Algorithms and their Applications
CS2004 (2020-2021)

Dr Mahir Arzoky

2.1 Foundations of Algorithm Analysis

Notices

Laboratory Sessions

- Two laboratory sessions
  - Laboratory 1
    - Mondays 13:00 to 15:00
  - Laboratory 2
    - Mondays 15:00 to 17:00
- Stick to your own laboratory session
- This week online only
- Blackboard Collaborate Ultra
- Moderated by staff and GTAs
- This week's laboratory sheets:
  - Worksheet 1 - Java Programming Environment (2020-2021)
  - Worksheet 2 - Simple Algorithms (2020-2021)

In-person Laboratory

- In-person laboratory sessions start week 3
- Consider if you don’t have a suitable workspace at home
- Advanced booking only
- There will be few staff and GTAs present in-person
- Bring your own laptops
  - There are few desktop machines available
  - University has laptops that can be loaned
- Health and safety
  - 2m social distancing
  - There are wipe and anti-bacterial gel stations…
  - Wipe the desk area and keyboard, mouse etc…
  - You are encouraged to wear face covering, if you are able to – I will!

The Booking System

- Very Limited number of spaces due to social distancing
- Access the booking system here:
  - https://ttbookings.brunel.ac.uk
- Every Wednesday afternoon you can book your session for the following week
- First come first served basis
- Sessions will close to bookings when it reaches capacity
- You will be able to cancel your booking if you no longer need it

Discussion boards

- ❑ On Teams
  - ❑ Link is released with the release of the worksheet
- ❑ There is a discussion thread for each Laboratory worksheet
- ❑ Staff and GTAs will be moderating the discussion boards
  - ❑ Mainly replying back during the laboratory sessions
- ❑ Feel free to help each other!
- ❑ Discussion only related to the worksheet
  - ❑ There is a ‘General’ channel for general questions with the module
- ❑ Please do NOT post any solutions on the thread!

The Story So Far…

- This module involves the analysis, implementation and development of (potentially complex) algorithms
- Last time we looked at:
  - How this module is organised
  - Some background and concepts, e.g. algorithms, computation, etc…

Foundations of Algorithm Analysis

- In this lecture we are going to cover some of the basic concepts that will be used and needed throughout this module:
  - Running time
  - Pseudo-Code
  - Counting primitive operations

How do you compare algorithms?

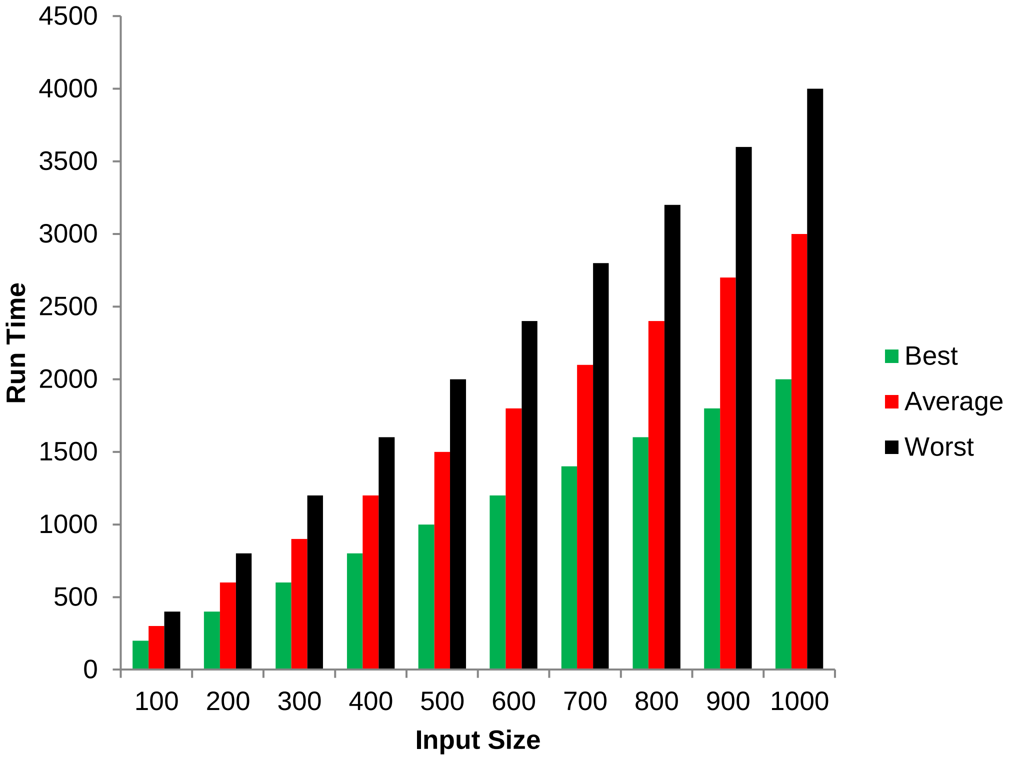
- Imagine you have two algorithms A and B
  - Both does the same thing
  - Which one to choose?
- How would you compare them?
  - Low level: Code of A vs. Code of B?
  - High level: Algorithm A vs. Algorithm B?
- What are the criteria?
  - They are compared in term of efficiency
  - Time it takes and resources consumed

Run Time – Part 1

- The running time of an algorithm varies with the input and typically grows with the input size
- An algorithm may run faster on certain data sets than on others
- Hence it would have the following run times:
  - Best case
  - Worst case
  - Average case

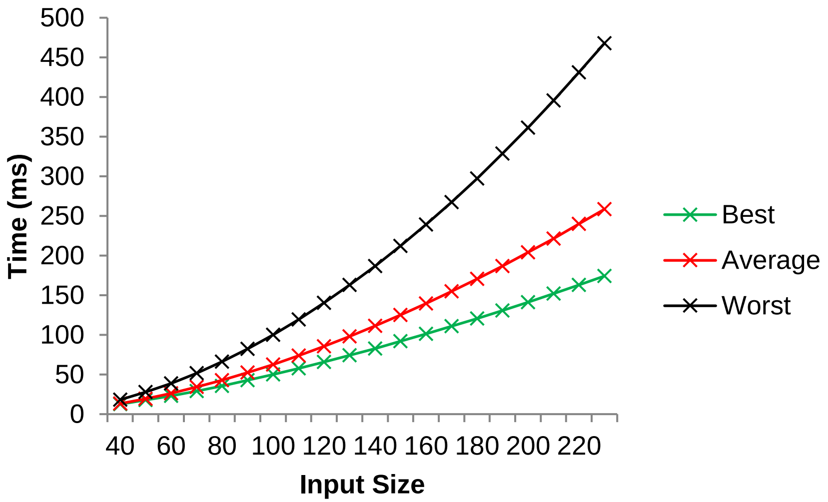
Run Time – Part 2

- The average case can be difficult to determine
- The best case is not very informative
- The worst case running time
  - Our main focus



Experimental Studies

- ❑ Write a program implementing the algorithm
- ❑ Run the program with inputs of varying size and composition, multiple times...
- ❑ Use a method such as `System.currentTimeMillis()` to get an accurate measure of the actual running time
- ❑ Plot the results
- ❑ Can you see any problems with this approach?



Limitations of Experimental Studies

- It is sometimes difficult to implement the algorithm
- Results may only be indicative of the running time for some inputs, but not for all possible inputs
- One algorithm can perform better than the rest for some inputs. For other inputs another algorithm could be better
- In order to compare two algorithms, the same hardware and software environments must be used

- **However**, we will still measure the run time of some of our experiments for practice purposes…

So What Can We Do?

- ❑ Can we calculate/estimate the running time without experiments?
- ❑ Is there an alternative to implementing and running a large number of experiments?
- ❑ Solution: **Asymptotic Analysis**!

Asymptotic Analysis

- What is **Asymptotic Analysis**?
- The technique uses a high-level description of the algorithm instead of an implementation
- We evaluate the performance of an algorithm in terms of input size.
- We calculate how does the time taken by an algorithm increases with the input size.

Estimating “Running Time”

- Write down an algorithm
  - Using Pseudo-Code
- In terms of a set of primitive operations
  - Count the number of steps
  - Consider worst case input
- Bound or “estimate” the running time
- This leads onto the Big-Oh (Big-O) notation for computational complexity
  - We will cover this in a later lecture

T(n) and O(n)

- We estimate the running time/computation of an algorithm
- We refer to this resultant formulae as $T(n)$ **where n is the size of the input**
- If there is more than one input we might have $T(n,m)$ (etc…) where n and m are the sizes of the inputs
- We can use $T(n)$ to compute a **very** important property called the Big-O $\mathrm{O}(n)$ - we will learn more in a later lecture

Pseudo-Code – Part 1

- Not an actual programming language…
- Pseudo-Code is a cross between a programming language and written text
- It is used to represent algorithms in a programming language independent manner
- We often specify the **input** and **output** and **number the lines**
- The Pseudo-Code I will now describe is an adaptation of a standard…
  - Once you can read one type of Pseudo-Code it is easy to follow any variations

Pseudo-Code – Part 2

We give all algorithms a number

A meaningful name

```
Algorithm 1. NameOfAlgorithm(x,y,z)
Input:
1) …
2) …
3) …
Output:
```

List all of the inputs as parameters (Give them meaningful names – unlike this example!!)

Define the type of what the input(s) should be

Number the lines of the algorithm

State the output of the algorithm

Pseudo-Code – Part 3

- We use the normal concepts of variables, loops and conditional statements
- Often we use mathematical notation, e.g. sets, equations, etc…
- We do not call functions unless we describe the Pseudo-Code for them or describe what they do
  - We would NOT say: `X = Y.toLowerCase();`
  - However we could say: `Let X equal the lower case version of Y`
  - Unless we have previous defined the method, function, sub-routine, algorithm, etc…

Pseudo-Code – Part 4

- Pseudo-Code more details
  - Control flow
    - `If … Then … [Else …] End If`
    - `While … End While`
    - `Repeat … Until …`
    - `For …End For …`
  - Indentation replaces braces [as in Python]
  - Method/algorithm declaration
    - `Algorithm` *number name* (*arguments/parameters*)
    - `Input …`
    - `...`
    - `Output …`

Pseudo-Code – Part 5

- Expressions
  - We use a single = sign for both assignment and equality testing
  - `Let x = 0`
  - `If x = y Then...`
  - The context differentiates the meaning of the usage…
- n^2 superscripts, subscripts n_i and other mathematical formatting is allowed
- We often write the algorithm in a different font such as `Courier New`
- Comments allowed

Pseudo-Code – Variables

We have defined an input variable x

```
Algorithm 2. UselessAlgorithm(x)
Input: x is an integer
1) Let y = x + 1
Output: y, the value of x incremented by 1
```

We define and set a new variable

State that y is the output

Pseudo-Code – `If` Statements

```
Algorithm 3. Sign(x)
Input: x is a number
1) If x > 0 Then
2)    Let Res = 1
3) Else
4)      Let Res = -1
5) End If
6) If x = 0 Then Res = 0
Output: Res, +1 if +ve, -1 if -ve or
             0 is zero
```

We do not always need an `Else` part or an `End If`

Pseudo-Code – `For` Loops

```
Algorithm 4. Sum(n)
Input: n is an Integer > 0
1) Let ResSum = 0
2) For i = 1 to n
3)     Let ResSum = ResSum + i
4) End For
Output: ResSum
```

Sum up the whole numbers from 1 to `n`

Pseudo-Code – Example

```
Algorithm 5. ArrayMax(Arr)
Input: A 1-D numerical array Arr of size n>0
1) Let CurrentMax = a_0
2) For i = 1 to n-1
3)   If a_i > CurrentMax Then CurrentMax = a_i
4) End For
Output: CurrentMax, the largest value in Arr
```

Primitive Operations – Part 1

- Basic computations performed by an algorithm
- Identifiable in Pseudo-Code
- Largely independent from the programming language
- Examples:
  - Evaluating an expression (`x>y`?)
  - Assigning a value to a variable (`x=0`)
  - Indexing into an array (for `A[0]` or `A[i]` we might use the mathematical notation a_0 or a_i)
  - Calling a method
  - Returning from a method

Primitive Operations – Part 2

- Consider the following lines of Pseudo-Code:

```
Let a be an array of size n where a_i=0
```

n for creating the array and n for setting to zero $T(n)$= 2n

```
Let x = 10
```

One operation (set) $T(n)$= 1

```
Let y = x
```

One read and one write $T(n)$= 2

```
Let z = x + y
```

Two reads, one arithmetic and one write/set $T(n)$= 4

```
For i = 1 to n
    Let a_i = a_i + x + y + 10
End For
```

(n operations)

(=9 – repeated n times for `For` loop $T(n)$= 9n)
(1 [2] for the write, 5 [4] reads, 3 operators)

No count – indicates the end of the loop

Primitive Operations – Part 3

- The most difficult part of the process is getting all of the nested loops correct
- We will see that later on - if you count 5 primitive operations for a line instead of 6 it does not really matter
  - However if you get the loops wrong, the error will matter
- This is why getting the correct level of indentation is important

Counting Primitive Operations – 1

- By inspecting the Pseudo-Code, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size (n)

Operations = 2

```
Algorithm 5. ArrayMax(Arr)
Input: A 1-D numerical array Arr of size n>0
1) Let CurrentMax = a0
2) For i = 1 to n-1
3)    If ai > CurrentMax Then CurrentMax = ai
4) End For
Output: CurrentMax, the largest value in Arr
```

Operations = n-1

Operations = 7(n-1)

Operations = 1 (optional)

Total Operations $T(n)$ = 2+(n-1)+7(n-1) = 2+8n-8 = 8n-6

Counting Primitive Operations – 2

```
Algorithm 6. ArrayMax(Arr)
Input: A 2-D numerical array Arr of size n rows
       by m columns
```

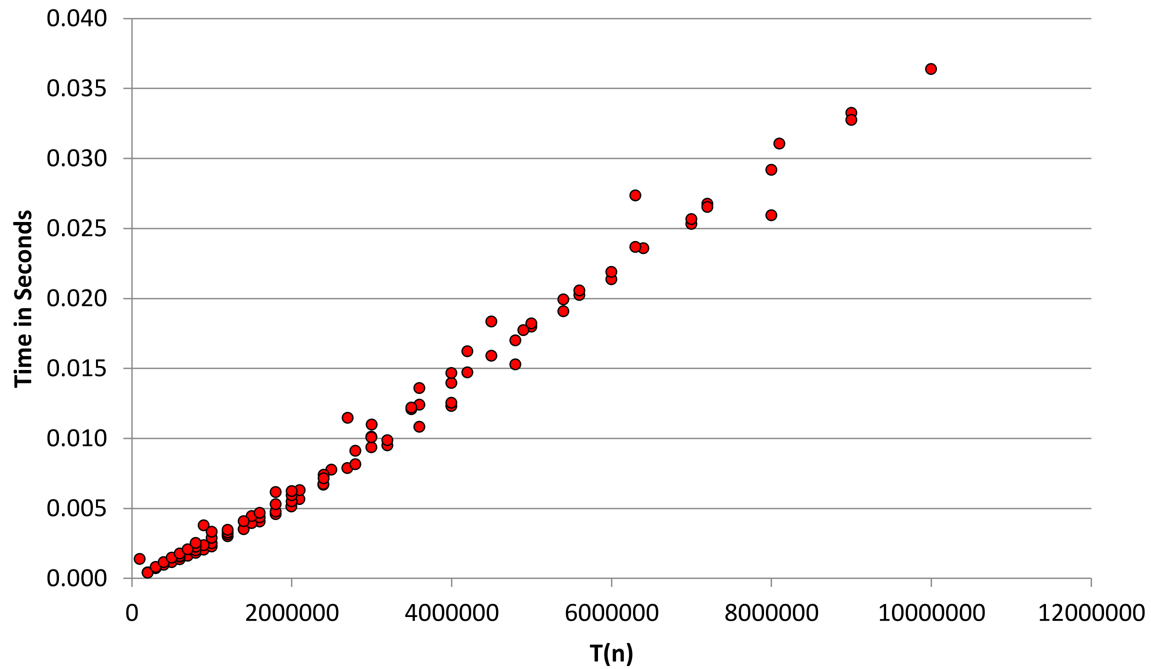
Two sizes of input!!!

```
1) Let CurrentMax = a00                                   (2)
2) For i = 0 to n-1                                       (n)
3)     For j = 0 to m-1                                   (m×(n))
4)         If aij > CurrentMax Then CurrentMax = aij      ((9×(m×(n)))
5)     End For
6) End For
Output: CurrentMax, the largest value in Arr
```

$$T(n,m) = 2+n+mn+9mn = 10mn+n+2$$

Counting Primitive Operations – 3

- Implemented the `ArrayMax 2-D` algorithm (in the previous slide) in `MatLab` for a number of n and m
  - n,m = 1,000 to 10,000 in steps of 1,000
- Recorded the runtime and computed $T(n,m)$ using the formulae from the previous slide



- A correlation of 0.994

Next Topic

- Lecture
  - Some mathematical background
- Laboratory
  - We will be implementing some simple algorithms
  - Note there are two worksheets
  - This will require some mathematical knowledge