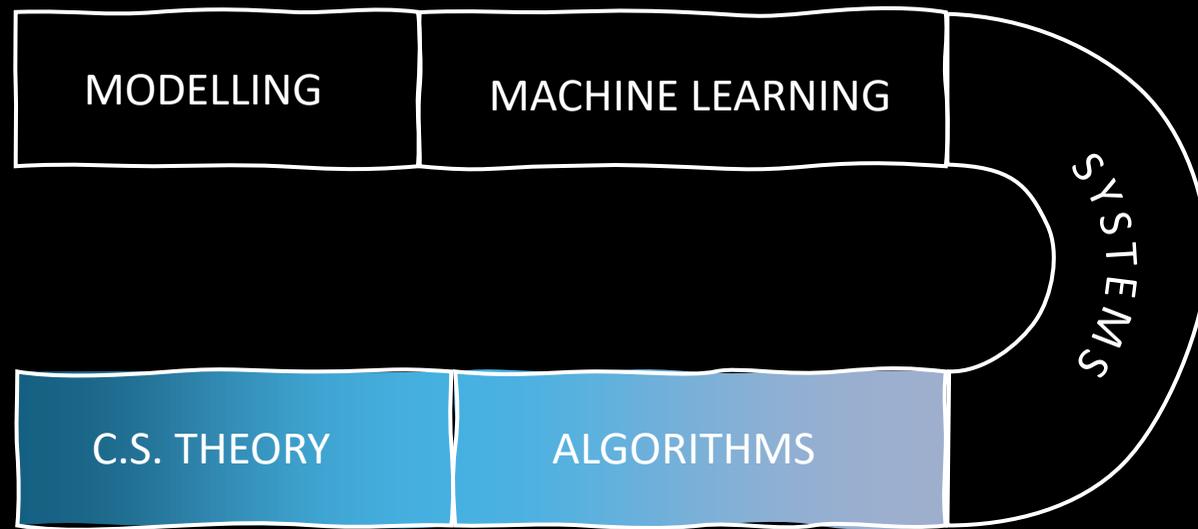


IA Algorithms 1 and 2

Dr Damon Wischik

~~What is an algorithm?~~

What sort of *activity* is working on algorithms?



Tight definitions
(but trivial proofs).
Elegant API design.

Tough proofs of correctness.
Cunning implementations
to get optimal performance.

SECTION 2

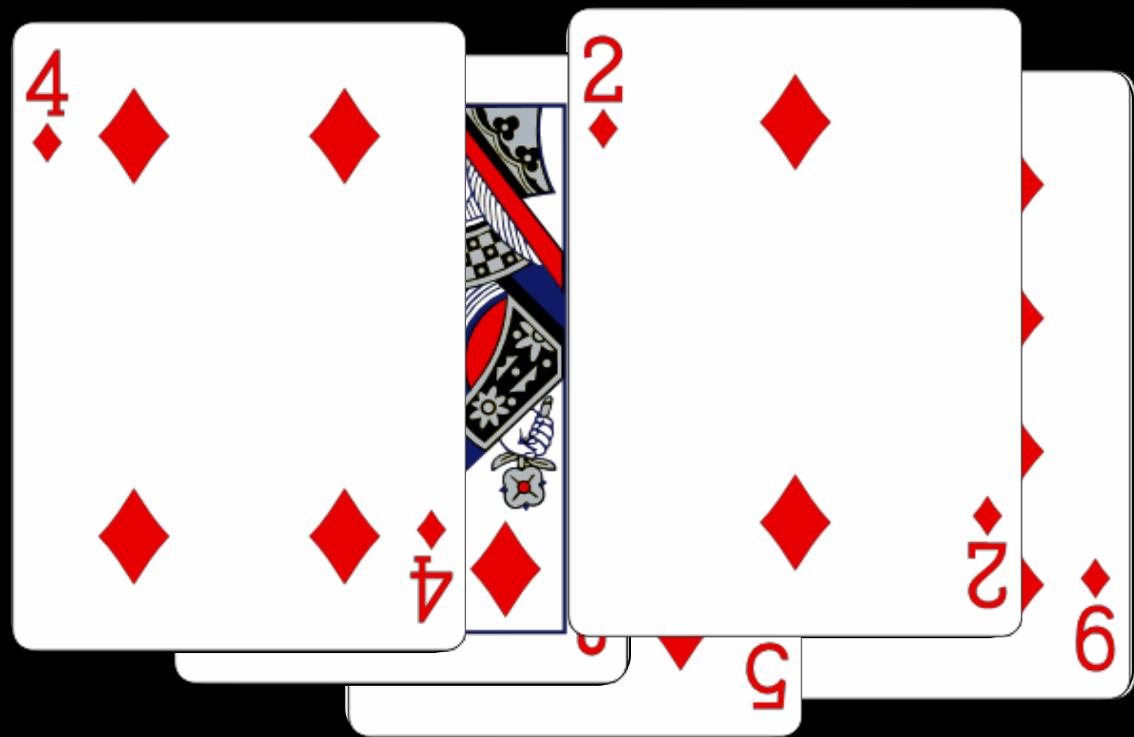
Sorting algorithms

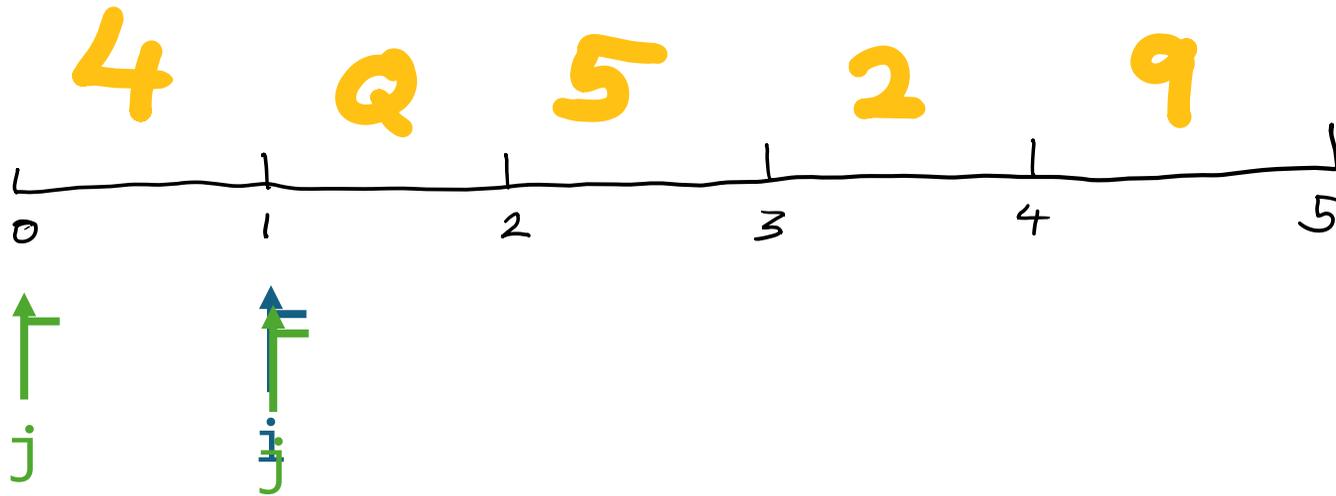
Insertion Sort

```
# let rec ins = function
  | x, [] -> [x]
  | x, y::ys ->
    if x <= y then
      x :: y :: ys
    else
      y :: ins (x, ys)

# let rec insort = function
  | [] -> []
  | x::xs -> ins (x, insort xs)
```

- Items from input are copied to the output
- Inserted in order, so the output is always sorted





NOTATION ALERT
Indexes refer to positions *between* cells.

$x[i]$ = cell just to the right of i

$x[i:j]$ = $x[i] \dots x[j-1]$

$n = \text{len}(x)$

$x[n]$ *X out-of-bounds error*

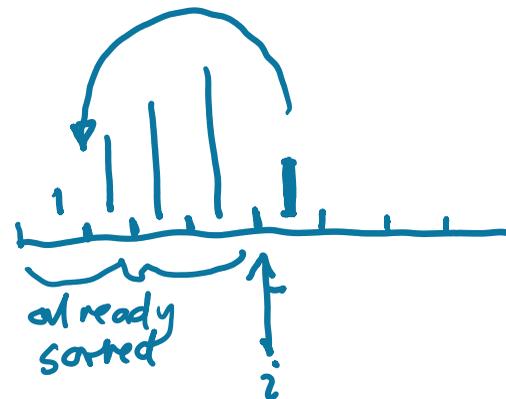
$x[n-1]$ last element.

```

1 def insert_sort(x):
2     for i in 1..(len(x)-1):
3         # assert x[0:i] is sorted
4         j = i - 1
5         while j >= 0 and x[j] > x[j+1]:
6             swap x[j] with x[j+1]
7             j = j - 1
8         # assert x[0:i+1] is sorted

```

1 ... len(x)-1 inclusive



- ❖ Is this algorithm correct?
- ❖ What is its performance?
Can we do better?

```
1 def insert_sort(x):
2     for i in 1..(len(x)-1):
3         # assert x[0:i] is sorted
4         j = i - 1
5         while j >= 0 and x[j] > x[j+1]:
6             swap x[j] with x[j+1]
7             j = j - 1
8         # assert x[0:i+1] is sorted
```

Department of Computer Science

cl.cam.ac.uk/teaching/2324/Algorithm1/materials.html

Teaching / Courses 2023–24 / Algorithms 1 / Course materials

Course pages 2023–24

Computer Laboratory ^

Teaching ^

Courses 2023–24 ^

Part IA CST ^

Algorithms 1

Databases

Digital Electronics

Discrete Mathematics

Foundations of Computer Science

Hardware Practical Classes

Introduction to Graphics

Object-Oriented Programming

OCaml Practical Classes

Registration

Scientific Computing Practical Course

Algorithms 1

Syllabus **Course materials** Ticks Recordings

Information for supervisors

Lecture notes — [printed notes](#) © Frank Stajano 2023 and [short handout](#)

Exercises — [Example sheets 1, 2, 3](#) (the same exercises as for 2022/23)

Announcements, Q&A, tick submission — [Moodle](#)

Schedule

This is the planned lecture schedule. It will be updated as and when actual lectures deviate from schedule. Slides will be uploaded the night before a lecture.

2. Sorting

Lecture 01 2.1 Insertsort
[\[slides\]](#) 2.3 Big-O notation

Lecture 02 2.4 The cost of sorting
2.6–2.9 Selectionsort, binary insertsort, bubblesort
2.9 Mergesort

Lecture 03 2.10 Heapsort

Lecture 04 2.11 Quicksort
2.12 Median and order statistics using quicksort
2.13 Stability
2.14 Faster sorting: counting sort, bucket sort, radix sort
Sorting algorithms compared
Example sheet 1, and Tick 1 (due 5 Feb)

3. Algorithm design

❖ lecture notes

❖ slides (uploaded the night before, and re-uploaded after the lecture)

❖ example sheets & ticks

❖ recordings

Pre-recorded videos from Prof Frank Stajano

Consent to recordings of live lectures

https://www.educationalpolicy.admin.cam.ac.uk/files/recording_policy.pdf

For any teaching session where your contribution is mandatory or expected, we must seek your consent to be recorded.

You are not obliged to give this consent, and you have the right to withdraw your consent after it has been given.

Q. Do you consent?

Official Part Ia Algorithms course at the University of Cambridge

Frank Stajano Explains
28 videos 21,027 views Last...

I have been lecturing the introductory Algorithms course at the University of C...More

Play all Shuffle

- 1 Introducing Algorithms: three hard problems (Algorithms @ Cambridge)
Frank Stajano Explains • 2.9K views • 11 months ago
- 2 Algorithms: insertsort
Frank Stajano Explains • 2.2K views • 3 years ago
- 3 Algorithms: asymptotic complexity, big-O notation, big-Theta, big-Omega, small-o, small-omega
Frank Stajano Explains • 1.8K views • 3 years ago
- 4 Algorithms: the cost of sorting
Frank Stajano Explains • 1.2K views • 3 years ago

How to learn effectively

PASSIVE LEARNING

- attend lectures
- read code snippets, watch animations, see examples
- read notes, watch videos

ACTIVE LEARNING

- copy out any handwriting from lectures
- annotate the code snippets in the handout

REFLECTIVE LEARNING

- example sheets and ticks
- execute algorithms by hand, and invent test cases to achieve full code coverage
- skeptical reading

§2.2 Proving correctness



Right from the beginning, and all through the course, we stress that the programmer's task is not just to write down a program, but that his main task is to give a formal proof that the program he proposes meets the equally formal functional specification.

Edsger Dijkstra, 1930—2002

What is an algorithm?



ChatGPT

An algorithm is a set of well-defined instructions or a step-by-step procedure designed to perform a specific task or solve a particular problem.



You

Are you an algorithm?



ChatGPT

Yes, I am a product of complex algorithms. Specifically, I'm powered by a type of algorithm known as a machine learning model.



You

If you are an algorithm, then what is the particular problem you solve?



ChatGPT

The overarching problem I solve is bridging the gap between human communication and machine understanding, making information and interaction more accessible through natural language processing.

§2.3 Computational complexity

Asymptotic complexity refers to how program costs grow with increasing inputs

Usually space or time, with time larger than the

Question: if we double our problem size, how much does our computation cost increase?

Insertion Sort

```
# let rec ins = function
| x, [] -> [x]
| x, y::ys ->
  if x <= y then
    x :: y :: ys
  else
    y :: ins (x, ys)

# let rec insort = function
| [] -> []
| x::xs -> ins (x, insort xs)
```

- Items from input are copied to the output
- Inserted in order, so the output is always sorted

Complexity is $O(n^2)$ comparisons vs the theoretical best case of $O(n \log n)$

We'll analyze our algorithm's running time on large problems, and pretend we're running on an idealized machine:

- we can create arrays as large as we'll need, in time proportional to array size
- any array element $x[i]$ can be accessed in constant time
- all numerical operations take constant time

```
1 def insert_sort(x):
2   for i in 1..(len(x)-1):
3     j = i - 1
4     while j >= 0 and x[j] > x[j+1]:
5       swap x[j] with x[j+1]
6       j = j - 1
```

let $n = \text{len}(x)$.

outer loop runs $n-1$ times

inner loop runs $\leq i$ times.

cost of inner loop $\leq k_1 i$ k_1 const.

full cost $\leq \sum_{i=1}^{n-1} (k_1 i + k_2)$ k_2 const
extra cost per outer loop

$$= \frac{1}{2} k_1 n(n-1) + k_2(n-1).$$

Why is this this style of complexity analysis OK?

- It's the conventional mathematical playground for this discipline
- It's often a good approximation to real performance, unless ...

Q. In what situations would this sort of analysis be a bad idea?

- ❖ Cache locality:
it's faster if our code has a small working set, so that most memory accesses hit the CPU's cache.
- ❖ Large constants:
complexity analysis just says "constant cost", it doesn't say what the constant is — and it may be huge.
- ❖ Small problems:
if n is small then there's no point in asymptotic analysis, and we should just benchmark.
- ❖ Arbitrarily large problems:
for an array of length n we need $\Theta(\log n)$ bits to even store a pointer, so memory access isn't really $O(1)$.