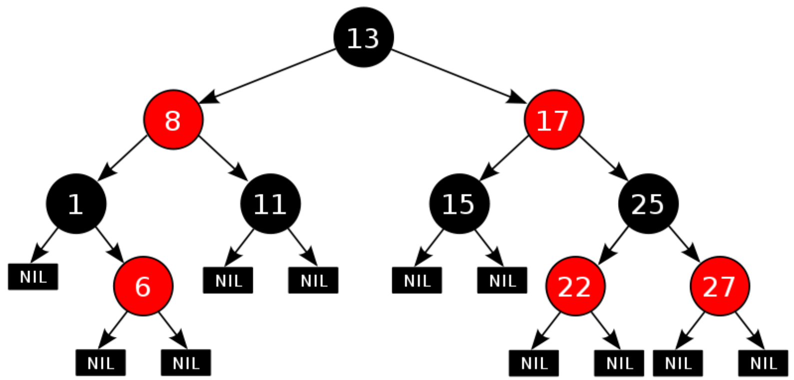
Algorithms I
Dr Robert Harle



CST Paper I

(IA NST CS, PPS CS and CST)

Easter 2009/10

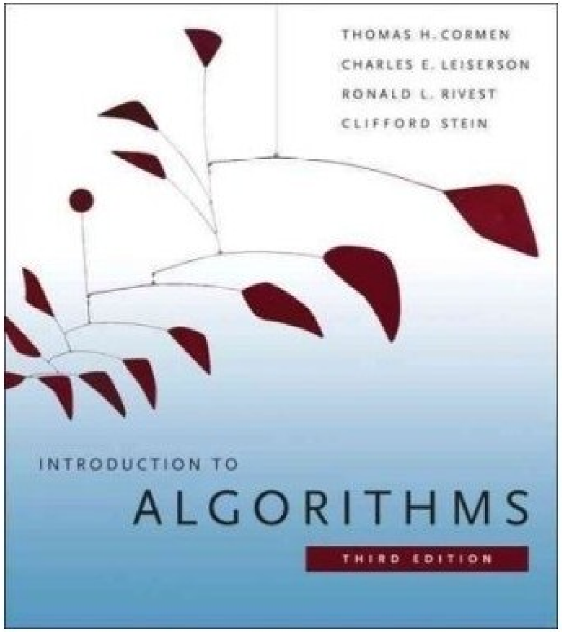
Algorithms I

- This course was developed by **Dr Frank Stajano**, who is on sabbatical this year
- I'm the “substitute teacher” :-)

- Dr Stajano's notes are very good: you have a copy of those as the handout. Those and the course textbook are probably all you need.
- However, I will post an annotated PDF of the notes I make in lectures as we go: check the course web page

- Three Parts
  - Sorting Algorithms
  - Algorithm Design
  - Data Structures

The CLR(S) Book



- Intro. To Algorithms
  - Cormen, Lieverson, Rivest, (Stein)

CLR

- The course is loosely based on this book
  - Definitely read the relevant bits of this book
  - Most libraries should have a copy
  - It contains some good exercises

Exercises

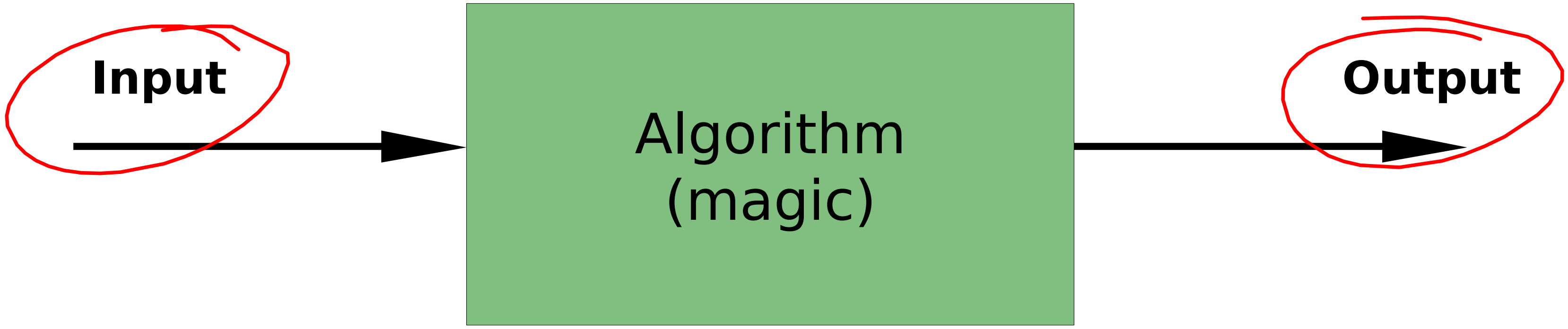
- There are some exercises dispersed throughout the notes
  - They aren't numbered
  - Most are just meant to be done as you read, rather than detailed problems

- There will be an exercise sheet available as a PDF on the course website that you may wish to use for supervisions.

Algorithms

- At its core, CS is really just about puzzle solving. But we aren't just interested in finding a solution (or “algorithm”), we're interested in finding the best solution given some definition of 'best'
- Everything else (programming, maths) is just a set of tools that turn out to be useful in supporting our puzzle solving.
- There is no “universal algorithm”; nor will there be.
  - But you can learn a lot from studying how to solve a variety of problems since many problems can be broken down into smaller problems to which established algorithms (or variants of) are appropriate

Algorithms Optimize Something



- We choose algorithms based on:
  - How soon they give us output (performance)
  - How much resource they use (space)
  - How good the output is (quality)
  - Combinations of the above

Example: Digital Cameras (JPEG)

- Digital cameras read in a load of pixels and have to convert them into a JPEG image
  - **Performance**: Need to do the conversion quickly so you can take another picture
  - **Space**: Need to do the conversion with minimal space overheads (to keep camera cost and size down)
  - **Quality**: Need to produce a small file that is still a good representation of the original data

Example: Search Engines

Pages: A B C D E F G H I J K L

Google: 1000 queries per second (2006)

14,000,000,000

140,000,000 MB

Index

GET	A	B	F	H			
A	G	D	K	I	J	B	D
FIRST	G	A					
THIS	E	F	I	G	A		
YEAR	C						

200,000 words in English

Google algorithms!

1. Page rank
   - better ordering
   - more money in
2. Mapreduce
   - use lots of cheap PCs to do the work

- Algorithms:
  - Look up the search term in the index
  - Optionally combine the results (AND, OR)
  - Arrange the results in some useful order

Part I: Sorting Algorithms

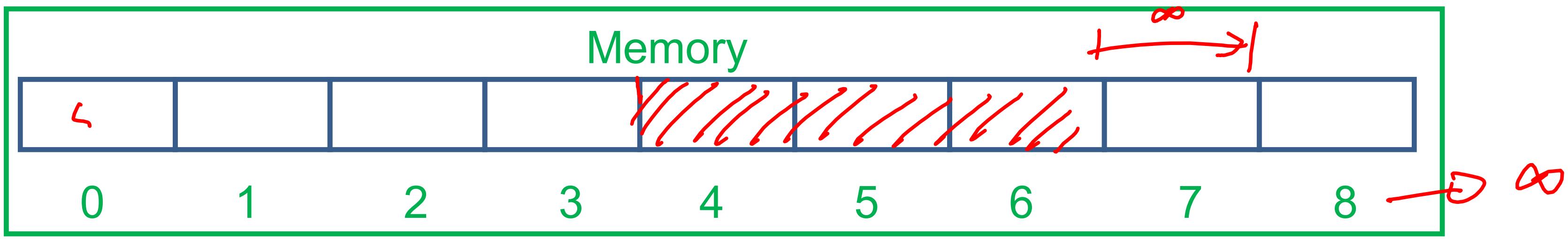
Why Sorting?

- There is an objective correct result
- <u>Many</u> sorting algorithms are available
  - Some really simple
  - Some more complex
- Sorting (and searching) are needed for most large-scale algorithms

- You have already met <u>some</u> of this in FoCS, but I'll recap anyway (it is revision time after all)
  - Plus you concentrated on sorting **lists** in FoCS: here we look at sorting **arrays**

Memory Model

- We'll use the simple model from OOP



- Key points:
  - Memory is addressed using numerical addresses and therefore random access
  - We will assume that we never run out of memory
  - We will not worry about the capacity of each memory slot (we'll assume any number can be represented in any slot)

Insertion Sort

sorted ← | → unsorted

3	6	4	5	1	2
3	6	4	5	1	2
3	4	6	5	1	2
3	4	5	6	1	2
1	3	4	5	6	2
1	2	3	4	5	6

Insertion Sort

```
def insertSort(a):
    '''BEHAVIOUR: Run the insertsort algorithm on the integer
    array a, sorting it in place.

    PRECONDITION: array a contains len(a) integer values.

    POSTCONDITION: array a contains the same integer values as before,
    but now they are sorted in ascending order.'''

    for k from 0 to len(a)-2:
        assert(the first k positions are already sorted)

        # Pick up item k+1 (call it a[j]) and let it sink to its correct place
        j = k+1
        while j > 0 and a[j-1] > a[j]:
            swap(a[j-1], a[j])
            j = j-1
```

constraints on input (if any)

"Contract" for what the alg will do

How 'good' is any algorithm?

- It's hard to put numbers to anything since the performance is presumably heavily dependent on the input

- As you know we usually study the limiting behaviour using the **asymptotic notation** you met in FoCS

Complexity Notations

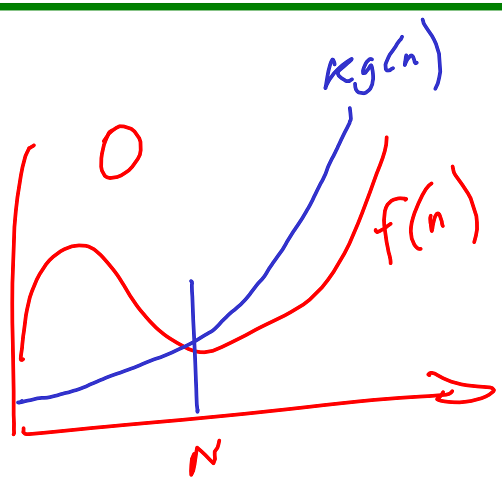
Big-O: $0 \le f(n) \le k.g(n)$

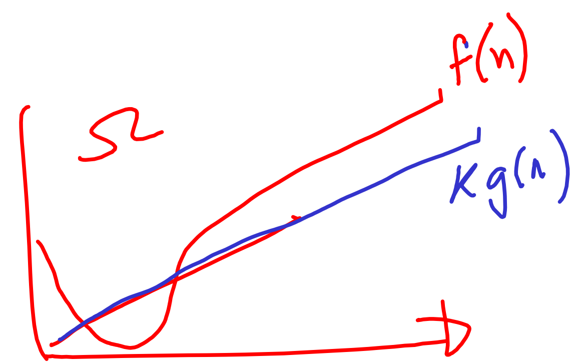
Θ: $0 \le k_1.g(n) \le f(n) \le k_2.g(n)$

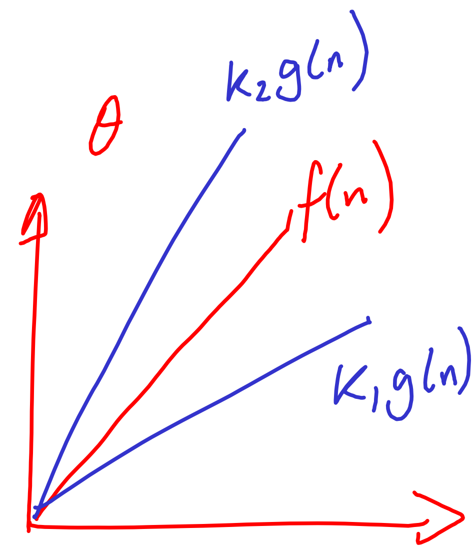
Ω: $0 \le k.g(n) \le f(n)$

For n>N

$K, k_1, k_2, N > 0$

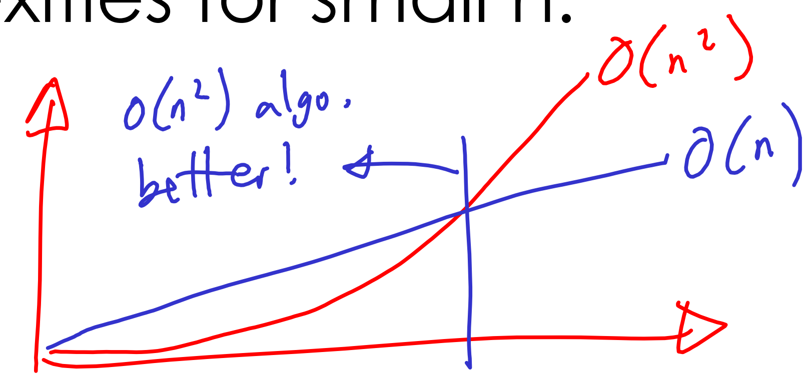






Notes

- $\log_a(x) = \log_b(x)/\log_b(a)$
  - So the base of any logarithm in g(n) is irrelevant "lg" or "log"
- The value of N above which the bound holds could be very big
  - i.e. Take care when comparing two complexities for small n.



Examples

- Show $(x+5)\lg(3x^2+7)$ is $O(x\lg x)$

$$(x+5)\lg(3x^2+7) \leq (x+5x)\lg(3x^2+7x^2) \qquad x \geq 1$$

$$\leq 6x\lg(10x^2)$$

$$6x\lg(10x^2) \leq 6x\lg(x^3) \qquad x \geq 10$$

$$= 18x\lg x$$

$$(x+5)\lg(3x^2+7) \leq 18x\lg x$$

$$O(x\lg x)$$

Examples

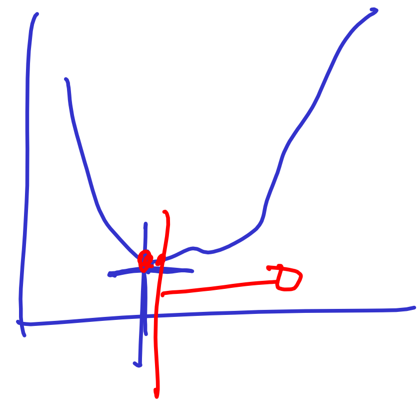
- Show n^3+20n is $\boxed{\Omega(n^2)}$

$$\boxed{n^3+20n \geq kn^2}$$

Defn

$$n + \frac{20}{n} \geq k$$

constant



5

Find minimum

$$\frac{d}{dn}\left(n+\frac{20}{n}\right) = 1 - \frac{20}{n^2} = 0$$

$$n^2 = 20 \qquad n = \sqrt{20} \approx 4.5$$

$$\boxed{N=5 \qquad k \leq 9}$$

$$5 + \frac{20}{5} = 9$$

Examples

- Show n^2-3n is $\Theta(n^2)$

Def$\underline{^n}$

$$\boxed{k_1 n^2 \le n^2 - 3n \le k_2 n^2}$$

$k_1, k_2 > 0$

$$k_1 \le 1 - \frac{3}{n} \le k_2$$

n	$1 - \frac{3}{n}$
1	-2
2	-0.5
3	0
4	$1/4$
⋮	⋮

$k_1 = 1/4$

$n \ge 4$

$k_2 \ge 1$

$\forall n > 1$

$\Theta(n^2)$

$k_1 = 1/4$ $N = 4$

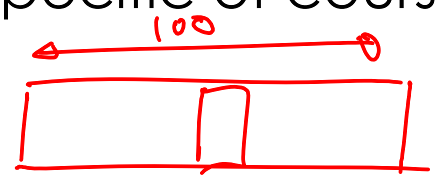
$k_2 = 1$

Relating to Running Time

- We assume:
  - Any memory access takes unit time $O(1)$
  - Any arithmetic takes unit time $O(1)$
- Thus the running time is linked to the number of operations the algorithm requires.
- Problem: this is often dependent on the input

Worst, Average and Amortized costs

- **Worst-case**
  - Analyse for the worst possible input. This gives you an upper bound for the performance.
- **Average-case**
  - Analyse for an 'average' input. Problem here is that the notion of average assumes some probability distribution of inputs, which we rarely have (and which is application specific of course).



- **Amortized analysis**
  - Sometimes we have a sequence of operations that occur: in this case we may *amortize* the total cost to run the sequence of operations so we get an average cost per operation. e.g. Garbage collection.