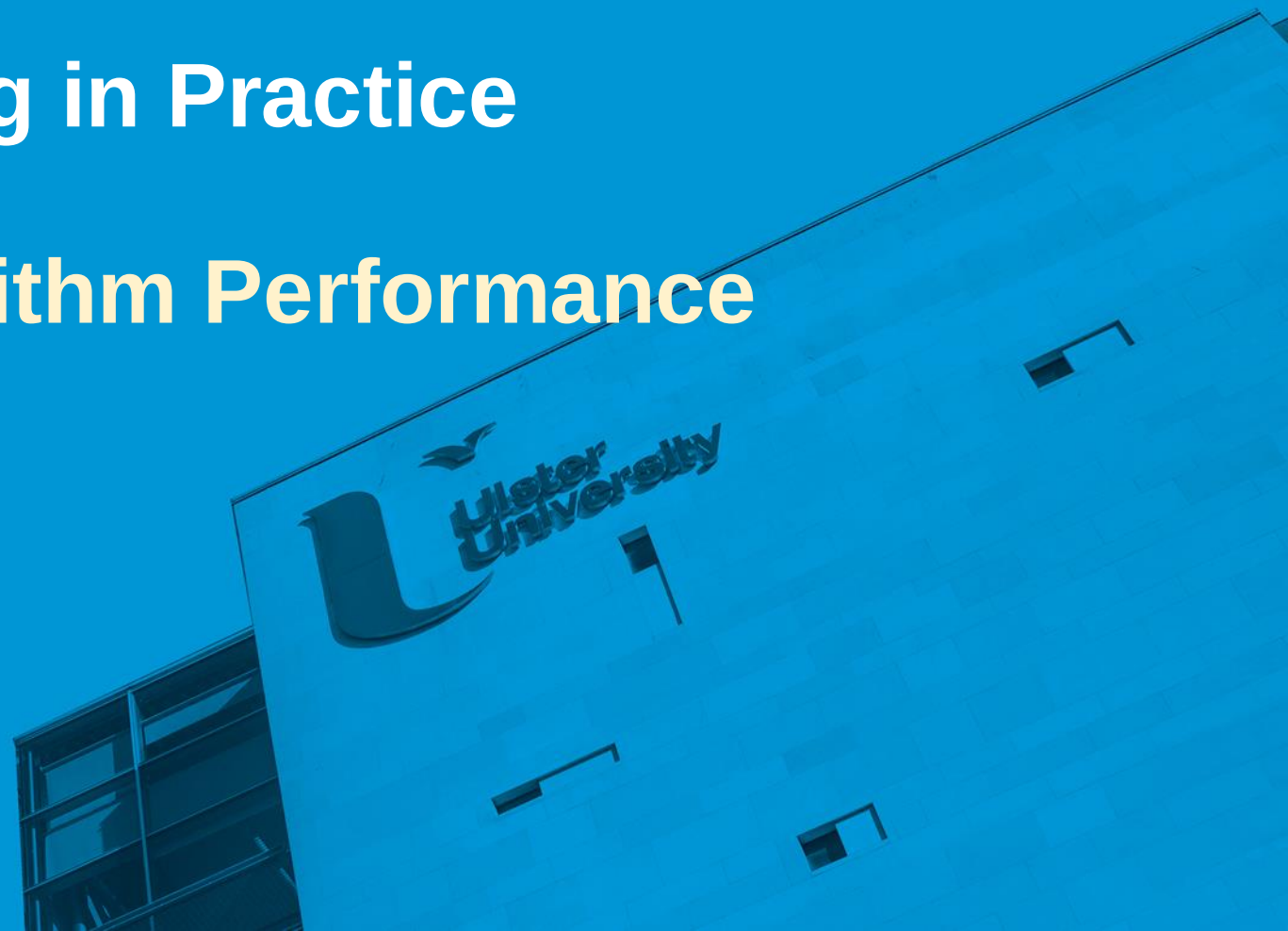
Ulster University

COM410 Programming in Practice

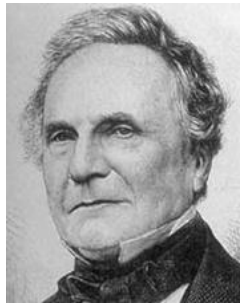
B1.1 Measuring Algorithm Performance

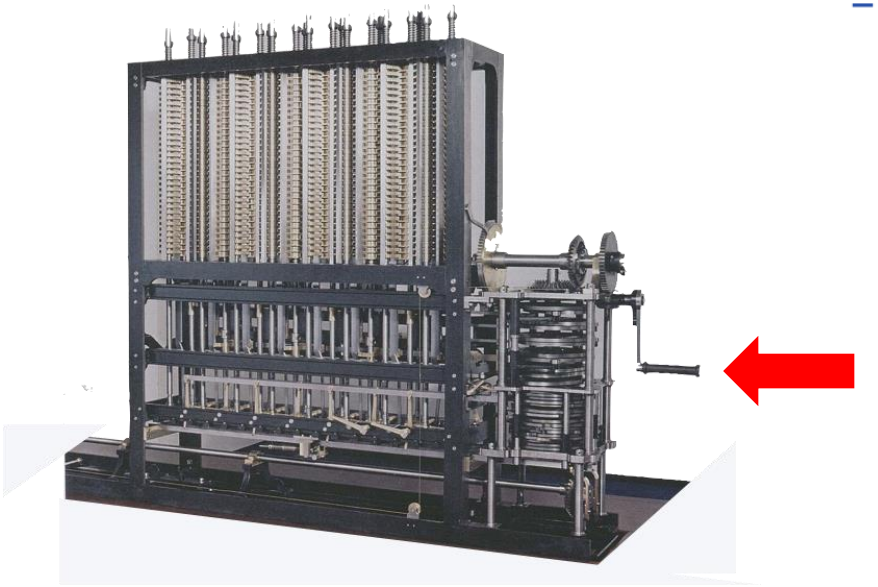


The Challenge (earliest days . . .)

“ 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





How many times do you have to turn the crank?

The Challenge (modern version . . .)

- *Will I be able to use my program to solve a large practical problem?*
- *How might I understand its performance characteristics in order to improve it?*

Why Study Program Performance?

1) To predict program behaviour
    - Will my program finish?
    - When will my program finish?
2) To compare algorithms and implementations
    - Will this change make my program faster?
    - How can I make my program faster?
3) To develop a basis for understanding the problem and developing new algorithms
    - Enables new technology

Example - Compute the Sum of Positive Integers

- Compute the sum **1 + 2 + . . . + *n*** for any positive integer ***n* > 0**
- Three possible algorithms for solving this problem

$$\sum_{i=1}^{n} i$$

```
// Algorithm A
long sum = 0;
for (long i=1; i<=n; i++) {
    sum = sum + i;
}
```

```
// Algorithm B
long sum = 0;
for (long i=1; i<= n; i++) {
   for (long j=1; j<=i; j++) {
       sum = sum + 1;
   }
}
```

```
// Algorithm C
long sum = 0;
sum = n * (n + 1) / 2;
```

Scenario

Compare algorithm performance

- Create a new Java project called **Analysis** and implement the class **`SumIntegers`** with methods **`sumA()`**, **`sumB()`** and **`sumC()`** as implementations of the algorithms presented on the previous slide. Each method takes an **`int`** parameter **`n`** and returns the **`long`** sum of all integers <= **`n`**.
- Call each method with parameter value 10000 and verify that they return the same result.
- Now, add code to time the execution of each method and print out the execution time in nanoseconds. Verify that there is a clear order in the execution speed of the algorithms

Analysis of Algorithms

- *How can we measure efficiency so that we can compare various approaches to solving a problem?*
  - Implementing several ideas before choosing one isn't practical / feasible
  - A program's execution time is dependent on computer and language used

    (better to measure an algorithm's efficiency before implementing it)

What is "best"?

- An algorithm has both time and space constraints (complexity)
  - Time complexity (time it takes to execute)
  - Space complexity (memory needed to execute)

A "best" algorithm might be the fastest one or the one that uses the least memory

Analysis of Algorithms

- Usually the *best* solution to a problem balances out various criteria (time, space, generality, programming effort, etc.)
- Focus will be on **time complexity** (usually more important than space complexity)
- An **inverse relationship** often exists **between** an algorithm's **time complexity** and its **space complexity**:
- If you revise an algorithm to save execution time, you will usually need more space
- If you reduce an algorithm's space requirements, it will likely require more time to execute
- The measure of complexity should be expressed in terms of the **size of the problem** (i.e. number of items an algorithm processes) enabling comparison of relative cost as a function of problem size

Analysis of Algorithms (Empirical)

- Empirical analysis may be used to compute the running time of an algorithm:

Run experiments

- Start with a moderate input size N
- Measure and record running time
- Double input size N
- Repeat
- Tabulate and plot results