IA Algorithms 1 and 2

Dr Damon Wischik
What is an algorithm?

**What sort of activity is working on algorithms?**

- **C.S. THEORY**
  - Tight definitions (but trivial proofs).
  - Elegant API design.

- **ALGORITHMS**
  - Tough proofs of correctness.
  - Cunning implementations to get optimal performance.

- **MACHINE LEARNING**

- **MODELLING**

- **SYSTEMS**
SECTION 2

Sorting algorithms
§2.1 Insertion sort

Insertion Sort

- Items from input are copied to the output
- Inserted in order, so the output is always sorted
```python
def insert_sort(x):
    for i in 1..(len(x)-1):
        # assert x[0:i] is sorted
        j = i - 1
        while j >= 0 and x[j] > x[j+1]:
            swap x[j] with x[j+1]
            j = j - 1
        # assert x[0:i+1] is sorted
```

NOTATION ALERT
Indexes refer to positions between cells.
❖ Is this algorithm correct?
❖ What is its performance?
Can we do better?

```python
def insert_sort(x):
    for i in range(1, len(x) - 1):
        # assert x[0:i] is sorted
        j = i - 1
        while j >= 0 and x[j] > x[j + 1]:
            swap x[j] with x[j + 1]
            j = j - 1
        # assert x[0:i+1] is sorted
```
- lecture notes
- slides (uploaded the night before, and re-uploaded after the lecture)
- example sheets & ticks
- recordings
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?
### How to learn effectively

<table>
<thead>
<tr>
<th>PASSIVE LEARNING</th>
<th>ACTIVE LEARNING</th>
<th>REFLECTIVE LEARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>- attend lectures</td>
<td>- copy out any handwriting from lectures</td>
<td>- example sheets and ticks</td>
</tr>
<tr>
<td>- read code snippets, watch animations, see examples</td>
<td>- annotate the code snippets in the handout</td>
<td>- execute algorithms by hand, and invent test cases to achieve full code coverage</td>
</tr>
<tr>
<td>- read notes, watch videos</td>
<td></td>
<td>- skeptical reading</td>
</tr>
</tbody>
</table>
§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.
def insert_sort(x):
    for i in 1..(len(x)-1):
        # assert x[0:i] is sorted
        j = i - 1
        while j >= 0 and x[j] > x[j+1]:
            swap x[j] with x[j+1]
            j = j - 1
        # assert x[0:i+1] is sorted

Problem statement:
Given an array x, sort it in-place.

Outline of proof of correctness of insert-sort:

let P(i) be the predicate "The assertion on line 3 didn't fail at i".

Q(i) \[ P(i) \]

Lemma. \[ P(1) \] is true. \( \checkmark \)

Lemma. \[ P(i) \Rightarrow Q(i) \] for \( i \leq \text{len}(x)-1 \) — see Ex. Sheet 2.

Lemma. \[ Q(i) \Rightarrow P(i+1) \] for \( i \leq \text{len}(x)-1 \) \( \checkmark \) trivial.

By induction \( Q(n-1) \) is true, i.e. when \( i = n-1 \) then \( x[0:i+1] \) is sorted (at line 8).

This is the last line of code that's run, so \( x \) can't change subsequently.

Thus, when the alg. terminates, \( x \) is sorted.
§2.3 Computational complexity

Asymptotic complexity refers to how program costs grow with increasing inputs

Usually space or time, with time being larger than the space

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

Insertion Sort

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

```haskell
# 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)
```

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

```python
def insert_sort(x):
    for i in 1..(len(x)-1):
        j = i - 1
        while j >= 0 and x[j] > x[j+1]:
            swap x[j] with x[j+1]
            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 for outer loop

\[
= \frac{1}{2} k_1 n (n-1) + k_2 (n-1)
\]
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 \( O(1) \).

Cache locality:
if our code has a small working set, so that most memory accesses hit the CPU’s cache, it’s faster.

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 \( O(1) \).

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 \( O(1) \).