ALGORITHMS AND BIG-O NOTATION

DR IAN CORNELIUS

HELLO

- Learning Objectives
    1. Understand what an algorithm is and the purpose of Big-O notation
    2. Demonstrate the ability to use algorithms and Big-O notation

INTRODUCTION TO ALGORITHMS

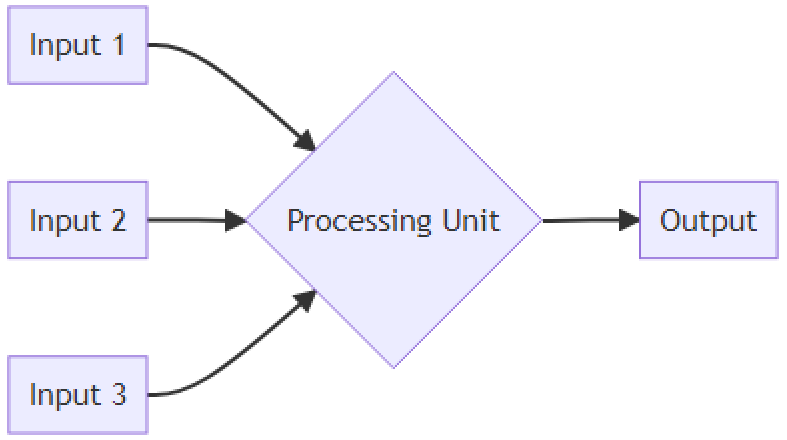
- An algorithm is a procedure or formula to solve a problem
  - they are based on a performing a sequence of steps/actions
- This could be a function that does a particular job each time it is called
- An algorithm will have a well-defined set of steps and provide an output
  - eventually, after `n` steps it will terminate
- They can be written in either pseudocode or displayed as a flow chart

COMPUTER PROGRAM VS. ALGORITHMS

- An algorithm is a self-contained, step-by-step set of operations
  - they are performed to solve a specific problem or a class of problems
- A computer program is a sequence of instructions
  - they comply to the rules of a specific programming language
  - they are written to perform a specific task with a computer
  - may contain multiple algorithms

DEFINING AN ALGORITHM

- Algorithms consist of:
  - **a problem**: defined to be a real-world problem
  - **an algorithm**: a defined step-by-step process designed for the problem
  - **inputs**: algorithm is provided necessary and desired inputs
  - **a processing unit**: inputs are processed to produce a desired output
  - **output**: the outcome of the algorithm



EXAMPLE OF AN ALGORITHM

- Sorting cards by their respective colour
    1. Pick up all the cards
    2. Pick a card from your hand and look at the colour
    3. If there is a pile of cards with that colour already, add it to that pile
    4. If there is not a pile of cards with that colour, make a new pile for this colour
    5. If there is a card still in your hand, go back to the second step
    6. If there are no cards in your hand, then the cards are sorted, and you are done
- The above example is readable by a human
- For a machine to understand it would require functions and nested `if` statements and control statements

PSEUDOCODE

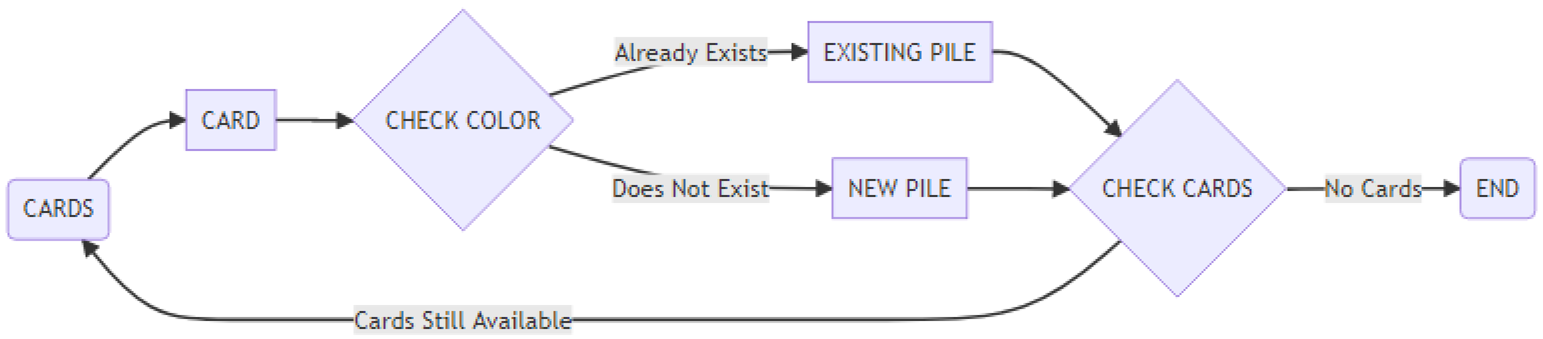
- Pseudocode is a language used to describe algorithms
- Written in a high-level method so anyone can read it and understand it
- Language similar to programming code will be used
  - therefore, it feels natural whilst developers read it

PSEUDOCODE FOR CARD SORTING

```
SORT_CARDS(CARDS)
  DATABASE COLORS = EMPTY
  FOR i <- 0 to length(CARDS)
    IF CARDS[i] == COLORS[i]
      COLORS[i] <- COLORS[i] + 1
    ELSE
      COLORS[i] = 1
```

FLOW DIAGRAM OF CARD SORTING

- Flow diagrams are great for visualising the steps of an algorithm
- Easy to visualise what is happening at each step



CHARACTERISTICS OF AN ALGORITHM

- An algorithm will consist of the following characteristics:
  - some sort of input
  - a collection of results, known as an output
  - instructions should be unambiguous and easy to understand
  - the algorithm should be finite and conclude to something
  - it should be effective
  - the algorithm should language agnostic

WRITING AN ALGORITHM

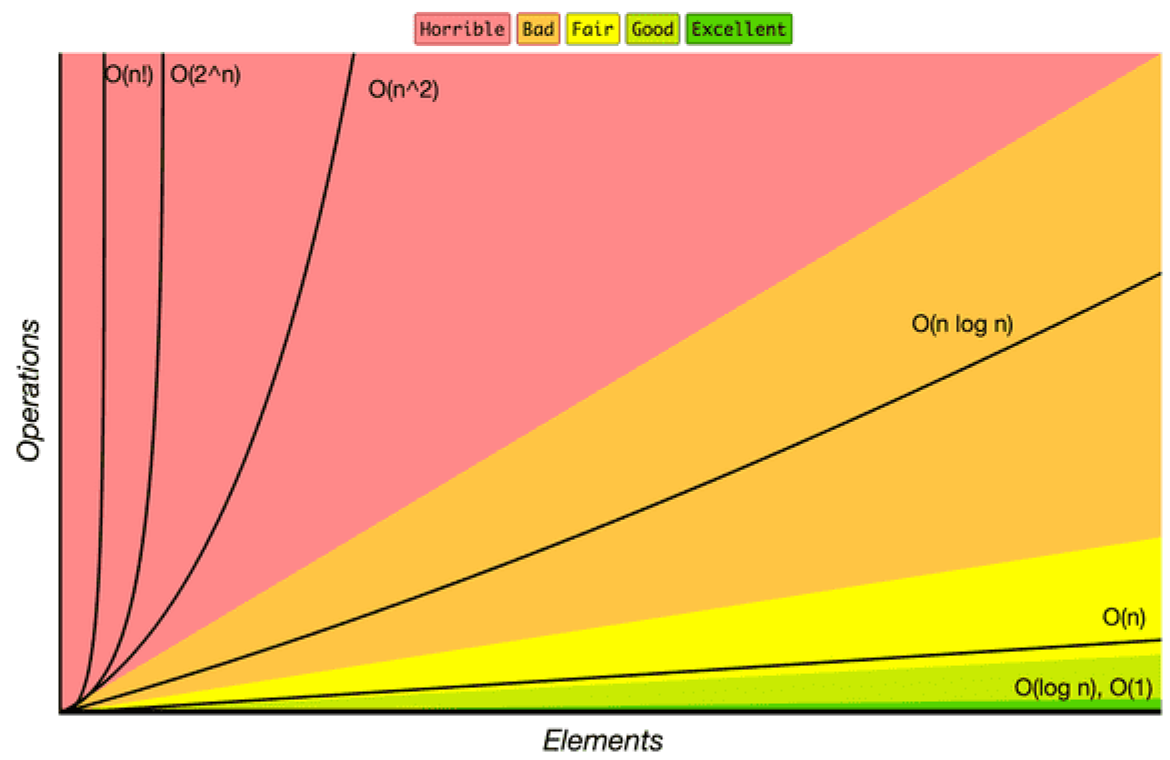
- When writing an algorithm, keep these ideologies in mind:
  - an algorithm is a step-by-step process
  - try and go-back to a step if a loop or condition fails
  - jump between statements if certain conditions are met
  - use the `break` keyword to stop and terminate the process when the condition is met

BIG-O NOTATION

- Used to describe the performance or complexity of an algorithm
  - describes the worst-case scenario
  - also describes the execution time or space used
    - i.e. memory or disk usage

GROWTH RATE OF COMPLEXITY

O	Complexity	Growth Rate
O(1)	constant	fast
O(log n)	logarithmic	
O(n)	linear time	
O(n log n)	log linear	
$O(n^2)$	quadratic	
$O(n^3)$	cubic	
$O(2^n)$	exponential	
O(n!)	factorial	slow



EXAMPLES OF BIG-O NOTATION (1)

O(1)

- Will always execute in the same time (or space)

```
def equal_to_one(_list):
    if _list[0] == 1:
        return True
```

O(N)

- Performance will grow linearly and in direct proportion of input data

```
def contains_number(_list, _number):
    for x in _list:
        if x == _number:
            return True
        else:
            return False
```

EXAMPLES OF BIG-O NOTATION (2)

$O(N^2)$

- Performance is directly proportional to the squared size of the input data
- Most common with algorithms that have nested iterations
  - deeper nested iterations will result in $O(N^3)$, $O(N^4)$ etc.

```
def contains_duplicates(_list):
    for i in range(len(_list)):
        for j in range (len(_list)):
            if i == j:
                continue
            if list[i] == list[j]:
                return True
    return False
```

EXAMPLES OF BIG-O NOTATION (3)

$O(2^N)$

- Denotes an algorithm where the growth doubles with each addition to the input
  - growth curve is considered to be exponential

```
def fibonacci(number):
    if number <= 1: return number
    return fibonacci(number - 2) + fibonacci(number - 1)
```

CALCULATING THE BIG-O VALUE OF AN ALGORITHM (1)

1. Break the algorithm into individual operations
2. Calculation the Big-O of each operation
3. Add up the Big-O of each operation together
4. Remove the constants
5. Find the highest-order term
    - this will be what we consider to be the Big-O of the algorithm

CALCULATING THE BIG-O VALUE OF AN ALGORITHM (2)

INSTRUCTIONS

1. Break the algorithm into individual operations
2. Calculation the Big-O of each operation
3. Add up the Big-O of each operation together
4. Remove the constants
5. Find the highest-order term
    - this will be what we consider to be the Big-O of the algorithm

```
def add(x, y):
    total = x + y
    return total
```

GOODBYE

- Questions?
  - Post them in the **Community Page** on Aula
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk