recompute it, but it might take up lots of memory

Why study algorithms?

- All datasets are getting bigger and bigger:
  - Internet – web search, packet routing…
  - Science – genetics, brain imaging, climate modeling…
  - Computers – circuit layout, file systems…
  - Computer graphics – movies, video games, virtual reality…
  - Security – cell phones, e-commerce, voting machines…
  - Social networks – recommendations, news feeds…
- We need efficient algorithms to design almost any modern computer system

Why study algorithms?

- To become a more proficient programmer:
  - “I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.” — Linus Torvalds (creator of Linux)
- Because they’re interesting:
  - “For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.” — Francis Sullivan

Why study algorithms?

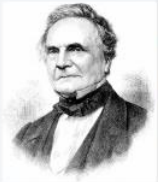
"People who analyze algorithms have double happiness. First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures. Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically."

— D. E. Knuth (1995)

Why study algorithms?

- To unlock the secrets of life and the universe:
  - “Computer models mirroring real life have become crucial for most advances made in chemistry today…. Today the computer is just as important a tool for chemists as the test tube.” — Royal Swedish Academy of Sciences
- To enable us to solve complex problems:
  - https://www.youtube.com/watch?v=RGJL6G9mw4E

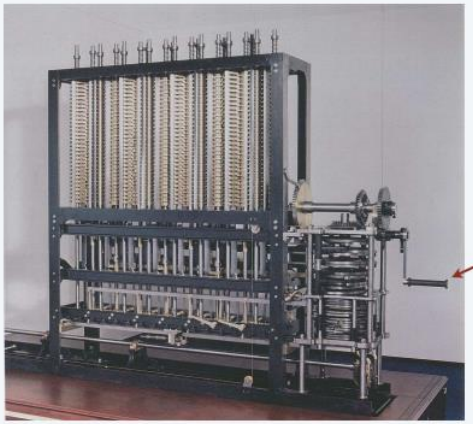
History of Algorithms



"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time?"

— Charles Babbage (1864)

Analytic Engine



how many times do you have to turn the crank?

History of Algorithms

"It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process . . ."

— Alan Turing (1947)

ROUNDING-OFF ERRORS IN MATRIX PROCESSES

By A. M. TURING

(*National Physical Laboratory, Teddington, Middlesex*)

[Received 4 November 1947]

SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known 'Gauss elimination process', it is found that the errors are normally quite moderate: no exponential build-up need occur.

Algorithm examples

- Guessing game: https://www.khanacademy.org/computing/computer-science/algorithms/intro-to-algorithms/a/a-guessing-game
- Route finding: https://www.khanacademy.org/computing/computer-science/algorithms/intro-to-algorithms/a/route-finding https://qiao.github.io/PathFinding.js/visual/
- Generating app names: http://mrsharpoblunto.github.io/foswig.js/
- Genetic algorithm: http://rednuht.org/genetic_walkers/

What makes an algorithm good?

- Want efficiency in *time* – produce an answer as quickly as possible
- Want efficiency in *memory* – use as little RAM as possible
- Want efficiency in *implementation* – the less complex the better
  - Want this to work on any dataset without hand-tuned conditions
- Want algorithms that scale well with size of data
  - If data gets 2x bigger, does algorithm require more than 2x as much time/space?

Measuring how an algorithm scales

- "Big-O notation"

- O(log N) – algorithm scales logarithmically with dataset size
- O(N) – algorithm scales linearly with dataset size (good)
- O(N * log N) – algorithm scales slightly worse that linearly
- $O(N^2)$ – algorithm scales quadratically with dataset size
- $O(2^N)$ – algorithm scales exponentially with dataset size

Measuring how an algorithm scales

	n	$n \log_2 n$	n^2	2^n
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	18 min
$n = 50$	< 1 sec	< 1 sec	< 1 sec	36 years
$n = 100$	< 1 sec	< 1 sec	< 1 sec	10^{17} years
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	very long

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

Example: Binary Search

- Given a sorted list of numbers, find where a specific number appears

  1 5 10 12 14 19 20

- At every step we can eliminate half the numbers

  1 5 10 ~~12 14 19 20~~

- So a list twice as big takes only *one* additional step!
- Running time is O(log N)

Example: Sum-3

- Given a list of numbers, are there three that sum to 0?

  -10 5 -2 3 -9 -8 20

- Easy algorithm: try every possible combination of three numbers
- Running time: $O(N^3)$

- Can we do better?

Sum-3 Algorithm

- First sort numbers

  -10 -9 -8 -2 3 5 20

- Assume we'll use first number
- Try pairing with first and last remaining numbers

  -10 -9 -8 -2 3 5 20

- Move in from left and right, depending on whether sum is above or below zero

  -10 -9 -8 -2 3 5 20

- If the green numbers hit each other, try again with assuming that the second number is used

  ~~-10~~ -9 -8 -2 3 5 20

Sum-3 Algorithm

- In the worst case we'll have to try all possible "first choice" red numbers
- For each of these, we might have to look at every other number
- Running time: $O(N^2)$
- If there are 1000 numbers, this is 1000 times faster!

- One trick: we assumed sorting was fast! Turns out we can sort faster than $O(N^2)$, so we can ignore this for now

Sieve of Eratosthenes

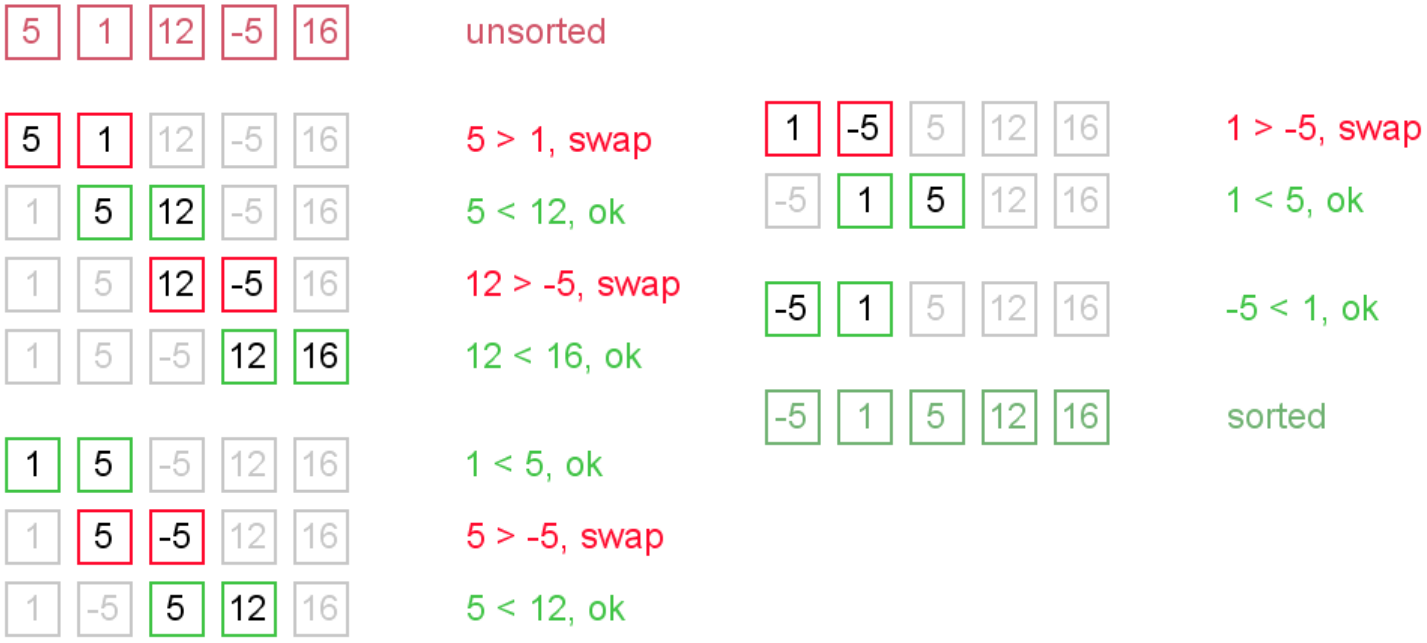
- http://www.algomation.com/algorithm/eratothanes-sieve

Warmup for next week's class: sorting

- Putting a list in order is a classic problem in computer science

- Simplest algorithm: "Bubble" sort
  - Compare every pair of adjacent elements
  - If they are out of order, swap them
  - Keep making passes over the list until it is sorted

Bubble sort


5 1 12 -5 16
unsorted
5 1 12 -5 16
5 > 1, swap
1 5 12 -5 16
5 < 12, ok
1 5 12 -5 16
12 > -5, swap
1 5 -5 12 16
12 < 16, ok
1 5 -5 12 16
1 < 5, ok
1 5 -5 12 16
5 > -5, swap
1 -5 5 12 16
5 < 12, ok
1 -5 5 12 16
1 > -5, swap
-5 1 5 12 16
1 < 5, ok
-5 1 5 12 16
-5 < 1, ok
-5 1 5 12 16
sorted

Assignment

- Implement bubble sort in any language
- How long does it take to sort 1,000 numbers? What about 10,000?
- Email your code and answers to these two questions to me at chrisb@princeton.edu
- Due before next class
- [FYI: Assignments are also posted at chrisbaldassano.com]

Re-scheduling December 10th class

- I will be away on December 10th
- We can reschedule the class for December 7th (Mon), or 11th (Fri)

- Part of your assignment: Email me which of these days you would be available