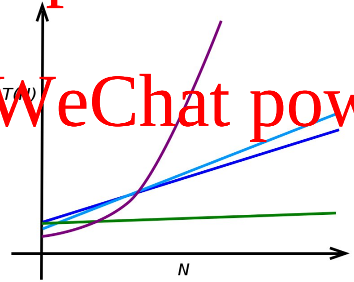
CO580 Algorithms

Dr Timothy Kimber

January 2018



Course Outline

The lecturer

- PhD in Computational Logic (Imperial)
- 5 years as Teaching Fellow/Senior Teaching Fellow
- Also teach Prolog to the MAC and Specialism classes

The structure

- 28 hours of interactive lectures (weeks 2–9)
- Sessions include unassessed group and individual exercises
- Two assessed exercises (one in Java) (10%)
- A 2-hour written examination *next term* (90%)

Books

- *Introduction to Algorithms*, Cormen et al., 3rd edn, 2009.
- *Algorithms*, Sedgewick & Wayne 4th edn, 2011.

Intended Learning Outcomes

At the end of this module YOU will be better able to ...

1. Communicate with other engineers about how to solve a computational problem*.
2. Organise and manage computational resources.
3. Deploy known solutions to common problems.
4. Create original solutions to problems using sound general approaches.
5. Design appropriate data structures.
6. Explain properties of a solution, existing or original, to customers.
7. Analyse performance of code using established engineering techniques and terminology.

*e.g. at an interview

Course Summary

This course: How To Write Good Programs

A good program ...

- Always gives some output (terminates)
- Always gives a correct output (sound)
- Gives an output for every possible input (complete)
- Uses as few resources as possible

Aims

This course is concerned with the resources consumed by your programs, principally space and time.

Questions To Answer

- How much time/space does a program use?
- What kind of program uses least time/space?

These questions are a bit harder to answer for time, but that is what we are usually most interested in, so that is where we will start.

An Algorithm

Example *(Sequence Search)*

Input: a sequence $L = \langle a_1, ..., a_N \rangle$ of N integers, and an integer k

Output: *True* if k is in L, *False* otherwise

Simplification: assume L is ordered

Procedure: SimpleSearch (Input: seq L, int k)


```
for each e in L
  if e == k
    return True
return False
```

Input Cases

When analysing an algorithm's performance it is essential to be clear what input cases are being considered. Most often this will be one of

- Best case (least possible time/space consumed)
- Worst case (greatest possible time/space consumed)
- Average case (see later)

Later, you will see how to combine these analyses to describe performance for any input.

Formal Analysis (Worst Case)

- The input has N elements
- The cost (time taken) for line i is represented by c_i.
- We know exactly how many times each instruction happens (only worst case considered).

Simple Search (Input: seq L and int k)

```
                                        Cost  Times
1 for each e in L                         c1    N+1
2   if e == k                             c2    N
3     return True                         c3    0
4 return False                            c4    1
```

Simple Search (Worst Case)

- So, the running time, or time complexity for a worst case input of size N is

$$T(N) = c_1 + Nc_1 + Nc_2 + c_4$$

- so,

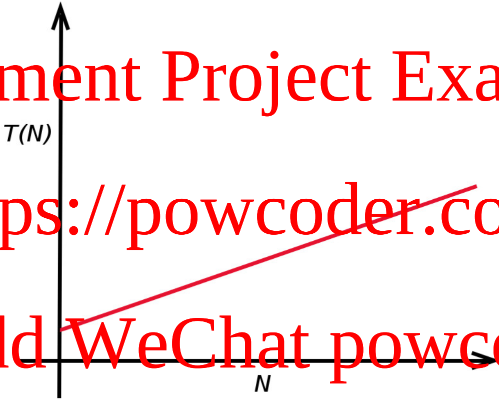
$$T(N) = (c_1 + c_4) + (c_1 + c_2)N$$

- so,

$$T(N) = aN + b$$

- There is a chunk of time a that is used for each element of L
- There is a chunk of time b that is used just once

Simple Search (Worst Case) Time Complexity



- Longer sequences take more time
- The increase is linear
- *a* and *b* will differ with language, hardware, load etc.

Comparing Algorithms

Given an input of size N, the worst case time complexity for a range of algorithms that all solve Problem X is:

- (A) $T(N) = a_1 N + a_2$
- (B) $T(N) = b_1 N$
- (C) $T(N) = c_1 N^2 + c_2 N + c_3$
- (D) $T(N) = d_1 N^3 + d_2$

where a_1, b_1, c_1 and d_1 are positive. In terms of time, which is:

- the best algorithm? (why?)
- the worst algorithm? (why?)

Highest Order Terms

For large N functions are dominated by their highest order N term

- Polylogarithmic functions include a term of the form $(\log_b N)^k$
- Any degree d positive polynomial grows faster than any polynomial of degree less than d, and any polylogarithm

Definition (Polynomial)

A polynomial of degree d (for $d \geq 1$) is a function $p(N)$ of the form

$$p(N) = a_0 + a_1 N + a_2 N^2 + \cdots + a_d N^d$$

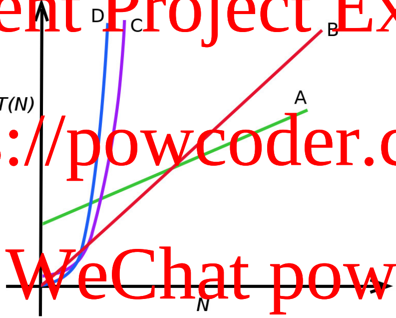
in which $a_d \neq 0$. The polynomial is asymptotically positive iff $a_d > 0$.

Exponential functions include a term of the form a^N

- If $a > 1$ then the function grows faster than any polynomial

Comparing Algorithms

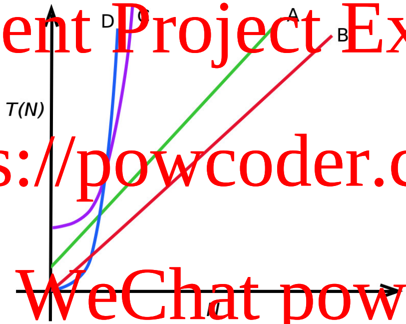
Depending on the constants, the time complexities might look like this ...



- Regardless of constant factors, D will take longest for large N
- "Large N" is usually small enough that we don't want C or D

Comparing Algorithms

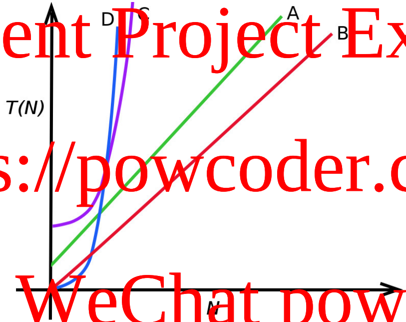
Or like this ...



- A and B are, in a sense, indistinguishable (constant factors)
- The value of large N, and if it exists, for A and B needs to be considered

Comparing Algorithms

So, we have clear(ish) goals



- Any "N^3 algorithm" is worse than any "N^2 algorithm"
- Any "N^2 algorithm" is worse than any "N algorithm"
- Unless we have big constants*, or small N
- *Normally, we don't